FERNANDO GONÇALVES DE ALMEIDA NETO

LOW-COMPLEXITY ADAPTIVE FILTERING

São Paulo
2015

FERNANDO GONÇALVES DE ALMEIDA NETO

LOW-COMPLEXITY ADAPTIVE FILTERING

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

São Paulo
2015

FERNANDO GONÇALVES DE ALMEIDA NETO

LOW-COMPLEXITY ADAPTIVE FILTERING

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Doutor em Engenharia Elétrica.

Área de Concentração:

Sistemas Eletrônicos

Orientador:

Vítor Heloiz Nascimento

São Paulo
2015

ACKNOWLEDGMENTS

I would like to thank my advisor, Vítor Heloiz Nascimento, for his inspirational and timely advice and constant encouragement over the last several years.

I also would like to thank Rodrigo de Lamare and Yury Zakharov for all the support during my PhD sandwich year at York.

Finally, I would to thank my friends Luiz Chamon and Wilder Lopes, and my beloved wife Amanda de Paula for possibly the best years of my academic life.

RESUMO

Neste texto são propostos algoritmos de filtragem adaptativa de baixo custo computacional para o processamento de sinais lineares no sentido amplo e para *beamforming*.

Novas técnicas de filtragem adaptativa com baixo custo computacional são desenvolvidas para o processamento de sinais lineares no sentido amplo, representados por números complexos ou por quaternions. Os algoritmos propostos evitam a redundância de estatísticas de segunda ordem na matriz de autocorrelação, o que é obtido por meio da substituição do vetor de dados original por um vetor de dados real contendo as mesmas informações. Dessa forma, evitam-se muitas operações entre números complexos (ou entre quaternions), que são substituídas por operações entre reais e números complexos (ou entre reais e quaternions), de menor custo computacional. Análises na média e na variância para qualquer algoritmo de quaternions baseados na técnica *least-mean squares* (LMS) são desenvolvidas. Também é obtido o algoritmo de quaternions baseado no LMS e com vetor de entrada real de mais rápida convergência.

Uma nova versão estável e de baixo custo computacional do algoritmo *recursive least squares* (RLS) amplamente linear também é desenvolvida neste texto. A técnica é modificada para usar o método do *dichotomous coordinate descent* (DCD), resultando em uma abordagem de custo computacional linear em relação ao comprimento N do vetor de entrada (enquanto o algoritmo original possui custo computacional quadrático em N).

Para aplicações em *beamforming*, são desenvolvidas novas técnicas baseadas no algoritmo *adaptive re-weighting homotopy*. As novas técnicas são aplicadas para *arrays* em que o número de fontes é menor do que o número de sensores, tal que a matriz de autocorrelação se torna mal-condicionada. O algoritmo DCD é usado para obter uma redução adicional do custo computacional.

Palavras-Chave – Filtros adaptativos de baixo custo computacional, processamento de sinais lineares no sentido amplo, *dichotomous coordinate descent*, *adaptive re-weighting homotopy*, *beamforming*

ABSTRACT

In this text, low-cost adaptive filtering techniques are proposed for widely-linear processing and beamforming applications.

New reduced-complexity versions of widely-linear adaptive filters are proposed for complex and quaternion processing. The low-cost techniques avoid redundant second-order statistics in the autocorrelation matrix, which is obtained replacing the original widely-linear data vector by a real vector with the same information. Using this approach, many complex-complex (or quaternion-quaternion) operations are substituted by less costly real-complex (or real-quaternion) computations in the algorithms. An analysis in the mean and in the variance is performed for quaternion-based techniques, suitable for any quaternion least-mean squares (LMS) algorithm. The fastest-converging widely-linear quaternion LMS algorithm with real-valued input is obtained.

For complex-valued processing, a low-cost and stable version of the widely-linear recursive least-squares (RLS) algorithm is also developed. The widely-linear RLS technique is modified to apply the dichotomous coordinate descent (DCD) method, which leads to an algorithm with computational complexity linear on the data vector length N (in opposition to the original WL technique, for which the complexity is quadratic in N).

New complex-valued techniques based on the adaptive re-weighting homotopy algorithm are developed for beamforming. The algorithms are applied to sensor arrays in which the number of interferer sources is less than the number of sensors, so that the autocorrelation matrix is ill-conditioned. DCD iterations are applied to further reduce the computational complexity.

Keywords – Low-cost adaptive filtering, widely-linear processing, dichotomous coordinate descent, adaptive re-weighting homotopy, beamforming.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

1 4-Ch-LMS = Four channel least-mean squares algorithm

2 ARH = Adaptive re-weighting homotopy algorithm

3 BPSK = Binary phase-shift keying

4 C-ARH = Complex adaptive re-weighting homotopy algorithm

5 CH = Complex homotopy algorithm

6 DCD = Dichotomous coordinate descent algorithm

7 DCD-It-C-ARH = Iterative complex adaptive re-weighting homotopy algorithm using dichotomous coordinate descent iterations

8 DCD-It-MC-C-ARH = Iterative multi-candidate complex adaptive re-weighting homotopy algorithm using dichotomous coordinate descent iterations

9 DCD-RLS = Dichotomous coordinate descent recursive least-squares algorithm

10 DCD-WL-RLS = Dichotomous coordinate descent widely-linear recursive least squares algorithm

11 DOA = Direction of arrival

12 EMSE = Excess mean-square error

13 FTF = Fast transversal filter

14 GLC = General-linear-combination-based robust Capon beamformer

15 It-C-ARH = Iterative complex adaptive re-weighting homotopy algorithm

16 It-MC-C-ARH = Iterative multi-candidate complex adaptive re-weighting homotopy algorithm

17 iQLMS = Quaternion least-mean squares algorithm obtained with the i-gradient

18 LMS = Least-mean squares algorithm

LS = Least-squares

MC = Multi-candidate

MC-C-ARH = Multi-candidate complex adaptive re-weighting homotopy algorithm

MSD = Mean-square deviation

MSE = Mean-square error

MVDR = Minimum variance distortionless response

NLMS = Normalized least-mean squares algorithm

PAM = Pulse amplitude modulation

PSK = Phase-shift keying

QAM = Quadrature-amplitude modulation

QLMS = Quaternion least-mean squares algorithm

QRLS = Quaternion recursive least-squares algorithm

RC = Reduced-complexity

RCB = Robust Capon Beamformer

RC-WL-iQLMS = Reduced-complexity widely-linear quaternion least-mean squares algorithm obtained with the i-gradient

RC-WL-LMS = Reduced-complexity widely-linear least-mean squares algorithm

RC-WL-QLMS = Reduced-complexity widely-linear quaternion least-mean squares algorithm

RC-WL-RLS = Reduced-complexity widely-linear recursive least-squares algorithm

RLS = Recursive least squares

SINR = Signal-to-interference-plus-noise ratio

SL = Strictly-linear

SNR = Signal-to-noise ratio

ULA = Uniform linear array

VL = Adaptive beamforming with variable loading

WL = Widely-linear

WL-iQLMS = Widely-linear quaternion least-mean squares algorithm obtained with the i-gradient

WL-LMS = Widely-linear least-mean squares algorithm

WL-QLMS = Widely-linear quaternion least-mean squares algorithm

WL-QRLS = Widely-linear quaternion recursive least-squares algorithm

WL-RLS = Widely-linear recursive least-squares algorithm

CONTENTS

1 INTRODUCTION

Signal processing techniques deal with restrictions, imposed by hardware and software limitations. These restrictions are strongly related to energy consumption and processing speed, which, in general, are antagonist variables. Since for many applications low energy consumption and fast processing are required, modern engineering is always seeking for ingenious alternatives to obtain a reasonable trade off, trying to satisfy them both. For this purpose, one can improve hardware and/or the algorithms applied to signal processing. In this dissertation, we focus on the improvement of algorithms, and we propose new alternatives to reduce the computational cost of adaptive filters.

In the first part of this text, we focus on the development of low-cost widely-linear (WL) adaptive filters [1, 2], which are applied to signals represented by complex numbers and quaternions [3]. WL algorithms were first proposed in the context of complex-valued estimation, to improve the mean-square error (MSE). They are applied in cases where the standard techniques are not able to extract full second-order information from the input data [1, 2, 4]. The standard methods – also known as strictly-linear (SL) – aim to estimate a complex-valued scalar d using a set of data $\boldsymbol{x}$, through the linear relation

$$y_{\text{SL}} = \boldsymbol{w}_{\text{SL}}^H \boldsymbol{x}, \tag{1.1}$$

where $\boldsymbol{w}_{\text{SL}}$ is a vector with the estimation parameters and $(\cdot)^H$ stands for the Hermitian transpose. This approach assumes that the autocorrelation matrix $\boldsymbol{C} = E\{\boldsymbol{x}\boldsymbol{x}^H\}$ [5] is enough to describe all second-order statistics of $\boldsymbol{x}$, and the input is defined as *proper* [2]. However, there are cases – which are detailed in Chapter 2 – in which an additional matrix $\boldsymbol{P} = E\{\boldsymbol{x}\boldsymbol{x}^T\}$ is needed ($(\cdot)^T$ stands for transposition). $\boldsymbol{P}$ is the pseudo-correlation matrix [1], and when it is also used, the estimate is given by

$$y_{\text{WL}} = \boldsymbol{w}_{\text{WL},1}^H \boldsymbol{x} + \boldsymbol{w}_{\text{WL},2}^H \boldsymbol{x}^*. \tag{1.2}$$

y_{WL} is called a "widely linear" estimate of d, since it depends linearly not only on $\boldsymbol{x}$, but also on $\boldsymbol{x}^*$.

SL techniques are the standard methods in complex adaptive filtering problems. In references [6] and [7], which assume $\boldsymbol{x}$ proper, SL adaptive filters are applied to channel estimation, acoustic echo cancellation, active noise control, adaptive beamforming, among many other applications. WL adaptive techniques, on the other hand, can be applied to enhance the estimation of the direction of arrival (DOA) in sensor arrays, if the signal of interest or the interferers are improper [8]. Some modulation schemes applied in communications also produce improper baseband signals, such as binary phase shift keying (BPSK) and pulse amplitude modulation (PAM) [8,9], requiring a WL approach for the best performance to recover the transmitted data. Additionally, signals obtained in functional magnetic resonance imaging (fMRI) [10], and applied to predict the direction of ocean currents and wind fields [11] are improper and also benefit from WL estimation. Many other examples of complex WL adaptive filtering applications can also be found in [10,11], in which this approach is carefully addressed.

Quaternion adaptive filters employ quaternion numbers to concisely represent multivariable data, which may not be possible with a complex-valued representation. They naturally exploit the relation between the quaternion entries to improve their estimation, which motivated the development of different SL and WL techniques (for instance, [12–19].) SL quaternion adaptive filters are applied to predict wind direction [14,19], denoising coloured images [17] and for data fusion [14]. Quaternion Kalman filters are also applied to navigation problems [20,21], since a quaternion approach leads to a orientation representation free of singularities (which does not occur when Euler angles are applied, for instance). Quaternion WL adaptive filters are applied to improve wind modelling [14], for channel prediction [22], and also to predict 3D noncircular and nonstationary body motion signals [19].

Despite the improved MSE performance that can be obtained with WL algorithms, the computational cost to compute y_{WL} is higher, since the vectors and matrices have their dimensions multiplied by two (in the case of complex-valued representation) or by four (when quaternions are applied). There is also redundant information in the WL input vector, that can be avoided with no information loss. In this text, we develop techniques that are able to exclude redundance, helping to reduce the computational cost. For this purpose, we apply a real input vector with the same information of the WL input, obtained stacking up the real and the imaginary parts of the original WL data vector. Using this approach, we eliminate the redundant data and also replace many complex-complex (quaternion-quaternion) computations by less costly real-complex (real-quaternion) operations in the estimation. In Chapters 2 and 3, we show that reduced-complexity (RC)

techniques, obtained with the real input vector, have exactly the same performance as their original WL counterparts. For the quaternion algorithms, we develop a general approach to describe quaternion gradients, and we use it to obtain mean and mean-square analyses for any technique based on the quaternion least-mean squares (QLMS) algorithm. The analyses are particularized for WL-QLMS algorithms with real-valued input, and we obtain the fastest-converging technique for this case.

In the second part of this text (Chapters 4 and 5), we employ dichotomous coordinate descent iterations (DCD) [23, 24] to develop reduced-complexity techniques. The DCD is an iterative algorithm for solving linear systems of equations such as

$$\boldsymbol{R}\boldsymbol{w} = \boldsymbol{p}, \tag{1.3}$$

where $\boldsymbol{R} = \boldsymbol{R}^H$ is a positive-definite matrix and $\boldsymbol{w}$ and $\boldsymbol{p}$ are vectors. The algorithm computes an approximate solution $\hat{\boldsymbol{w}}$, that is obtained with low computational cost, since it avoids multiplications and divisions, costly to implement in hardware. In adaptive filtering, among many applications (for instance [25–31]), it is applied to obtain a low-cost solution for the least-squares (LS) problem [25, 28], the DCD-RLS algorithm. The technique applies the solution of instant $n-1$ as the initial condition to instant n, so that DCD only requires a low number of iterations to compute the update of the solution. Using this approach, DCD-RLS is implemented with computational cost linear in the number of elements N of the input, while traditional RLS is quadratic in N. The solution is numerically stable, since the technique does not propagate the inverse autocorrelation matrix (as it occurs in the standard RLS [32]).

In this dissertation, motivated by the results of DCD-RLS, we use DCD in two applications. In the first, we modify the reduced-complexity WL-RLS algorithm, presented in Chapter 2, to employ DCD iterations. Since RC-WL-RLS is an RLS-based technique, its computational complexity is quadratic with the number of elements of the input vector, and it also suffers from numeric instability. Modifying the technique to apply the DCD, we obtain the DCD-WL-RLS algorithm. To further reduce the computational cost, we apply the new technique for cases in which the entries of the real and the imaginary parts of the input are tap delay-lines, so that a cheap update of the estimated autocorrelation matrix can be employed. Using this approach, a low-cost (linear in N) and stable version of WL-RLS is obtained.

In the second application using DCD, we develop algorithms for beamforming [7]. In sensor-arrays, when the number of sources is less than the number of sensors, the auto-

correlation matrix can become ill-conditioned. Since this matrix is required to compute the beamformer, regularization must be added to reduce the condition number. There are some techniques in the literature (for instance, [33–35]) that regularize the autocorrelation matrix, allowing the computation of beamformers such as the MVDR (minimum variance distortionless response) [7]. However, these methods are in general costly and depend on the number of sensors, such that for a large number of sensors, the complexity to compute the beamformer can become prohibitively high. Techniques such as the robust Capon beamformer [33], for instance, have complexity cubic in the number of sensors, and can be very difficult to implement for large arrays.

To develop low-cost techniques to regularize the autocorrelation matrix, we use an ℓ_1-norm regularized approach, based on complex-homotopy techniques [36, 37]. In general, ℓ_1-norm regularized techniques are applied to exploit sparsity on systems of equations. For our approach, we show that it allows us to add the minimum amount of regularization to the matrix, leading to a low-cost solution and also improving the signal-to-interference-plus-noise ratio (SINR). Based on the complex homotopy (CH) [36] and on the adaptive re-weighted homotopy (ARH) [37] algorithms, we propose the complex ARH (C-ARH), and we use it to develop low-cost iterative techniques to regularize the autocorrelation matrix. It is shown that the techniques must solve a set of low-dimension system of equations at each snapshot, which can be computed at reduced cost using DCD. We show that the DCD-based techniques have quadratic complexity in the number of sensors N, if the number of interferers does not increase with N. If the number of interferers grows linearly with N, the complexity increases, but it is shown that our approach is still advantageous for a range of values of N.

The publications that resulted from the research presented in this PhD dissertation are the following:

1. The low-cost WL techniques developed in Chapter 2 were presented in the conference paper [38].

2. The advances regarding reduced complexity quaternion WL adaptive filters were presented in two conference papers ([18] and [39]), and a journal paper is being prepared.

3. The DCD-WL-RLS algorithm of Chapter 4 was presented in the conference paper [29].

4. Preliminary results with the beamforming algorithms appeared in the conference

paper [40]. A journal paper [41] was accepted for publication in the IEEE Transactions on Aerospace and Electronic Systems, and is expected to be published in July.

1.1 Contributions of this work

The main contributions of this text are the following:

1. New reduced-complexity adaptive algorithms are proposed for complex-valued and quaternion WL estimation. The complexity reduction is achieved by eliminating the redundant second-order statistics from the autocorrelation matrix. The computational complexity of the new algorithms is shown to be equal to that of their SL counterparts.

2. We obtain a general equation for the quaternion gradient, which allows us to study the convergence and steady-state performance of any algorithm based on the quaternion least-mean squares technique (QLMS). The general gradient is applied to show that gradients similar to the i-gradient [42] lead to a fastest-converging algorithm when the regressor vector is real or when there is correlation between at most two quaternion elements (for instance, the real entries and the elements in the i axis).

3. The general quaternion gradient is used to obtain a new WL quaternion LMS algorithm. The proposed technique has about one-fourth of the computational complexity of the standard widely-linear quaternion LMS filter, and is proved to be the fastest-converging algorithm in its class.

4. A second-order analysis is proposed for the new quaternion algorithm. We obtain simple equations to compute the steady-state excess mean-square error (EMSE) and the mean-square deviation (MSD) [32], and also a second-order model to study the convergence rate of the technique.

5. We show that our new quaternion algorithm and WL-iQLMS have the same convergence performance. We also prove that the proposed technique corresponds to the four-channel least-mean squares algorithm (4-CH-LMS) written in the quaternion domain.

6. We obtain a low-cost and numerically stable DCD version of the RC-WL-RLS algorithm, which is suitable for tap delay-line input. The technique has a cost proportional to N, where N is the length of the SL input.

7. We obtain new low-cost homotopy-based techniques for beamforming. The techniques are proposed to regularize ill-conditioned systems of equations which appear in the computation of the beamformer. Using this ℓ_1-norm regularized approach, we obtain low-cost techniques, which achieve further complexity reduction when they also apply DCD iterations.

1.2 Organization of the text

The dissertation is divided in two parts: in the first part, we propose low-cost techniques which eliminate the redundant second-order information from the autocorrelation matrix in complex and quaternion widely-linear adaptive filters. In the second part, we propose WL adaptive filters and beamforming techniques which exploit DCD-iterations to obtain low-cost algorithms, suitable for hardware implementation.

In Chapter 2, we study complex-valued widely-linear estimation, introducing some basic concepts that will be used throughout the text. We compare SL and WL estimation, and we show the conditions under which WL techniques are advantageous. We develop two new techniques for complex-valued widely-linear estimation: the RC-WL-LMS and the RC-WL-RLS algorithms.

In Chapter 3, we propose a universal description to quaternion gradients, which allows us to develop a new reduced-complexity WL algorithm.

Chapter 4 starts presenting the DCD algorithm and its two versions. The DCD is extended to solve the normal equations of the RC-WL-RLS, resulting in the new low-cost DCD-WL-RLS algorithm.

In Chapter 5, we develop low-cost beamforming techniques based on the ℓ_1-norm regularized adaptive re-weighting homotopy technique. DCD iterations are used to further reduce computational complexity.

In Chapter 6, we list and discuss briefly some possibilities to continue the research.

In Chapter 7, we conclude the work.

1.3 Notation

We use lower case to describe scalar quantities (e.g.: a) and bold lower case to describe column vectors (e.g.: $\mathbf{b}$). Bold capital letters represent matrices (e.g.: $\mathbf{A}$). $\mathbf{a}_l$ stands for

the l-th column of $\mathbf{A}$, while $a_{l,m}$ denotes the entry of $\mathbf{A}$ in the l-th row and in the m-th column. We use $\mathbf{A}^{(l:m,p)}$ to obtain the elements from the position l to m, in the p-th column of $\mathbf{A}$. For a vector $\mathbf{b}$, we denote b_l as its l-th element. The conjugation of a complex or quaternion number is denoted by $(\cdot)^*$. $(\cdot)^T$ stands for transposition, while $(\cdot)^H$ indicates the transposition and conjugation of a complex or quaternion matrix or vector. We define the operator diag$(\cdot)$ for two situations: when the argument is a matrix $\mathbf{A}$, diag$(\mathbf{A})$ denotes a column vector with the diagonal elements of $\mathbf{A}$. When the argument is a vector $\mathbf{b}$, diag$(\mathbf{b})$ denotes a diagonal matrix where the non-zero elements are given by $\mathbf{b}$. We use $\boldsymbol{j}$ to identify the imaginary unity ($\boldsymbol{j} = \sqrt{-1}$) in the complex field, and we employ $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ as the imaginary units in the quaternions case. The operations $\mathcal{I}m\{\cdot\}$ and $\mathcal{R}e\{\cdot\}$ take only the imaginary and the real parts of a complex number or a quaternion, respectively, and we use the subscripts R, I, J and K to identify the real and the imaginary parts of a quaternion. $E\{\cdot\}$ is the expectation operator and $\boldsymbol{I}_N$ is the $N\times N$ identity matrix. We use col$(\cdot)$ to define a column vector, and $||\cdot||_p$ is the ℓ_p-norm. $\mathbf{0}_{M\times N}$ correspond to an $M \times N$ matrix of zeros. The symbol $\otimes$ denotes the Kronecker product.

PART I

LOW-COMPLEXITY WIDELY LINEAR ALGORITHMS

2 LOW-COMPLEXITY WIDELY-LINEAR ADAPTIVE ALGORITHMS FOR COMPLEX-VALUED SIGNALS

Widely-linear algorithms are employed to extract full second-order statistics, when their strictly-linear counterparts are unable to access this information.

Strictly linear adaptive filters are the standard approach to adaptive filtering problems in the complex domain. In the first applications of complex estimation, the SL approach was sufficient ([8] states that only after the 1990s WL estimation started to attract more attention). Therefore, important reference books such as [7] and [32] present adaptive techniques in the complex domain following only the SL approach, under the assumption that the autocorrelation matrix is sufficient for capturing all the relevant second-order information. In this case, all the applications and examples presented in these books use SL techniques. An example of application in which an SL approach can capture full second-order statistics can be obtained from [32]: an adaptive channel equalization scheme applied to recover a M-QAM ($M = 4^k$) sequence of symbols (quadrature-amplitude modulation) corrupted by Gaussian noise. In this case, the autocorrelation matrix can fully extract second-order information from the M-QAM sequence [9], so that an SL adaptive filter is the best approach for this problem. Another modulation scheme for which SL filters are the best choice is 8-PSK (phase-shift keying) [9]. In the context of array processing, SL algorithms also are adequate to estimate the direction of arrival (DOA) when the signal (or signals) of interest are obtained from one of these modulations [8] – or other inputs for which the pseudo-correlation matrix vanishes. However, other modulation schemes applied in communications, such as binary phase shift keying (BPSK) and pulse amplitude modulation (PAM) [8,9], cannot be properly processed by SL approaches. The prediction of the direction of wind fields and of ocean streams [11] also achieve lower MSE when performed by a WL adaptive filter. There are also other applications and techniques developed for WL processing, such as [43] and [44], applied for beamforming, and [29, 38], used for system identification, among many others. For all these cases, an SL adaptive filter would lead to poor performance.

The WL approach has a drawback when compared to traditional SL algorithms: the complex WL regressor vector is twice as long as the traditional SL vector, since the WL regressor consists of a concatenation of the SL regressor and its conjugate. This results in an increase of the computational complexity, an increase in the *excess* mean-square error, and possibly also (most importantly for LMS-based algorithms) a reduction in convergence speed [2]. All these problems may overshadow the possible estimation gains for some applications.

In this chapter, we present reduced complexity versions of the widely-linear LMS and RLS algorithms. For this purpose, we introduce a real regressor vector obtained with the concatenation of the real and the imaginary parts of the complex data. With this simple modification, we exclude redundant second-order information from the autocorrelation matrix, and the computational complexity of the WL filters is reduced back to almost the complexity of strictly linear filters. We prove that the modified filters are equivalent to their standard WL counterparts, only with a reduced complexity, and we give a few examples for the modified WL-LMS and WL-RLS algorithms.

The specific contribution of this chapter is the following:

- We use a linear transformation to show that the redundant terms of the correlation matrix can be avoided, without loosing second-order information. Using this approach, we obtain new versions of the WL-LMS and WL-RLS algorithms, which keep the same performance as the standard WL filters, but reduce the computational cost by a factor of one half or less.

The chapter is organized as follows. We start with a brief review on adaptive filtering algorithms in Section 2.1.1. In Section 2.1.2, we present basic results on SL and WL complex estimation, and in Section 2.2, we propose new reduced-complexity WL techniques based on real-valued regressor vector. In Section 2.3, we provide simulation examples, and in Section 2.4 we present the conclusions of the chapter.

2.1 Preliminaries

In this section, we present some preliminary concepts required to develop our reduced-complexity widely-linear techniques. We start with a brief description of adaptive filtering techniques, and then we present some results about widely-linear estimation.

2.1.1 Adaptive filtering

Adaptive filters [24,32,45] are applied when the specifications are unknown or cannot be satisfied by time-invariant filters. The parameters of an adaptive filter continually change to meet a performance requirement, using data extracted from the environment. Some applications of adaptive filtering are noise canceling, echo cancellation, system identification, equalization, adaptive beamforming and control [7, 24, 32, 45], among many others .

In general, it is assumed in adaptive filtering that at the time instant n a scalar $d(n)$ and a vector $\boldsymbol{x}(n)$ are available. The expression that models the relation between $d(n)$ and $\boldsymbol{x}(n)$ is given by[1]

$$d(n) = \boldsymbol{w}_{\text{opt}}^H \boldsymbol{x}(n) + v(n), \tag{2.1}$$

where $\boldsymbol{w}_{\text{opt}}$ is the set of optimum parameters, assumed constant here for simplicity, and $v(n)$ is noise. The goal of the adaptive filter is to compute an approximation

$$y(n) = \boldsymbol{w}^H(n)\boldsymbol{x}(n), \tag{2.2}$$

where $\boldsymbol{w}(n)$ estimates $\boldsymbol{w}_{\text{opt}}$. Using $d(n)$ and $y(n)$, an approximation error is defined, i.e.,

$$e(n) = d(n) - y(n), \tag{2.3}$$

which is applied by the filter to adjust $\boldsymbol{w}(n)$, based on some optimization criterion that minimizes a function of $e(n)$ (for instance, $f(e) = e^2(n)$) with respect to $\boldsymbol{w}(n)$ (see Figure 1).

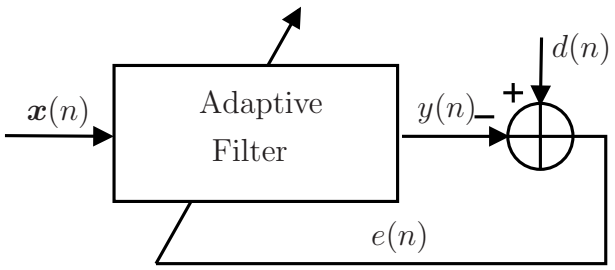

Figure 1: General description of an adaptive filter

There are different ways to update $\boldsymbol{w}(n)$, which gives rise to many techniques. When $f(e) = e^2(n)$, the technique obtained is the least mean-squares (LMS) algorithm [45], which is presented in Table 1. Using the LMS algorithm, an iterative update of $\boldsymbol{w}(n)$ is computed at each instant n, where the update is weighted by a small constant μ,

[1]It is assumed that we deal with complex quantities.

Table 1: LMS algorithm

Initialization
$\boldsymbol{w}(0) = \mathbf{0}$
for n $= 0, 1, 2, 3...$
$y(n) = \boldsymbol{w}^H(n)\boldsymbol{x}(n)$
$e(n) = d(n) - y(n)$
$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu e^*(n)\boldsymbol{x}(n)$
end

the step size of the technique. When the function to be minimized is given by $f(e) = \sum_{i=0}^{n} \nu^{n-i} e^2(i)$, the recursive least-squares (RLS) algorithm [45] is obtained. The technique is summarized in Table 2, where $\mathbf{\Phi}(n)$ is the estimate of the inverse autocorrelation matrix, while $\mathbf{k}(n)$ is the Kalman gain and $\gamma(n)$ is the conversion factor. ν is the forgetting factor and ϵ is a small constant for initializing $\mathbf{\Phi}(n)$.

Table 2: RLS algorithm

Initialization
$\boldsymbol{w}(0) = \mathbf{0}$
$\mathbf{\Phi}(0) = \epsilon\mathbf{I}$
for n $= 0, 1, 2, 3...$
$\gamma(n) = (\nu + \boldsymbol{x}^H(n)\mathbf{\Phi}(n)\boldsymbol{x}(n))^{-1}$
$\mathbf{k}(n) = \mathbf{\Phi}(n)\boldsymbol{x}(n)\gamma(n)$
$e(n) = d(n) - \boldsymbol{w}^H(n)\boldsymbol{x}(n)$
$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mathbf{k}(n)e^*(n)$
$\mathbf{\Phi}(n+1) = \frac{1}{\nu}(\mathbf{\Phi}(n) - \mathbf{k}(n)\mathbf{k}^H(n)\gamma(n))$
end

Note that the techniques presented in this section assume that $d(n)$ is modeled by (2.1), such that (2.2) is a good approximation to $d(n)$. These techniques are SL algorithms, since the estimate is linear in $\boldsymbol{x}(n)$. On the other hand, WL approaches assume that $d(n)$ is better modeled by

$$d(n) = \boldsymbol{w}_{\text{opt},1}^H \boldsymbol{x}(n) + \boldsymbol{w}_{\text{opt},2}^H \boldsymbol{x}^*(n) + v(n), \tag{2.4}$$

so that two times more coefficients are estimated, increasing the computational complexity of the algorithms. In this text, we concentrate on WL techniques, but we propose alternative versions to reduce the number of computations. In the next section, we present WL estimation in detail, which is the basis to develop the reduced-complexity techniques proposed in this work.

2.1.2 Complex-valued widely linear estimation

In this section, SL and WL estimation are presented based on the results of [1] and [2]. We start with SL estimation, and present some literature results which allow us to compute the optimum mean-square error. We proceed similarly to present WL estimation, and then we compare both approaches to show cases in which WL estimation is advantageous. A time coefficient n is used in the variables to indicate that they are time-variant.

Assume that, at each time instant, we have access to a data vector $\boldsymbol{x}(n)$, which is related with $d(n)$ by the expression in (2.1). Using a SL estimator, $d(n)$ is estimated with

$$y_{\mathrm{SL}}(n) = \boldsymbol{w}_{\mathrm{SL}}^H(n)\boldsymbol{x}(n), \tag{2.5}$$

where $\boldsymbol{w}_{\mathrm{SL}}(n)$ are the SL estimation coefficients. The optimum set of coefficients (in the mean-square sense) is obtained through the minimization of the cost function [1]

$$J_{SL}(\boldsymbol{w}_{\mathrm{SL}}(n)) = E\{|e_{\mathrm{SL}}(n)|^2\}, \tag{2.6}$$

where

$$e_{\mathrm{SL}}(n) = d(n) - y_{\mathrm{SL}}(n). \tag{2.7}$$

The minimization of (2.6) is equivalent to solving the system of equations

$$\boldsymbol{C}\boldsymbol{w}_{\mathrm{SL,opt}} = \boldsymbol{p}, \tag{2.8}$$

where

$$\boldsymbol{C} = E\{\boldsymbol{x}(n)\boldsymbol{x}^H(n)\} = (\boldsymbol{R}_{\mathrm{RR}} + \boldsymbol{R}_{\mathrm{II}}) + \boldsymbol{j}(-\boldsymbol{R}_{\mathrm{RI}} + \boldsymbol{R}_{\mathrm{IR}}), \tag{2.9}$$

$$\boldsymbol{R}_{\mathrm{RR}} = E\{\boldsymbol{x}_{\mathrm{R}}(n)\boldsymbol{x}_{\mathrm{R}}^T(n)\}, \quad \boldsymbol{R}_{\mathrm{II}} = E\{\boldsymbol{x}_{\mathrm{I}}(n)\boldsymbol{x}_{\mathrm{I}}^T(n)\}, \quad \boldsymbol{R}_{\mathrm{RI}} = \boldsymbol{R}_{\mathrm{IR}}^T = E\{\boldsymbol{x}_{\mathrm{R}}(n)\boldsymbol{x}_{\mathrm{I}}^T(n)\}, \tag{2.10}$$

is the autocorrelation matrix and $\boldsymbol{p} = E\{d^*(n)\boldsymbol{x}(n)\}$ is the cross-correlation vector [1]. $\boldsymbol{w}_{\mathrm{SL,opt}}$ is the optimum SL solution to (2.8)[2]. A key property of $\boldsymbol{w}_{\mathrm{SL,opt}}$ is that the optimum error is orthogonal to $\boldsymbol{x}$, since

$$\begin{aligned} E\{e_{\mathrm{SL}}^*(n)\boldsymbol{x}(n)\} &= E\{\boldsymbol{x}(n)(d^*(n) - \boldsymbol{x}^H(n)\boldsymbol{w}_{\mathrm{SL,opt}})\} \\ &= E\{\boldsymbol{x}(n)d^*(n)\} - E\{\boldsymbol{x}(n)\boldsymbol{x}^H(n)\}\boldsymbol{w}_{\mathrm{SL,opt}} \\ &= \boldsymbol{p} - \boldsymbol{C}\boldsymbol{w}_{\mathrm{SL,opt}} = \boldsymbol{0}. \end{aligned}$$

Using $\boldsymbol{w}_{\mathrm{SL,opt}}$ and the orthogonality principle, we can compute the minimum value of

[2]The optimum solution to a quadratic problem such as (2.6) is also known as the Wiener solution [45].

the cost function, i.e.,

$$\begin{aligned} J_{SL}(\boldsymbol{w}_{\mathrm{SL,opt}}) &= E\{|e_{\mathrm{SL}}(n)|^2\} \\ &= E\{(d(n) - y_{\mathrm{SL}}(n))^* e_{\mathrm{SL}}(n)\} \\ &= E\{d^*(n)e_{\mathrm{SL}}(n)\} - E\{y^*_{\mathrm{SL}}(n)e_{\mathrm{SL}}(n)\} \\ &= E\{d^*(n)e_{\mathrm{SL}}(n)\} - \underbrace{E\{\boldsymbol{x}^H(n)e_{\mathrm{SL}}(n)\}}_{E\{e^*_{\mathrm{SL}}(n)\boldsymbol{x}(n)\}^H=\mathbf{0}}\boldsymbol{w}_{\mathrm{SL,opt}} \\ &= E\{d^*(n)(d(n) - y_{\mathrm{SL}}(n)\} \\ &= E\{|d(n)|^2\} - E\{d^*(n)\boldsymbol{w}^H_{\mathrm{SL,opt}}\boldsymbol{x}(n)\} \\ &= E\{|d(n)|^2\} - \underbrace{\boldsymbol{w}^H_{\mathrm{SL,opt}}}_{(\boldsymbol{C}^{-1}\boldsymbol{p})^H}\underbrace{E\{d^*(n)\boldsymbol{x}(n)\}}_{\boldsymbol{p}} \\ &= E\{|d(n)|^2\} - \boldsymbol{p}^H\boldsymbol{C}^{-1}\boldsymbol{p}, \end{aligned} \tag{2.11}$$

which is a lower bound to the mean-square error performance of any technique that employs the SL approach.

Note that to compute $\boldsymbol{w}_{\mathrm{SL,opt}}$, we just required $\boldsymbol{C}$ and $\boldsymbol{p}$, obtained assuming that the approximation $y_{\mathrm{SL}}(n)$ linearly depends only on $\boldsymbol{x}(n)$. For a WL estimator, on the other hand, $y_{\mathrm{WL}}(n)$ depends on the extended vector

$$\boldsymbol{x}_{\mathrm{WL}}(n) = \begin{bmatrix} \boldsymbol{x}^T(n) & \boldsymbol{x}^H(n) \end{bmatrix}^T, \tag{2.12}$$

and is given by

$$y_{\mathrm{WL}}(n) = \underbrace{\begin{bmatrix} \boldsymbol{w}_{\mathrm{WL},1}(n) \\ \boldsymbol{w}_{\mathrm{WL},2}(n) \end{bmatrix}^H}_{\boldsymbol{w}^H_{\mathrm{WL}}(n)} \boldsymbol{x}_{\mathrm{WL}}(n) = \boldsymbol{w}^H_{\mathrm{WL},1}(n)\boldsymbol{x}(n) + \boldsymbol{w}^H_{\mathrm{WL},2}(n)\boldsymbol{x}^*(n). \tag{2.13}$$

In this case, the cost function used to obtain the optimum solution is given by

$$J_{\mathrm{WL}}(\boldsymbol{w}_{\mathrm{WL}}(n)) = E\{|e_{\mathrm{WL}}(n)|^2\}, \tag{2.14}$$

where

$$e_{\mathrm{WL}}(n) = d(n) - y_{\mathrm{WL}}(n), \tag{2.15}$$

which is equivalent to compute the solution to the system of equations

$$\begin{aligned} \boldsymbol{C}\boldsymbol{w}_{\mathrm{WL,opt},1} + \boldsymbol{P}\boldsymbol{w}_{\mathrm{WL,opt},2} &= \boldsymbol{p} \\ \boldsymbol{P}^*\boldsymbol{w}_{\mathrm{WL,opt},1} + \boldsymbol{C}^*\boldsymbol{w}_{\mathrm{WL,opt},2} &= \boldsymbol{q}^*, \end{aligned} \tag{2.16}$$

where

$$\boldsymbol{P} = (\boldsymbol{R}_{\mathrm{RR}} - \boldsymbol{R}_{\mathrm{II}}) + \boldsymbol{j}(\boldsymbol{R}_{\mathrm{RI}} + \boldsymbol{R}_{\mathrm{IR}}) \tag{2.17}$$

and $\boldsymbol{q} = E\{d(n)\boldsymbol{x}(n)\}$. $\boldsymbol{P}$ is known as the pseudo-correlation matrix [1]. Equation (2.16) can also be written as

$$\boldsymbol{C}_{\mathrm{WL}}\boldsymbol{w}_{\mathrm{WL,opt}} = \boldsymbol{p}_{\mathrm{WL}}, \tag{2.18}$$

where we define

$$\boldsymbol{C}_{\mathrm{WL}} = \begin{bmatrix} \boldsymbol{C} & \boldsymbol{P} \\ \boldsymbol{P}^* & \boldsymbol{C}^* \end{bmatrix} \tag{2.19}$$

and

$$\boldsymbol{p}_{\mathrm{WL}} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{q}^* \end{bmatrix}, \tag{2.20}$$

to shorten the notation.

In order to solve (2.18), assume that $\boldsymbol{C}$ and $\boldsymbol{P}$ are invertible matrices. To compute $\boldsymbol{w}_{\mathrm{WL,opt,1}}$, left-multiply the first equation of (2.16) by $-\boldsymbol{P}^{-1}$ and left-multiply the second equation of (2.16) by $\boldsymbol{C}^{-1*}$. Adding up the equations, one gets

$$(\boldsymbol{C}^{-1*}\boldsymbol{P}^* - \boldsymbol{P}^{-1}\boldsymbol{C})\boldsymbol{w}_{\mathrm{WL,opt,1}} = (-\boldsymbol{P}^{-1}\boldsymbol{p} + \boldsymbol{C}^{-1*}\boldsymbol{q}^*), \tag{2.21}$$

and left-multiplying (2.21) by $-\boldsymbol{P}$, yields

$$(\boldsymbol{C} - \boldsymbol{P}\boldsymbol{C}^{-1*}\boldsymbol{P}^*)\boldsymbol{w}_{\mathrm{WL,opt,1}} = (\boldsymbol{p} - \boldsymbol{P}\boldsymbol{C}^{-1*}\boldsymbol{q}^*). \tag{2.22}$$

Finally, to isolate $\boldsymbol{w}_{\mathrm{WL,opt,1}}$, multiply (2.22) by $(\boldsymbol{C} - \boldsymbol{P}\boldsymbol{C}^{-1*}\boldsymbol{P}^*)^{-1}$ from the left to obtain

$$\boldsymbol{w}_{\mathrm{WL,opt,1}} = \left[\boldsymbol{C} - \boldsymbol{P}\boldsymbol{C}^{-1*}\boldsymbol{P}^*\right]^{-1}\left[\boldsymbol{p} - \boldsymbol{P}\boldsymbol{C}^{-1*}\boldsymbol{q}^*\right]. \tag{2.23}$$

A similar approach can be applied to compute $\boldsymbol{w}_{\mathrm{WL,opt,2}}$, leading to the equation

$$\boldsymbol{w}_{\mathrm{WL,opt,2}} = \left[\boldsymbol{C}^* - \boldsymbol{P}^*\boldsymbol{C}^{-1}\boldsymbol{P}\right]^{-1}\left[\boldsymbol{q}^* - \boldsymbol{P}^*\boldsymbol{C}^{-1}\boldsymbol{p}\right]. \tag{2.24}$$

Using the solution of the extended system of equations, we can compute $J_{\mathrm{WL}}(\boldsymbol{w}_{\mathrm{WL,opt}})$ to compare the minimum mean-square error achieved by SL and WL estimators. First, we need to extend the orthogonality condition to the WL case and show that the optimum error is orthogonal to $\boldsymbol{x}(n)$ and $\boldsymbol{x}^*(n)$, that is

$$\begin{aligned} E\{e^*(n)\boldsymbol{x}(n)\} &= E\{(d^*(n) - y^*_{\mathrm{WL}}(n))\boldsymbol{x}(n)\} \\ &= E\{d^*(n)\boldsymbol{x}(n)\} - E\{\boldsymbol{x}(n)(\boldsymbol{x}^H(n)\boldsymbol{w}_{\mathrm{WL,opt.1}} + \boldsymbol{x}^T(n)\boldsymbol{w}_{\mathrm{WL,opt.2}})\} \\ &= \boldsymbol{p} - (\boldsymbol{C}\boldsymbol{w}_{\mathrm{WL,opt,1}} + \boldsymbol{P}\boldsymbol{w}_{\mathrm{WL,opt,2}}) = \boldsymbol{0}, \end{aligned} \tag{2.25}$$

where we used the first equation of (2.16) to obtain the last equality, and

$$\begin{aligned} E\{e^*(n)\boldsymbol{x}^*(n)\} &= E\{(d^*(n) - y^*_{\mathrm{WL}}(n))\boldsymbol{x}^*(n)\} \\ &= E\{d^*(n)\boldsymbol{x}^*(n)\} - E\{\boldsymbol{x}^*(n)(\boldsymbol{x}^H(n)\boldsymbol{w}_{\mathrm{WL,opt},1} + \boldsymbol{x}^T(n)\boldsymbol{w}_{\mathrm{WL,opt.2}})\} \\ &= \boldsymbol{q}^* - (\boldsymbol{P}^*\boldsymbol{w}_{\mathrm{WL,opt},1} + \boldsymbol{C}^*\boldsymbol{w}_{\mathrm{WL,opt},2}) = \boldsymbol{0}, \end{aligned} \tag{2.26}$$

where the last equality is obtained by applying the second equation of (2.16).

Equation (2.14) can be expressed as

$$\begin{aligned} J_{\mathrm{WL}}(\boldsymbol{w}_{\mathrm{WL,opt}}) &= E\{|e_{\mathrm{WL}}(n)|^2\} \\ &= E\{(d(n) - y_{\mathrm{WL}}(n))^* e_{\mathrm{WL}}(n)\} \\ &= E\{d^*(n)e_{\mathrm{WL}}(n)\} - E\{y^*_{\mathrm{WL}}(n)e_{\mathrm{WL}}(n)\}, \end{aligned} \tag{2.27}$$

so that we must apply the orthogonality conditions to compute $E\{y^*_{\mathrm{WL}}(n)e_{\mathrm{WL}}(n)\}$, i.e.,

$$\begin{aligned} E\{y^*_{\mathrm{WL}}(n)e_{\mathrm{WL}}(n)\} &= E\{e_{\mathrm{WL}}(n)y^*_{\mathrm{WL}}(n)\} \\ &= E\{e_{\mathrm{WL}}(n)(\boldsymbol{x}^H(n)\boldsymbol{w}_{\mathrm{WL,opt},1} + \boldsymbol{x}^T(n)\boldsymbol{w}_{\mathrm{WL,opt},2})\} \\ &= \underbrace{E\{e_{\mathrm{WL}}(n)\boldsymbol{x}^H(n)\}}_{=E\{e^*_{\mathrm{WL}}(n)\boldsymbol{x}(n)\}^H=\boldsymbol{0}}\boldsymbol{w}_{\mathrm{WL,opt},1} + \underbrace{E\{e_{\mathrm{WL}}(n)\boldsymbol{x}^T(n)\}}_{=E\{e^*_{\mathrm{WL}}(n)\boldsymbol{x}^*(n)\}^H=\boldsymbol{0}}\boldsymbol{w}_{\mathrm{WL,opt},2} = \boldsymbol{0}. \end{aligned} \tag{2.28}$$

Using this result, equation (2.27) reduces to

$$\begin{aligned} J_{\mathrm{WL}}(\boldsymbol{w}_{\mathrm{WL,opt}}) &= E\{d^*(n)e_{\mathrm{WL}}(n)\} \\ &= E\{d^*(n)(d(n) - y_{\mathrm{WL}}(n))\} \\ &= E\{|d(n)|^2\} - E\{d^*(n)\boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{x}(n)\} - E\{d^*(n)\boldsymbol{w}^H_{\mathrm{WL,opt},2}\boldsymbol{x}^*(n)\} \\ &= E\{|d(n)|^2\} - \boldsymbol{w}^H_{\mathrm{WL,opt},1}\underbrace{E\{d^*(n)\boldsymbol{x}(n)\}}_{\boldsymbol{p}} - \boldsymbol{w}^H_{\mathrm{WL,opt},2}\underbrace{E\{d^*(n)\boldsymbol{x}^*(n)\}}_{\boldsymbol{q}^*}, \end{aligned} \tag{2.29}$$

which expresses the MSE lower bound for WL estimators.

We can express the difference between the optimum mean-square error obtained with SL and WL estimators using (2.11) and (2.29),

$$\begin{aligned} \Delta J &= J_{\mathrm{SL}}(\boldsymbol{w}_{\mathrm{SL,opt}}) - J_{\mathrm{WL}}(\boldsymbol{w}_{\mathrm{WL,opt}}) \\ &= -\boldsymbol{p}^H\boldsymbol{C}^{-1}\boldsymbol{p} + \boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{p} + \boldsymbol{w}^H_{\mathrm{WL,opt},2}\boldsymbol{q}^*. \end{aligned} \tag{2.30}$$

From the first equation of (2.16), we can write $\boldsymbol{w}_{\mathrm{WL,opt},1} = \boldsymbol{C}^{-1}\boldsymbol{p} - \boldsymbol{C}^{-1}\boldsymbol{P}\boldsymbol{w}_{\mathrm{WL,opt},2}$. Substituting this relation in (2.30), after some algebra one gets

$$\begin{aligned} \Delta J &= \boldsymbol{w}^H_{\mathrm{WL,opt},2}(\boldsymbol{q}^* - \boldsymbol{P}^H\boldsymbol{C}^{-1}\boldsymbol{p}) \\ &= \left[\boldsymbol{q}^* - \boldsymbol{P}^*\boldsymbol{C}^{-1}\boldsymbol{p}\right]^H \left(\left[\boldsymbol{C}^* - \boldsymbol{P}^*\boldsymbol{C}^{-1}\boldsymbol{P}\right]^{-1}\right)^H \left[\boldsymbol{q}^* - \boldsymbol{P}^H\boldsymbol{C}^{-1}\boldsymbol{p}\right]. \end{aligned} \tag{2.31}$$

Note that (2.31) can be simplified. Using the fact that $\boldsymbol{C}$ is hermitian and invertible, such that $\boldsymbol{C}^{-1}$ is also hermitian, and showing that

$$\boldsymbol{P}^H = E\{\boldsymbol{x}(n)\boldsymbol{x}^T(n)\}^H = E\{\boldsymbol{x}^*(n)\boldsymbol{x}^H(n)\} = E\{\boldsymbol{x}(n)\boldsymbol{x}^T(n)\}^* = \boldsymbol{P}^*, \tag{2.32}$$

we can simplify

$$\begin{aligned} (\left[\boldsymbol{C}^* - \boldsymbol{P}^*\boldsymbol{C}^{-1}\boldsymbol{P}\right]^{-1})^H &= (\left[\boldsymbol{C}^* - \boldsymbol{P}^*\boldsymbol{C}^{-1}\boldsymbol{P}\right]^H)^{-1} \\ &= \left[(\boldsymbol{C}^*)^H - \boldsymbol{P}^H(\boldsymbol{C}^{-1})^H(\boldsymbol{P}^*)^H\right]^{-1} \\ &= \left[\boldsymbol{C}^* - \boldsymbol{P}^*\boldsymbol{C}^{-1}\boldsymbol{P}\right]^{-1}, \end{aligned} \tag{2.33}$$

and

$$\Delta J = \left[\boldsymbol{q}^* - \boldsymbol{P}^*\boldsymbol{C}^{-1}\boldsymbol{p}\right]^H \left[\boldsymbol{C}^* - \boldsymbol{P}^*\boldsymbol{C}^{-1}\boldsymbol{P}\right]^{-1} \left[\boldsymbol{q}^* - \boldsymbol{P}^*\boldsymbol{C}^{-1}\boldsymbol{p}\right]. \tag{2.34}$$

Noting that $\left[\boldsymbol{C}^* - \boldsymbol{P}^*\boldsymbol{C}^{-1}\boldsymbol{P}\right]$ is a positive-definite matrix (since it corresponds to the Schur complement [46] of $\boldsymbol{C}$ in $\boldsymbol{C}_{\text{WL}}$), $\Delta J \geq 0$ always. Therefore, the minimum mean-square-error obtained with the WL estimator is never worse than that obtained with a SL one.

To identify cases in which the WL approach outperforms SL estimation, or when they achieve the same MSE performance, one must study the statistics of $\boldsymbol{x}(n)$ and $d(n)$, and verify how expression (2.34) is affected by them. For this purpose, we present some results on the statistics of $\boldsymbol{x}(n)$ and d to define circularity – the condition for which SL and WL estimator have the same MSE. Then, we present some examples of WL estimators and how they are affected by different conditions on the statistics of $\boldsymbol{x}(n)$ and $d(n)$.

2.1.2.1 Circularity and second-order circularity

A complex random variable $x = x_{\text{R}} + \boldsymbol{j}x_{\text{I}}$, where x_{R} and x_{I} are real entries, is *circular in the strict-sense* [2] (or only *circular*) if its joint probability density function (pdf)

$$f_x(x) = f_{x_{\text{R}},x_{\text{I}}}(x_{\text{R}}, x_{\text{I}}) \tag{2.35}$$

is invariant to rotations, i.e., if x and $xe^{j\theta}$ have the same pdf. Circularity is a strong property and requires the pdf of x or all the moments [5] of $f_x(x)$ and $f_x(xe^{j\theta})$ to be verified. Since these quantities are not available in general, a less restrictive criterion can be used to verify if the variable is at least *second-order circular* or *proper* [2]. This condition takes into account only second-order moments of the pdf, which may be easier to access or to estimate.

To present the conditions for which second-order circularity holds, consider the $N \times 1$

complex random vector $\boldsymbol{x}$. Without loss of generality, assume that their entries are zero-mean. The second-order statistics of $\boldsymbol{x}$ can be completely defined by two matrices, $\boldsymbol{C}$ (see eq. (2.9)) and $\boldsymbol{P}$ (see (2.17))[1, 2]. We define $\boldsymbol{x}$ as second-order circular [2] when $\boldsymbol{P} = \boldsymbol{0}$, which is equivalent to the conditions

$$\boldsymbol{R}_{\mathrm{RR}} = \boldsymbol{R}_{\mathrm{II}} \text{ and } \boldsymbol{R}_{\mathrm{RI}} = -\boldsymbol{R}_{\mathrm{IR}} \tag{2.36}$$

simultaneously (see (2.10)). When $\boldsymbol{x}$ is second-order circular, all the second-order statistics can be obtained from $\boldsymbol{C}$. However, when $\boldsymbol{P} \neq \boldsymbol{0}$, both matrices are required to fully exploit these statistics of $\boldsymbol{x}$. Additionally, $\boldsymbol{x}$ and d are jointly second-order circular when the vector

$$[\boldsymbol{x}^T \; d]^T \tag{2.37}$$

is proper, which implies $\boldsymbol{P} = \boldsymbol{0}$ and the additional condition

$$\boldsymbol{q} = E\{d(n)\boldsymbol{x}(n)\} = \boldsymbol{0}. \tag{2.38}$$

As shown in Section 2.1.2, SL estimators are unable to access $\boldsymbol{P}$ and $\boldsymbol{q}$, while their WL counterparts exploit all the second-order information, which can be used to reduce the mean-square error [2]. In the next sections, we present some examples to show how WL estimators perform for different conditions on $\boldsymbol{P}$ and $\boldsymbol{q}$.

2.1.2.2 Jointly second-order circular case

This case occurs when $\boldsymbol{P} = \boldsymbol{0}$ and $\boldsymbol{q} = \boldsymbol{0}$, which implies $\Delta J = 0$ (see eq. (2.34)). In this situation, SL and WL approaches have the same optimum mean-square error, since $\boldsymbol{P}$ and $\boldsymbol{q}$ do not contribute to the estimation. However, since $\boldsymbol{w}_{\mathrm{WL}}(n)$ has two times the number of entries of $\boldsymbol{w}_{\mathrm{SL}}(n)$, the WL estimator is more costly to compute. Due to the higher number of entries to estimate, one must also expect slower convergence rate in WL estimators and higher excess mean-square error [2].

2.1.2.3 Second-order circular regressor

Assume that $\boldsymbol{P} = \boldsymbol{0}$, but $\boldsymbol{q} \neq \boldsymbol{0}$. Equations (2.23) and (2.24) simplify to

$$\begin{aligned} \boldsymbol{w}_{\mathrm{WL,opt,1}} &= \boldsymbol{C}^{-1}\boldsymbol{p} \\ \boldsymbol{w}_{\mathrm{WL,opt,2}} &= (\boldsymbol{C}^*)^{-1}\boldsymbol{q}^*. \end{aligned} \tag{2.39}$$

In this case, the vector $\boldsymbol{w}_{\mathrm{WL,opt,1}}$ is equal to the coefficients estimated by the SL estimator, but there is an additional vector $\boldsymbol{w}_{\mathrm{WL,opt,2}}$, obtained by the WL estimator. Using eq. (2.39)

in (2.34), ΔJ can be evaluated as given by

$$\Delta J = \boldsymbol{q}^T(\boldsymbol{C}^*)^{-1}\boldsymbol{q}^* = \boldsymbol{q}^H\boldsymbol{C}^{-1}\boldsymbol{q} > 0, \tag{2.40}$$

which shows that the WL approach leads to smaller MSE than the SL estimation in this situation.

2.1.2.4 Real-valued desired signal (real $d(n)$)

When $d(n)$ is real[3], but $\boldsymbol{x}(n)$ is still a complex vector, one can manipulate eq. (2.16) to show that $\boldsymbol{p} = \boldsymbol{q}$. Using this result in (2.23) and (2.24), one gets $\boldsymbol{w}_{\mathrm{WL,opt},2} = \boldsymbol{w}^*_{\mathrm{WL,opt},1}$, which can be used in eq. (2.13) to obtain

$$y_{\mathrm{WL}}(n) = \boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{x}(n) + (\boldsymbol{w}^H_{\mathrm{WL,opt},1})^*\boldsymbol{x}^*(n) = 2\mathcal{R}e\{\boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{x}(n)\}. \tag{2.41}$$

Substituting (2.41) in the cost function (2.14), $J_{\mathrm{WL}}(\boldsymbol{w}_{\mathrm{WL,opt}})$ is given by

$$\begin{aligned} J_{\mathrm{WL}}(\boldsymbol{w}_{\mathrm{WL,opt}}) &= \sigma_d^2 - \boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{p} - \underbrace{\boldsymbol{w}^H_{\mathrm{WL,opt},2}}_{\boldsymbol{w}^T_{\mathrm{WL,opt},1}}\boldsymbol{p}^* \\ &= \sigma_d^2 - \boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{p} - (\boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{p})^* \\ &= \sigma_d^2 - 2\mathcal{R}e\{\boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{p}\}. \end{aligned} \tag{2.42}$$

We can make a direct comparison of SL and WL estimation if we also assume that process $\{x(n)\}$ is second-order circular, which implies $\boldsymbol{P} = \boldsymbol{0}$ [4].

Assuming that $\boldsymbol{P} = 0$, the first equation of (2.23) reduces to $\boldsymbol{C}\boldsymbol{w}_{\mathrm{WL,opt},1} = \boldsymbol{p}$, such that $\boldsymbol{w}_{\mathrm{SL,opt}} = \boldsymbol{w}_{\mathrm{WL,opt},1}$. Applying this result in eq. (2.42), one gets

$$\begin{aligned} J_{\mathrm{WL}}(\boldsymbol{w}_{\mathrm{WL,opt}}) &= \sigma_d^2 - \boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{p} - (\boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{p})^* \\ &= \sigma_d^2 - \boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{C}\boldsymbol{w}_{\mathrm{WL,opt},1} - (\boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{C}\boldsymbol{w}_{\mathrm{WL,opt},1})^*. \end{aligned} \tag{2.43}$$

Recalling that $\boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{C}\boldsymbol{w}_{\mathrm{WL,opt},1}$ results in a real number, (2.43) reduces to

$$\begin{aligned} J_{\mathrm{WL}}(\boldsymbol{w}_{\mathrm{WL,opt}}) &= \sigma_d^2 - 2\boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{C}\boldsymbol{w}_{\mathrm{WL,opt},1} \\ &= \sigma_d^2 - 2\boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{p}, \end{aligned} \tag{2.44}$$

such that the difference

$$\Delta J = J_{\mathrm{SL}}(\boldsymbol{w}_{\mathrm{SL,opt}}) - J_{\mathrm{WL}}(\boldsymbol{w}_{\mathrm{WL,opt}}) = \boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{p} = \boldsymbol{w}^H_{\mathrm{WL,opt},1}\boldsymbol{C}\boldsymbol{w}_{\mathrm{WL,opt},1} > 0 \tag{2.45}$$

[3]This case is presented since it appears as an WL example in [2], and is also applied in a simulation presented in this chapter.

[4]Note that even if $\boldsymbol{P} \neq \boldsymbol{0}$, eq. (2.34) can also be used to compare WL and SL approaches.

is always positive. Equation (2.45) proves that the WL estimator is also advantageous when $d(n)$ is real and $\boldsymbol{P} = \mathbf{0}$, improving the MSE performance.

In this section, we presented WL estimation and gave some examples in which this approach outperforms SL techniques. Based on these results, in the next section we propose low-cost versions of WL adaptive techniques, and show that they achieve the same MSE obtained with the original methods.

2.2 Reduced-complexity widely-linear adaptive filters

WL adaptive filters are direct extensions of their SL counterparts, only with twice as large regressors and weight vectors. The WL versions of the LMS and the RLS algorithms, for instance, are presented in Tables 3 and 4.

Table 3: WL-LMS algorithm

Initialization
$\quad \boldsymbol{w}_{\mathrm{WL}}(0) = \mathbf{0}_{2N\times 1}$
for n $= 0, 1, 2, 3...$
$\quad y_{\mathrm{WL}}(n) = \boldsymbol{w}_{\mathrm{WL}}^H(n)\boldsymbol{x}_{\mathrm{WL}}(n)$
$\quad e_{\mathrm{WL}}(n) = d(n) - y_{\mathrm{WL}}(n)$
$\quad \boldsymbol{w}_{\mathrm{WL}}(n+1) = \boldsymbol{w}_{\mathrm{WL}}(n) + \mu_{\mathrm{WL}} e_{\mathrm{WL}}^*(n)\boldsymbol{x}_{\mathrm{WL}}(n)$
end

Table 4: WL-RLS algorithm

Initialization
$\quad \boldsymbol{w}_{\mathrm{WL}}(0) = \mathbf{0}_{2\mathrm{N}\times 1}$
$\quad \boldsymbol{\Phi}_{\mathrm{WL}}(0) = \epsilon\mathbf{I}_{2\mathrm{N}\times 2\mathrm{N}}$
for n $= 0, 1, 2, 3...$
$\quad \gamma(n) = (\nu + \boldsymbol{x}_{\mathrm{WL}}^H(n)\boldsymbol{\Phi}_{\mathrm{WL}}(n)\boldsymbol{x}_{\mathrm{WL}}(n))^{-1}$
$\quad \mathbf{k}_{\mathrm{WL}}(n) = \boldsymbol{\Phi}_{\mathrm{WL}}(n)\boldsymbol{x}_{\mathrm{WL}}(n)\gamma(n)$
$\quad e_{\mathrm{WL}}(n) = d(n) - \boldsymbol{w}_{\mathrm{WL}}^H(n)\boldsymbol{x}_{\mathrm{WL}}(n)$
$\quad \boldsymbol{w}_{\mathrm{WL}}(n+1) = \boldsymbol{w}_{\mathrm{WL}}(n) + \mathbf{k}_{\mathrm{WL}}(n)e_{\mathrm{WL}}^*(n)$
$\quad \boldsymbol{\Phi}_{\mathrm{WL}}(n+1) = \frac{1}{\nu}(\boldsymbol{\Phi}_{\mathrm{WL}}(n) - \mathbf{k}_{\mathrm{WL}}(n)\mathbf{k}_{\mathrm{WL}}^H(n)\gamma(n))$
end

Using the WL approach, we can recover full second-order information from $\boldsymbol{C}_{\mathrm{WL}}$, since both $\boldsymbol{C}$ and $\boldsymbol{P}$ are available, as shown in Section 2.1.2. However, matrices $\boldsymbol{R}_{\mathrm{RR}}$, $\boldsymbol{R}_{\mathrm{II}}$ and $\boldsymbol{R}_{\mathrm{RI}}$ appear redundantly in the block-matrices which constitute $\boldsymbol{C}_{\mathrm{WL}}$, since each block ($\boldsymbol{C}$, $\boldsymbol{C}^*$, $\boldsymbol{P}$ and $\boldsymbol{P}^*$) contains all these terms (see equations (2.9), (2.10) and (2.17)).

The redundance can be avoided using a $2N$-real regressor vector which contains the same information of the $2N$-complex WL regressor vector. Below, we show that using this approach one can obtain low-cost techniques for WL processing, and we propose reduced-complexity versions of the WL-LMS and WL-RLS algorithms.

2.2.1 Avoiding redundant information in the autocorrelation matrix

Considering (2.9) and (2.17), we can observe that the information contained in matrices $\mathbf{R}_{\text{RR}}$, $\mathbf{R}_{\text{II}}$ and $\mathbf{R}_{\text{RI}}$ is repeated in $\boldsymbol{C}$ and in $\boldsymbol{P}$ and, consequently, in $\boldsymbol{C}_{\text{WL}}$. This argument intuitively leads to the idea of obtaining an equivalent real matrix for $\boldsymbol{C}_{\text{WL}}$, avoiding redundance. It is possible to redefine the regressor vector in terms of its real and imaginary parts, i.e.,

$$\boldsymbol{x}_{\text{RC}}(n) = \begin{bmatrix} \boldsymbol{x}_{\text{R}}(n) \\ \boldsymbol{x}_{\text{I}}(n) \end{bmatrix}. \tag{2.46}$$

In this case, the autocorrelation matrix is given by

$$\boldsymbol{C}_{\text{RC}} = \begin{bmatrix} \mathbf{R}_{\text{RR}} & \mathbf{R}_{\text{RI}} \\ \mathbf{R}_{\text{IR}} & \mathbf{R}_{\text{II}} \end{bmatrix}.$$

Since $\boldsymbol{x}_{\text{RC}}(n)$ and $\boldsymbol{C}_{\text{RC}}$ contain the same information as in $\boldsymbol{x}_{\text{WL}}(n)$, algorithms based on $\boldsymbol{x}_{\text{RC}}(n)$ will access the same statistics accessed by standard WL filters. However, the computational cost will be significantly lower: since the regressor is now real, an algorithm based on $\boldsymbol{x}_{\text{RC}}(n)$ will replace several multiplications of two complex numbers by multiplications of a complex with real numbers, which require each 2 multiplications and 2 sums less.

Considering the advantages that may be obtained with the real-input vector, our goal is to propose algorithms which apply $\boldsymbol{x}_{\text{RC}}(n)$ instead of $\boldsymbol{x}_{\text{WL}}(n)$. However, we first show the relation between the optimum weight vectors obtained with both inputs, in order to prove that they lead to the same estimation results. For this purpose, define the transformation matrix

$$\mathbf{U} = \begin{bmatrix} \mathbf{I}_N & \boldsymbol{j}\mathbf{I}_N \\ \mathbf{I}_N & -\boldsymbol{j}\mathbf{I}_N \end{bmatrix}, \tag{2.47}$$

where $\mathbf{I}_N$ is the identity and $\mathbf{U}\mathbf{U}^H = 2\mathbf{I}$. Note that

$$\boldsymbol{x}_{\text{RC}}(n) = \mathbf{U}^H \boldsymbol{x}_{\text{WL}}(n) \tag{2.48}$$

so that one can write

$$\boldsymbol{C}_{\mathrm{RC}} = \mathbf{U}^H \boldsymbol{C}_{\mathrm{WL}} \mathbf{U}. \tag{2.49}$$

The cross-correlation vector obtained with $\boldsymbol{x}_{\mathrm{RC}}(n)$ corresponds to

$$\boldsymbol{p}_{\mathrm{RC}} = E\{d(n)\boldsymbol{x}_{\mathrm{RC}}\} = E\{d(n)\boldsymbol{U}^H \boldsymbol{x}_{\mathrm{WL}}\} = \boldsymbol{U}^H \boldsymbol{p}_{\mathrm{WL}}. \tag{2.50}$$

The Wiener solution is given by

$$\boldsymbol{w}_{\mathrm{RC,opt}} = \boldsymbol{C}_{\mathrm{RC}}^{-1}\boldsymbol{p}_{\mathrm{RC}} = \left(\mathbf{U}^H \boldsymbol{C}_{\mathrm{WL}} \mathbf{U}\right)^{-1} \mathbf{U}^H \boldsymbol{p}_{\mathrm{WL}} = \frac{1}{2}\mathbf{U}^H \boldsymbol{w}_{\mathrm{WL,opt}},$$

so that the optimum coefficients obtained with $\boldsymbol{x}_{\mathrm{RC}}(n)$ are just a transformed version of $\boldsymbol{w}_{\mathrm{WL,opt}}$, which is applied in standard WL adaptive filtering.

Using a similar approach, we show the relation between the weight vectors and the estimates obtained with both approaches. Applying $\mathbf{U}$ to (2.13), we obtain

$$y_{\mathrm{WL}}(n) = \frac{1}{2}\boldsymbol{w}_{\mathrm{WL}}^H(n)\mathbf{U}\mathbf{U}^H \boldsymbol{x}_{\mathrm{WL}}(n).$$

Defining

$$\boldsymbol{w}_{\mathrm{RC}}(n) = \frac{1}{2}\mathbf{U}^H \boldsymbol{w}_{\mathrm{WL}}(n), \tag{2.51}$$

we note that the estimates computed using the transformed regressor and weight vector are equivalent to those obtained with the original vectors, since

$$y_{\mathrm{WL}}(n) = y_{\mathrm{RC}}(n) = \boldsymbol{w}_{\mathrm{RC}}^H(n)\boldsymbol{x}_{\mathrm{RC}}(n).$$

From these results, one can see that using the transformed regressor, we compute the same estimate as obtained with $\boldsymbol{x}_{\mathrm{WL}}(n)$, and that both approaches achieve the same optimum MSE. Next, we derive transformed versions of the widely-linear LMS and RLS algorithms, obtained with matrix $\mathbf{U}$ in order to use real regressor vectors.

2.2.2 Reduced-complexity widely linear LMS (RC-WL-LMS)

Applying the transformation $\mathbf{U}$ to the WL-LMS equations (Table 3) and using the real regressor vector $\boldsymbol{x}_{\mathrm{RC}}(n)$, we obtain

$$y_{\mathrm{RC}}(n) = \boldsymbol{w}_{\mathrm{RC}}^H(n)\boldsymbol{x}_{\mathrm{RC}}(n),$$

$$e_{\mathrm{RC}}(n) = d(n) - y_{\mathrm{RC}}(n)$$

and

$$\boldsymbol{w}_{\mathrm{RC}}(n+1) = \boldsymbol{w}_{\mathrm{RC}}(n) + \mu_{\mathrm{RC}} e^{*}_{\mathrm{RC}}(n)\boldsymbol{x}_{\mathrm{RC}}(n),$$

with the initial condition

$$\boldsymbol{w}_{\mathrm{RC}}(0) = \mathbf{0}_{2\mathrm{N}\times 1}.$$

Note that the relation between the step sizes, $\mu_{\mathrm{RC}} = 2\mu_{\mathrm{WL}}$, comes from the transformation $\mathbf{U}$.

Indeed, $y_{\mathrm{RC}}(n)$ and $e_{\mathrm{RC}}(n)$ are exactly the same as in the standard WL version, since the transformation $\mathbf{U}$ does not affect them. Furthermore, the analysis of the WL-LMS algorithm in [2] can be easily extended to the reduced complexity version. Since the relation between $\boldsymbol{w}_{\mathrm{RC}}(n)$ and $\boldsymbol{w}_{\mathrm{WL}}(n)$ is through a multiple of a unitary matrix (2.47), the mean-square behavior of both algorithms is similar.

2.2.3 Reduced-complexity widely linear RLS (RC-WL-RLS)

We can also obtain a reduced-complexity version of the WL-RLS algorithm applying the transformation $\mathbf{U}$. For this purpose, recall the equations of WL-RLS, presented in Table 4. We start showing that the estimate of the inverse correlation matrix is a transformed version of its WL counterpart. To prove this, first note that using (2.49), the inverse of the correlation matrix $\boldsymbol{C}_{\mathrm{RC}}$ can be written as given by

$$\boldsymbol{C}^{-1}_{\mathrm{RC}} = \mathbf{U}^{-1}\boldsymbol{C}^{-1}_{\mathrm{WL}}\left(\mathbf{U}^{H}\right)^{-1}. \tag{2.52}$$

Using properties of the similarity transformation, i.e., $\mathbf{U}^{-1} = \mathbf{U}^{H}/2$ and $\left(\mathbf{U}^{H}\right)^{-1} = \mathbf{U}/2$, (2.52) can be rewritten as

$$\boldsymbol{C}^{-1}_{\mathrm{RC}} = \frac{\mathbf{U}^{H}}{2}\boldsymbol{C}^{-1}_{\mathrm{WL}}\frac{\mathbf{U}}{2}. \tag{2.53}$$

Denoting by $\boldsymbol{\Phi}_{\mathrm{RC}}(n)$ the estimate of $\boldsymbol{C}^{-1}_{\mathrm{RC}}$, the following relation holds

$$\boldsymbol{\Phi}_{\mathrm{RC}}(n) = \frac{\mathbf{U}^{H}}{2}\boldsymbol{\Phi}_{\mathrm{WL}}(n)\frac{\mathbf{U}}{2}. \tag{2.54}$$

Therefore, since matrix $\boldsymbol{\Phi}_{\mathrm{WL}}(n)$ is initialized as $\boldsymbol{\Phi}_{\mathrm{WL}}(0) = \epsilon\mathbf{I}$ in the WL-RLS algorithm, to ensure the equivalence to the reduced-complexity version proposed here, the matrix $\boldsymbol{\Phi}_{\mathrm{RC}}(n)$ must be initialized as $\boldsymbol{\Phi}_{\mathrm{RC}}(0) = (\epsilon/2)\mathbf{I}$.

Analogously, applying the transformation $\mathbf{U}$ to the conversion factor and the Kalman gain, we arrive at

$$\gamma(n) = \left[\nu + \boldsymbol{x}^{H}_{\mathrm{WL}}(n)\frac{\mathbf{U}\mathbf{U}^{H}}{2}\boldsymbol{\Phi}_{\mathrm{WL}}(n)\frac{\mathbf{U}\mathbf{U}^{H}}{2}\boldsymbol{x}_{\mathrm{WL}}(n)\right]^{-1} \tag{2.55}$$

and

$$\mathbf{k}_{\mathrm{RC}}(n) = \frac{\mathbf{U}^H}{2}\mathbf{k}_{\mathrm{WL}}(n). \tag{2.56}$$

Now, using $\boldsymbol{x}_{\mathrm{RC}}(n)$ and $\mathbf{w}_{\mathrm{RC}}(n)$ as defined in (2.46) and (2.51), we obtain the RC-WL-RLC, i.e.,

$$\begin{aligned}
\gamma(n) &= (\nu + \boldsymbol{x}_{\mathrm{RC}}^H(n)\boldsymbol{\Phi}_{\mathrm{RC}}(n)\boldsymbol{x}_{\mathrm{RC}}(n))^{-1}, \\
\mathbf{k}_{\mathrm{RC}}(n) &= \boldsymbol{\Phi}_{\mathrm{RC}}(n)\boldsymbol{x}_{\mathrm{RC}}(n)\gamma(n), \\
e_{\mathrm{RC}}(n) &= d(n) - \boldsymbol{w}_{\mathrm{RC}}^H(n)\boldsymbol{x}_{\mathrm{RC}}(n), \\
\boldsymbol{w}_{\mathrm{RC}}(n+1) &= \boldsymbol{w}_{\mathrm{RC}}(n) + \mathbf{k}_{\mathrm{RC}}(n)e_{\mathrm{RC}}^*(n),
\end{aligned}$$

and

$$\boldsymbol{\Phi}_{\mathrm{RC}}(n+1) = \frac{1}{\nu}(\boldsymbol{\Phi}_{\mathrm{RC}}(n) - \mathbf{k}_{\mathrm{RC}}(n)\mathbf{k}_{\mathrm{RC}}^H(n)\gamma(n)).$$

Again, the transformation reduces the presence of complex numbers in the calculations, which results in an algorithm with smaller cost.

Tables 5 and 6 show the computational cost of the SL, standard WL and RC-WL algorithms based on LMS and RLS approaches, in terms of the number of real sums ($+$), real multiplications ($\times$) and real divisions ($\div$). We assume that the SL regressor vector has N entries.

Table 5: Computational complexity for SL-LMS, WL-LMS and RC-WL-LMS in terms of real operations per iteration

Algorithm	$+$	$\times$	$\div$
SL-LMS	$8N$	$8N+2$	-
WL-LMS	$16N$	$16N+2$	-
RC-WL-LMS	$8N$	$8N+2$	-

Table 6: Computational complexity for SL-RLS, WL-RLS and RC-WL-RLS in terms of real operations per iteration

Algorithm	$+$	$\times$	$\div$
SL-RLS	$6N^2+14N-1$	$7N^2+21N+1$	1
WL-RLS	$24N^2+28N-1$	$28N^2+42N+1$	1
RC-WL-RLS	$6N^2+11N$	$8N^2+14N+1$	1

From Table 5, we notice that the RC-WL-LMS and SL-LMS algorithms have the same complexity, which is almost two times smaller than that of the WL-LMS algorithm.

Similarly, Table 6 presents the complexity results for RLS. In this comparison, we have taken advantage of the symmetry of $\mathbf{\Phi}_{\mathrm{RC}}(n)$ to compute only the elements in the main diagonal and above it. We have used the approach of [32], p.201. Surprisingly, the number of sums needed by the RC-WL-RLS algorithm is smaller than that of the other two algorithms. The number of real multiplications is only a little higher than that of SL-RLS.

2.3 Simulations

In order to compare the SL, WL and RC-WL algorithms, we show some simulations considering the identification system model used in [2]. We defined a complex-valued random process $\boldsymbol{x}(n) = \sqrt{1-\alpha^2}\boldsymbol{x}_{\mathrm{R}}(n) + \boldsymbol{j}\alpha\boldsymbol{x}_{\mathrm{I}}(n)$, where $\boldsymbol{x}_{\mathrm{R}}(n)$ and $\boldsymbol{x}_{\mathrm{I}}(n)$ are two uncorrelated real-valued Gaussian processes with zero mean. The factor α is chosen between 0 and 1. The process is circular if $\alpha = 1/\sqrt{2}$. The system coefficients were defined as

$$w_{\mathrm{opt,k}} = \beta(1 + \cos(2\pi(k-3)/5) - \boldsymbol{j}(1 + \cos(2\pi(k-3)/10))), \tag{2.57}$$

with $k = 1, 2, ..., 5$ and $\beta = 0.432$. The desired signal included a Gaussian noise with 20dB signal to noise ratio (SNR). The WL-LMS algorithm used a step-size $\mu_{\mathrm{WL}} = 0.04$. The RLS forgetting factor was chosen as $\nu = 0.999$ and the initial condition $\mathbf{\Phi}_{\mathrm{WL}}(0) = 0.01\mathbf{I}$. The regressor was generated using both $\alpha = 0.1$ and $\alpha = 1/\sqrt{2}$. Note that we define $d(n) = \mathcal{R}e\{\boldsymbol{w}_{\mathrm{opt}}^H\boldsymbol{x}(n)\}$, so the widely-linear solution achieves an MSE that is better than that of the SL solution, even when the regressor is proper (since the regressor and desired sequences will not be jointly second-order circular).

To ensure the same convergence characteristics, the step size of the RC-WL-LMS was chosen as $\mu_{\mathrm{RC}} = 2\mu_{\mathrm{WL}} = 0.08$ and the RC-WL-RLS initial condition used for $\mathbf{\Phi}_{\mathrm{RC}}(0)$ was $0.005\mathbf{I}$. In Figures 2 and 3, we observe different versions of the LMS and RLS algorithms, considering the SL, WL and RC-WL estimation. These figures show that the WL and RC-WL estimators are clearly better than the SL approach in this situation. It is also important to notice that the WL and RC-WL algorithms also achieve the same MSE levels, reaffirming the equivalence between those approaches.

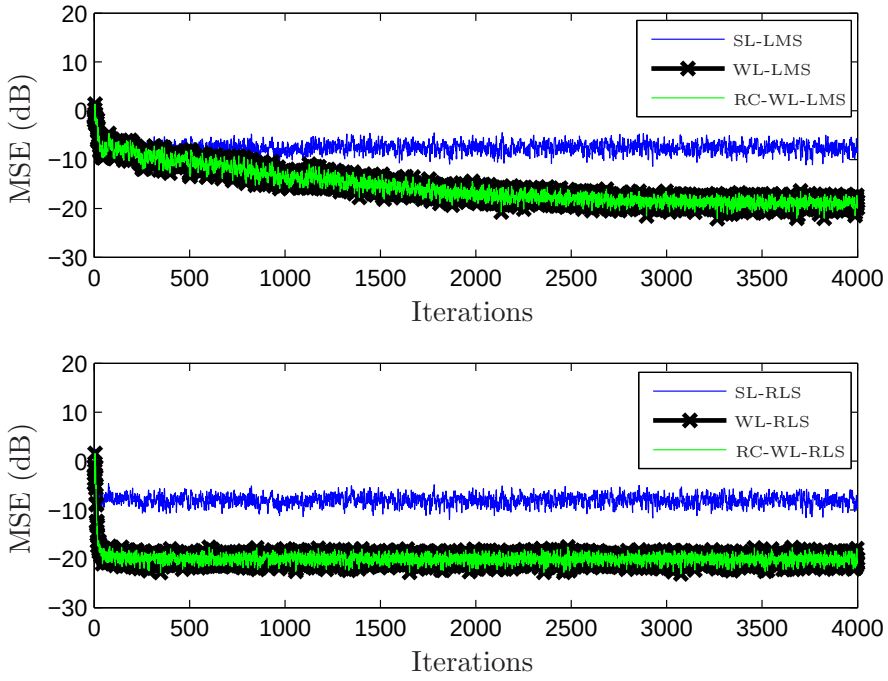

Figure 2: MSE for WL and RC-WL algorithms with circular input ($\alpha = 1/\sqrt{2}$). On the top: SL-LMS, WL-LMS and RC-WL-LMS ($\mu_{\text{RC}} = 2\mu_{\text{WL}} = 0.08$). On the bottom: SL-RLS, WL-RLS and RC-WL-RLS ($\mathbf{\Phi}_{\text{RC}}(0) = \mathbf{\Phi}_{\text{WL}}(0)/2 = 0.005\mathbf{I}$)

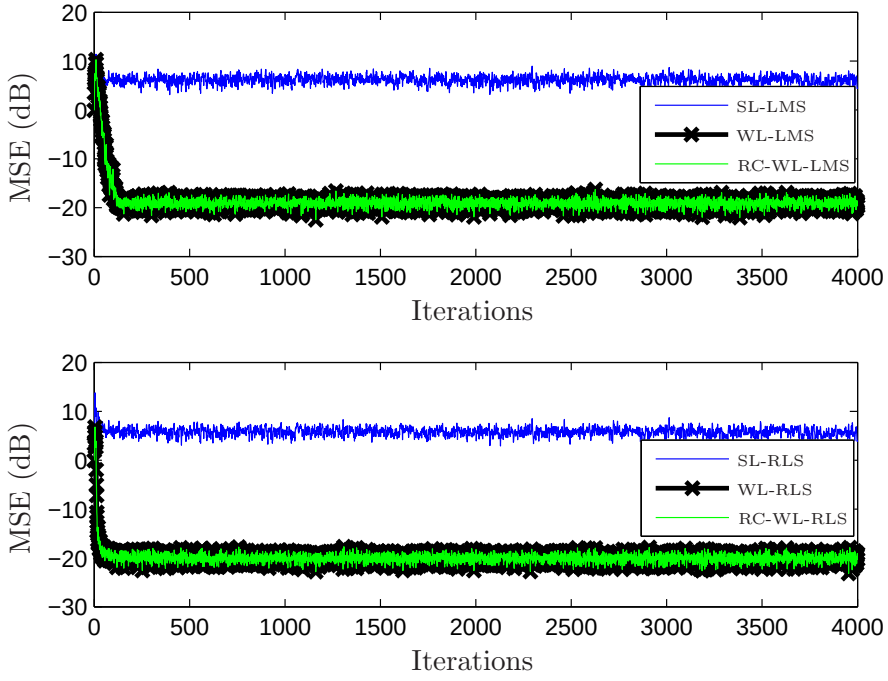

Figure 3: MSE for WL and RC-WL algorithms with non-circular input ($\alpha = 0.1$). On the top: SL-LMS, WL-LMS and RC-WL-LMS ($\mu_{\text{RC}} = 2\mu_{\text{WL}} = 0.08$). On the bottom: SL-RLS, WL-RLS and RC-WL-RLS ($\mathbf{\Phi}_{\text{RC}}(0) = \mathbf{\Phi}_{\text{WL}}(0)/2 = 0.005\mathbf{I}$)

2.4 Conclusions

In this chapter, we showed that, when using a real regressor vector constituted by the real and the imaginary parts of the complex data, it is possible to reduce the number of operations in WL algorithms. The RC-WL-LMS achieved the same complexity as the SL-LMS algorithm, which is almost a 50% reduction compared to the standard WL-LMS. Since the complexity of the standard RLS grows quadratically with the filter length, the reduction obtained by RC-WL-RLS is even larger, almost a factor of four. The simulations confirm the theoretical equivalence between the reduced-complexity algorithms and the WL ones when the input is proper or improper.

3 REDUCED-COMPLEXITY QUATERNION ADAPTIVE FILTERS

Quaternion numbers [3] were invented by Hamilton in the 19th century, as a generalization of complex numbers to a higher-dimensional domain. They consist of one real part and three imaginary elements, usually identified by $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$, where $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = -1$ and $\boldsymbol{ijk} = -1$. Quaternions appear in many fields, and their applications have been spreading recently, since they can be used to concisely describe multi-variable data.

Quaternion algebra is traditionally employed to represent rotations in control applications and in image processing. In the first case, quaternion algebra provides mathematical robustness to represent rotations. Its avoidance of the gimbal lock problem [47] (that appears when using Euler angle representations) is exploited by attitude control systems [48, 49]. On the other hand, in image processing, many techniques employ a quaternion-based model to describe images, allowing a concise representation of the color attributes in a single entity [50, 51], and leading to development of many quaternion-based tools, such as [52–54]. In recent applications, quaternions have been applied to help studying DNA structures [55], and to estimate quantities in neural networks [13], beamforming [56] and adaptive filtering [12, 14, 15, 18, 57, 58], among many others. The last field has experimented a large development lately, and a variety of algorithms has been proposed for multi-variable estimation.

In recent years, different definitions of quaternion differentiation were proposed ([59], [42] and [60]), giving rise to different quaternion adaptive algorithms. The first to be proposed was QLMS [14], whose update equation contains an extra term (when compared to the complex LMS update law [32]) to take into account the non-commutative nature of quaternion multiplication. Later, after a new definition of the derivative to account for quaternion involutions [59], a lower-cost and faster-converging technique emerged, iQLMS [42]. However, both QLMS and iQLMS are designed for $\mathbb{Q}$-proper data [61], for which the autocorrelation matrix is sufficient to assess full second-order statistics [61]. When the inputs are non $\mathbb{Q}$-proper, these strictly linear algorithms are not able to account for

full second-order statistics. In this case, widely-linear algorithms [15] can be applied to improve performance.

In the quaternion case, the definition of WL processes has led to the augmented QLMS [14] (which uses the original SL regressor vector and its conjugate as the WL input data) and to WL-QLMS [58], where the SL data vector and three quaternion involutions are the inputs. Later, WL-iQLMS [62] was also proposed as an improved WL-QLMS algorithm. However, for all these WL methods, the input vector is four times the length of the original SL data vector, and thus the computational cost is significantly higher.

Similar to the complex-valued WL algorithms presented in Chapter 2, quaternion WL techniques have redundance of second-order statistics in the autocorrelation matrix. In this chapter, we propose new WL quaternion algorithms which avoid redundant data and thus reduce the computational complexity. To obtain the new algorithms, we develop an unified description for the diverse quaternion gradients proposed in the literature, and we use it to study the convergence of WL-QLMS-based algorithms. We prove that a class of gradients which includes the i-gradient of [59] leads to the fastest-converging WL-QLMS algorithm, under some conditions on the correlation of the input data.

We also use the general quaternion gradient to develop a further reduced-complexity WL adaptive algorithm for real regressors – the RC-WL-iQLMS algorithm. For this purpose, we replace the original WL input vector by a real vector, obtained with a concatenation of the real and the three imaginary parts of the SL input. We show that using this approach, the redundant statistics are avoided, and many quaternion-quaternion computations are substituted by real-quaternion operations, leading to a low-cost technique. We also prove that the fastest-converging WL-QLMS-based algorithm with real-regressor vector corresponds to the four-channel LMS algorithm (4-Ch-LMS) written in the quaternion domain.

The contributions of this chapter can be summarized as follows:

1. We propose a general approach to describe the different quaternion gradients proposed in the literature, and show that different definitions for the quaternion derivative may lead to the same quaternion gradient.

2. We obtain a general update law, which describes all the QLMS algorithms proposed in the literature. We use it to study the convergence in the mean and in the variance of WL algorithms. We show that gradients similar to the i-gradient of [59] and to the gradient proposed in [60] lead to the fastest-converging WL-QLMS algorithms in

two situations: i) when at most two of the quaternion elements in the input vector are correlated; and ii) when the regressor vector is real.

3. Based on the general gradient, we propose a new technique which applies real regressor vector: the RC-WL-iQLMS algorithm, with one fourth the complexity of WL-iQLMS.

4. We show that the fastest-converging WL-QLMS algorithm with real regressor vector is our new algorithm, RC-WL-iQLMS. We also show that RC-WL-iQLMS corresponds to the 4-Ch-LMS algorithm written in the quaternion domain.

5. We develop a second-order model valid for any WL-QLMS algorithm with a real-regressor vector. The analysis is suitable for correlated and uncorrelated inputs. Concise equations to compute the EMSE (excess mean-square-error) and the MSD (mean square deviation) [45] are also derived.

The chapter is organized as follows. We present a brief review on quaternion algebra and Kronecker products (which are applied in our analysis) in Section 3.1. In Section 3.2, we present basic concepts of quaternion estimation and $\mathbb{Q}$-properness, while Section 3.3 introduces our general approach to write quaternion gradients. We present the new reduced-complexity algorithm in Section 3.3.2, and develop the analysis of QLMS-based algorithms in Sections 3.4.1 and 3.4.2. Simulations are presented in Section 3.5, and in Section 3.6 we conclude the chapter.

3.1 Preliminaries

In this section, we briefly summarize some properties of quaternion algebra and Kronecker products. These concepts simplify the analysis and the equations derived in this chapter.

3.1.1 Review on quaternion algebra

A quaternion q is defined as

$$q = q_{\mathrm{R}} + \boldsymbol{i} q_{\mathrm{I}} + \boldsymbol{j} q_{\mathrm{J}} + \boldsymbol{k} q_{\mathrm{K}},$$

where q_{R}, q_{I}, q_{J} and q_{K} are real numbers and $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ are the imaginary parts, which satisfy $\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1$. The main difference between a quaternion and a

complex number is that the multiplication in the quaternion field $\mathbb{Q}$ is not commutative, since [3]

$$\boldsymbol{ij} = -\boldsymbol{ji} = \boldsymbol{k}, \quad \boldsymbol{jk} = -\boldsymbol{kj} = \boldsymbol{i}, \quad \boldsymbol{ki} = -\boldsymbol{ik} = \boldsymbol{j},$$

such that for two quaternions q_1 and q_2, in general, $q_1 q_2 \neq q_2 q_1$.

Similar to complex algebra, the conjugate and the absolute value of a quaternion q are given by

$$q^* = q_{\mathrm{R}} - \boldsymbol{i} q_{\mathrm{I}} - \boldsymbol{j} q_{\mathrm{J}} - \boldsymbol{k} q_{\mathrm{K}}$$

and $|q| = \sqrt{q q^*}$, respectively. We can also define the following involutions of q[1]

$$\begin{aligned} q^i &\triangleq -\boldsymbol{i} q \boldsymbol{i} = q_{\mathrm{R}} + \boldsymbol{i} q_{\mathrm{I}} - \boldsymbol{j} q_{\mathrm{J}} - \boldsymbol{k} q_{\mathrm{K}} \\ q^j &\triangleq -\boldsymbol{j} q \boldsymbol{j} = q_{\mathrm{R}} - \boldsymbol{i} q_{\mathrm{I}} + \boldsymbol{j} q_{\mathrm{J}} - \boldsymbol{k} q_{\mathrm{K}} \\ q^k &\triangleq -\boldsymbol{k} q \boldsymbol{k} = q_{\mathrm{R}} - \boldsymbol{i} q_{\mathrm{I}} - \boldsymbol{j} q_{\mathrm{J}} + \boldsymbol{k} q_{\mathrm{K}}. \end{aligned}$$

The involutions are used in the definition of WL algorithms (e.g. [18], [58], [15]).

These are the main definitions of quaternion algebra used in this chapter. See reference [3] for more details.

3.1.2 Properties of Kronecker products

The Kronecker product [63] is an efficient manner to compactly represent some large matrices which have a block-structure. Given two matrices $\mathbf{A}$ and $\mathbf{B}$, the Kronecker product – which is represented by the operator $\otimes$ – is given by

$$\mathbf{A} \otimes \mathbf{B} \triangleq \begin{bmatrix} a_{11}\mathbf{B} & \dots & a_{1N}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{M1}\mathbf{B} & \dots & a_{MN}\mathbf{B} \end{bmatrix}, \tag{3.1}$$

For the purpose of the analyses performed in this chapter, the most relevant properties of Kronecker products are

1. $\mathbf{A} \otimes (\mathbf{B} + \mathbf{C}) = \mathbf{A} \otimes \mathbf{B} + \mathbf{A} \otimes \mathbf{C}$.
2. $\alpha (\mathbf{A} \otimes \mathbf{B}) = (\alpha \mathbf{A}) \otimes \mathbf{B} = \mathbf{A} \otimes (\alpha \mathbf{B})$.
3. $(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = \mathbf{AC} \otimes \mathbf{BD}$, where the number of rows of $\mathbf{C}$ ($\mathbf{B}$) and the number of columns of $\mathbf{A}$ ($\mathbf{D}$) are equal.
4. $(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}$.

[1] Note that the conjugate of a quaternion q is also an involution of q

5. $(\mathbf{A} \otimes \mathbf{B})^T = \mathbf{A}^T \otimes \mathbf{B}^T$.

6. $\mathrm{Tr}(\mathbf{A} \otimes \mathbf{B}) = \mathrm{Tr}(\mathbf{A})\mathrm{Tr}(\mathbf{B})$.

7. The eigenvalues of $(\mathbf{A} \otimes \mathbf{B})$, where $\mathbf{A}$ is $N \times N$ and $\mathbf{B}$ is $M \times M$, are given by $\lambda_{\mathbf{A}_l}\lambda_{\mathbf{B}_m}$, for $l = 1, 2, \ldots, N$ and $m = 1, 2, \ldots, M$, where $\lambda_{\mathbf{A}_l}$ and $\lambda_{\mathbf{B}_m}$ are the eigenvalues of $\mathbf{A}$ and $\mathbf{B}$, respectively.

These properties appear implicitly or explicitly in the analyses that follow. They make the equations easier to manipulate and simplify the interpretation of the resulting expressions.

3.2 Strictly-linear and widely-linear quaternion estimation

In the context of complex WL estimation, the concept of second-order circularity or properness is required to define when WL algorithms outperform SL ones (see Section 2.1.2.1). For quaternion quantities, this concept can be extended to $\mathbb{Q}$-properness. In this section, we present $\mathbb{Q}$-properness and compare SL and WL quaternion estimation, based on some results of [58] and [61].

Define the $N \times 1$ quaternion data vector

$$\boldsymbol{q}(n) = \boldsymbol{q}_{\mathrm{R}}(n) + \boldsymbol{i}\boldsymbol{q}_{\mathrm{I}}(n) + \boldsymbol{j}\boldsymbol{q}_{\mathrm{J}}(n) + \boldsymbol{k}\boldsymbol{q}_{\mathrm{K}}(n), \tag{3.2}$$

where $\boldsymbol{q}_{\mathrm{R}}(n)$, $\boldsymbol{q}_{\mathrm{I}}(n)$, $\boldsymbol{q}_{\mathrm{J}}(n)$ and $\boldsymbol{q}_{\mathrm{K}}(n)$ are real vectors. Given a desired quaternion sequence $d(n)$, the problem solved by a strictly linear estimator is the computation of $\boldsymbol{w}_{\mathrm{SL}}(n)$ in

$$y_{\mathrm{SL}}(n) = \boldsymbol{w}_{\mathrm{SL}}^H(n)\boldsymbol{q}(n) \tag{3.3}$$

which minimizes the mean-square error $E\{|e_{\mathrm{SL}}(n)|^2\}$, where

$$e_{\mathrm{SL}}(n) = d(n) - y_{\mathrm{SL}}(n). \tag{3.4}$$

From the orthogonality principle [32], it must be true that

$$E\{\boldsymbol{q}(n)e_{\mathrm{SL}}^*(n)\} = \mathbf{0}, \tag{3.5}$$

and substituting eq. (3.4) in (3.5), we get

$$E\{\boldsymbol{q}(n)d^*(n)\} = E\{\boldsymbol{q}(n)y_{\mathrm{SL}}^*(n)\}. \tag{3.6}$$

Using eq. (3.3) in (3.6), $\boldsymbol{w}_{\mathrm{SL}}(n)$ must satisfy

$$\boldsymbol{C}_{\boldsymbol{q}}\boldsymbol{w}_{\mathrm{SL}}(n) = \boldsymbol{p}_{\boldsymbol{q}}, \tag{3.7}$$

where $\boldsymbol{C}_{\boldsymbol{q}} = E\{\boldsymbol{q}(n)\boldsymbol{q}^H(n)\}$ is the autocorrelation matrix and $\boldsymbol{p}_{\boldsymbol{q}} = E\{\boldsymbol{q}(n)d^*(n)\}$ is the cross-correlation vector [1].

For a quaternion WL estimator, the data vector is modified to account for the involutions [12], and it is given by[2]

$$\boldsymbol{q}_{\mathrm{WL}}(n) = \mathrm{col}(\, \boldsymbol{q}(n), \quad \boldsymbol{q}^i(n), \quad \boldsymbol{q}^j(n) \quad \boldsymbol{q}^k(n)\,), \tag{3.8}$$

which is four times the length of the original vector $\boldsymbol{q}(n)$. For this approach, one must find the vector $\boldsymbol{w}_{\mathrm{WL}}(n)$ which minimizes the MSE condition $E\{|e_{\mathrm{WL}}(n)|^2\}$, where

$$e_{\mathrm{WL}}(n) = d(n) - y_{\mathrm{WL}}(n), \tag{3.9}$$

$$y_{\mathrm{WL}}(n) = \boldsymbol{w}_{\mathrm{WL}}^H(n)\boldsymbol{q}_{\mathrm{WL}}(n). \tag{3.10}$$

Again, the orthogonality condition implies that $e_{\mathrm{WL}}(n)$ must be orthogonal to all involutions, i.e.,

$$E\{\boldsymbol{q}_{\mathrm{WL}}(n)e_{\mathrm{WL}}^*(n)\} = \boldsymbol{0}. \tag{3.11}$$

Using eqs. (3.9), (3.10) and (3.11), we obtain

$$\boldsymbol{C}_{\mathrm{WL}}\boldsymbol{w}_{\mathrm{WL}}(n) = \boldsymbol{p}_{\mathrm{WL}}, \tag{3.12}$$

and

$$\boldsymbol{C}_{\mathrm{WL}} = E\{\boldsymbol{q}_{\mathrm{WL}}(n)\boldsymbol{q}_{\mathrm{WL}}^H(n)\} = \begin{bmatrix} \boldsymbol{C}_{\boldsymbol{q}} & \boldsymbol{C}_{\boldsymbol{q}\boldsymbol{q}^i} & \boldsymbol{C}_{\boldsymbol{q}\boldsymbol{q}^j} & \boldsymbol{C}_{\boldsymbol{q}\boldsymbol{q}^k} \\ \boldsymbol{C}_{\boldsymbol{q}\boldsymbol{q}^i}^H & \boldsymbol{C}_{\boldsymbol{q}^i} & \boldsymbol{C}_{\boldsymbol{q}^i\boldsymbol{q}^j} & \boldsymbol{C}_{\boldsymbol{q}^i\boldsymbol{q}^k} \\ \boldsymbol{C}_{\boldsymbol{q}\boldsymbol{q}^j}^H & \boldsymbol{C}_{\boldsymbol{q}^i\boldsymbol{q}^j}^H & \boldsymbol{C}_{\boldsymbol{q}^j} & \boldsymbol{C}_{\boldsymbol{q}^j\boldsymbol{q}^k} \\ \boldsymbol{C}_{\boldsymbol{q}\boldsymbol{q}^k}^H & \boldsymbol{C}_{\boldsymbol{q}^i\boldsymbol{q}^k}^H & \boldsymbol{C}_{\boldsymbol{q}^j\boldsymbol{q}^k}^H & \boldsymbol{C}_{\boldsymbol{q}^k} \end{bmatrix}, \tag{3.13}$$

where

$$\boldsymbol{C}_{\boldsymbol{\alpha}} = E\{\boldsymbol{\alpha}\boldsymbol{\alpha}^H\}, \text{ for } \boldsymbol{\alpha} \in \{\boldsymbol{q}(n), \boldsymbol{q}^i(n), \boldsymbol{q}^j(n), \boldsymbol{q}^k(n)\} \tag{3.14}$$

and

$$\boldsymbol{C}_{\boldsymbol{\alpha}\boldsymbol{\beta}} = E\{\boldsymbol{\alpha}\boldsymbol{\beta}^H\}, \text{ for } \boldsymbol{\alpha}, \boldsymbol{\beta} \in \{\boldsymbol{q}(n), \boldsymbol{q}^i(n), \boldsymbol{q}^j(n), \boldsymbol{q}^k(n)\} \text{ and } \boldsymbol{\alpha} \neq \boldsymbol{\beta}. \tag{3.15}$$

The matrices $\boldsymbol{C}_{\boldsymbol{\alpha}\boldsymbol{\beta}}$ are the cross-correlation terms between $\boldsymbol{q}(n)$ and their involutions.

[2]Note that other sets of four different involutions of $\boldsymbol{q}(n)$ can be chosen, since they provide similar estimation performance [42].

$\boldsymbol{p}_{\mathrm{WL}}$ is given by

$$\boldsymbol{p}_{\mathrm{WL}} = \mathrm{col}(\ \boldsymbol{p}_{\boldsymbol{q}},\ \ \boldsymbol{p}_{\boldsymbol{q}^i},\ \ \boldsymbol{p}_{\boldsymbol{q}^j},\ \ \boldsymbol{p}_{\boldsymbol{q}^k}\ ) \tag{3.16}$$

and $\boldsymbol{p}_{\boldsymbol{\alpha}} = E\{\boldsymbol{\alpha} d^*(n)\}$, $\boldsymbol{\alpha} \in \{\boldsymbol{q}(n), \boldsymbol{q}^i(n), \boldsymbol{q}^j(n), \boldsymbol{q}^k(n)\}$.

$\boldsymbol{C}_{\mathrm{WL}}$ and $\boldsymbol{p}_{\mathrm{WL}}$ have four times the dimension of their SL counterparts. Using (3.12), we are able to exploit full second-order statistics of the input data, which may not be fully available in (3.7). Equation (3.7) takes advantage of full second-order information only if the data pair $\{\boldsymbol{q}(n), d(n)\}$ captures all relevant second-order statistics. In this case, $\boldsymbol{q}(n)$ and $d(n)$ are said to be jointly $\mathbb{Q}$-proper. This happens under the conditions described below.

3.2.0.1 $\mathbb{Q}$-properness

Vector $\boldsymbol{q}(n)$ is $\mathbb{Q}$-proper if all the cross-correlation matrices vanish [61], i.e.,

$$\boldsymbol{C}_{\boldsymbol{q}\boldsymbol{q}^i} = \boldsymbol{C}_{\boldsymbol{q}\boldsymbol{q}^j} = \boldsymbol{C}_{\boldsymbol{q}\boldsymbol{q}^k} = \boldsymbol{C}_{\boldsymbol{q}^i\boldsymbol{q}^j} = \boldsymbol{C}_{\boldsymbol{q}^i\boldsymbol{q}^k} = \boldsymbol{C}_{\boldsymbol{q}^j\boldsymbol{q}^k} = \boldsymbol{0}. \tag{3.17}$$

When (3.17) holds, eq. (3.13) can be expressed as

$$\boldsymbol{C}_{\mathrm{WL}} = \begin{bmatrix} \boldsymbol{C}_{\boldsymbol{q}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{C}_{\boldsymbol{q}^i} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{C}_{\boldsymbol{q}^j} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{C}_{\boldsymbol{q}^k} \end{bmatrix}. \tag{3.18}$$

Recalling that

$$\boldsymbol{C}_{\boldsymbol{q}^\alpha} = E\{\boldsymbol{q}^\alpha \boldsymbol{q}^{\alpha H}\} = E\{-\alpha \boldsymbol{q} \alpha (-\alpha) \boldsymbol{q}^H \alpha\} = -\alpha E\{\boldsymbol{q}\boldsymbol{q}^H\}\alpha = -\alpha \boldsymbol{C}_{\mathbf{q}} \alpha, \tag{3.19}$$

for $\alpha \in \{\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}\}$, eq. (3.18) reduces to

$$\boldsymbol{C}_{\mathrm{WL}} = \boldsymbol{T}(\boldsymbol{I}_4 \otimes \boldsymbol{C}_{\boldsymbol{q}})\boldsymbol{T}^H, \tag{3.20}$$

where $\boldsymbol{T} = \mathrm{diag}(\ 1,\ \ -\boldsymbol{i},\ \ -\boldsymbol{j},\ \ -\boldsymbol{k}\ ) \otimes \boldsymbol{I}_N$.

Notice that for a $\mathbb{Q}$-proper input, only the SL autocorrelation matrix $\boldsymbol{C}_{\boldsymbol{q}}$ is required to access the second-order statistics of $\boldsymbol{q}(n)$, similarly to the complex-proper case reported in Chapter 2.1. However, only this does not guarantee that full second-order information of the pair $\{\boldsymbol{q}(n), d(n)\}$ is available for the SL estimator: $d(n)$ and $\boldsymbol{q}(n)$ must be jointly $\mathbb{Q}$-proper for this to hold.

3.2.0.2 Joint $\mathbb{Q}$-properness

$\mathbb{Q}$-properness is related only to the input signal $\boldsymbol{q}(n)$. However, $d(n)$ and $\boldsymbol{q}(n)$ can also be jointly $\mathbb{Q}$-proper [61], which implies the additional restrictions

$$E\{\boldsymbol{q}^{\alpha}(n)d^{*}(n)\} = \mathbf{0},\ \forall\alpha \in \{\boldsymbol{i},\ \boldsymbol{j},\ \boldsymbol{k}\}. \tag{3.21}$$

Using eq. (3.21) in (3.16), we obtain

$$\boldsymbol{p}_{\mathrm{WL}} = \mathrm{col}(1,\ 0,\ 0,\ 0) \otimes \boldsymbol{p}_{\boldsymbol{q}},$$

and the system of equations that must be solved by the WL estimator simplifies to

$$\begin{bmatrix} \boldsymbol{C}_{\boldsymbol{q}} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{C}_{\boldsymbol{q}^{i}} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{C}_{\boldsymbol{q}^{j}} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{C}_{\boldsymbol{q}^{k}} \end{bmatrix} \boldsymbol{w}_{\mathrm{WL}}(n) = \begin{bmatrix} \boldsymbol{p}_{\boldsymbol{q}} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}. \tag{3.22}$$

Since $\boldsymbol{C}_{\mathrm{WL}}$ is a block-diagonal matrix, we can define

$$\boldsymbol{w}_{\mathrm{WL}} = \mathrm{col}(\ \boldsymbol{w}_{\mathrm{WL},1}(n),\quad \boldsymbol{w}_{\mathrm{WL},2}(n)\quad \boldsymbol{w}_{\mathrm{WL},3}(n)\quad \boldsymbol{w}_{\mathrm{WL},4}(n)\ ),$$

and compute the solution to (3.22) using four lower-dimension systems of equations, i.e.,

$$\boldsymbol{C}_{\boldsymbol{q}}\boldsymbol{w}_{\mathrm{WL},1}(n) = \boldsymbol{p},\quad \boldsymbol{C}_{\boldsymbol{q}^{i}}\boldsymbol{w}_{\mathrm{WL},2}(n) = \mathbf{0},\quad \boldsymbol{C}_{\boldsymbol{q}^{j}}\boldsymbol{w}_{\mathrm{WL},3}(n) = \mathbf{0},\quad \text{and } \boldsymbol{C}_{\boldsymbol{q}^{k}}\boldsymbol{w}_{\mathrm{WL},4}(n) = \mathbf{0}. \tag{3.23}$$

In the usual case, where $\boldsymbol{C}_{\boldsymbol{q}}$ is full-rank, $\boldsymbol{w}_{\mathrm{WL},2}(n) = \boldsymbol{w}_{\mathrm{WL},3}(n) = \boldsymbol{w}_{\mathrm{WL},4}(n) = \mathbf{0}$, so that $\boldsymbol{w}_{\mathrm{WL}}$ must satisfy (3.7). This result shows that the SL and WL approaches are equivalent in terms of MSE performance when the input and the desired signal are jointly $\mathbb{Q}$-proper. However, from the point of view of implementation of algorithms, a WL approach is not attractive in this case, since its computational complexity is much higher than that of a SL technique. The EMSE is also expected to increase in this case. The EMSE is computed as $\mathrm{Tr}(\boldsymbol{w}_{\mathrm{WL}}(n)\boldsymbol{w}_{\mathrm{WL}}^{H}(n))$. Since the algorithm will try to estimate each entry of $\boldsymbol{w}_{\mathrm{WL}}(n)$, nonzero values can be obtained for weights that should be zero, so that errors can be added to EMSE expression, increasing its value (when compared to the SL approach). Finally, the convergence rate is also expected to be slower, since it estimates four times the number of parameters computed by an SL approach.

In summary, when $\boldsymbol{q}(n)$ and $d(n)$ are jointly $\mathbb{Q}$-proper, SL and WL approaches achieve the same MSE performance, but SL techniques are preferred since they are easier to

implement and converge faster. On the other hand, when joint $\mathbb{Q}$-properness does not hold, a WL-based technique can extract full second-order statistics of the input data to solve (3.12), improving the MSE performance.

3.2.0.3 Widely-linear estimation using real regressor vector

In this chapter, we propose low-cost WL techniques using a real regressor vector instead of the original WL vector $\boldsymbol{q}_{\mathrm{WL}}(n)$ of eq. (3.8). In this section, we explain the advantage of this approach to avoid redundant second-order statistics in the correlation matrix.

To show the data redundance in $\boldsymbol{C}_{\mathrm{WL}}$, consider its first block-element $\boldsymbol{C}_{\boldsymbol{q}}$. This matrix is a sum of the correlations among the quaternion parts of $\boldsymbol{q}(n)$, and can be expressed as[3]

$$\begin{aligned}\boldsymbol{C}_{\boldsymbol{q}} =& E\{\boldsymbol{q}\boldsymbol{q}^H\} = E\{\boldsymbol{q}_{\mathrm{R}}\boldsymbol{q}_{\mathrm{R}}^T + \boldsymbol{q}_{\mathrm{I}}\boldsymbol{q}_{\mathrm{I}}^T + \boldsymbol{q}_{\mathrm{J}}\boldsymbol{q}_{\mathrm{J}}^T + \boldsymbol{q}_{\mathrm{K}}\boldsymbol{q}_{\mathrm{K}}^T\} + \boldsymbol{i}E\{-\boldsymbol{q}_{\mathrm{R}}\boldsymbol{q}_{\mathrm{I}}^T + \boldsymbol{q}_{\mathrm{I}}\boldsymbol{q}_{\mathrm{R}}^T - \boldsymbol{q}_{\mathrm{J}}\boldsymbol{q}_{\mathrm{K}}^T + \boldsymbol{q}_{\mathrm{K}}\boldsymbol{q}_{\mathrm{J}}^T\} \\ &+ \boldsymbol{j}E\{-\boldsymbol{q}_{\mathrm{R}}\boldsymbol{q}_{\mathrm{J}}^T + \boldsymbol{q}_{\mathrm{I}}\boldsymbol{q}_{\mathrm{K}}^T + \boldsymbol{q}_{\mathrm{J}}\boldsymbol{q}_{\mathrm{R}}^T - \boldsymbol{q}_{\mathrm{K}}\boldsymbol{q}_{\mathrm{K}}^T\} + \boldsymbol{k}E\{-\boldsymbol{q}_{\mathrm{R}}\boldsymbol{q}_{\mathrm{K}}^T - \boldsymbol{q}_{\mathrm{I}}\boldsymbol{q}_{\mathrm{J}}^T + \boldsymbol{q}_{\mathrm{J}}\boldsymbol{q}_{\mathrm{I}}^T + \boldsymbol{q}_{\mathrm{K}}\boldsymbol{q}_{\mathrm{R}}^T\}.\end{aligned}$$

All the correlation and cross-correlation terms obtained with $\boldsymbol{q}_{\mathrm{R}}$, $\boldsymbol{q}_{\mathrm{I}}$, $\boldsymbol{q}_{\mathrm{J}}$ and $\boldsymbol{q}_{\mathrm{K}}$ appear in $\boldsymbol{C}_{\boldsymbol{q}}$, but cannot be recovered individually from $\boldsymbol{C}_{\boldsymbol{q}}$ and (3.15). On the other hand, using the real regressor vector

$$\boldsymbol{q}_{\mathrm{RC}}(n) = \mathrm{col}\left(\ \boldsymbol{q}_{\mathrm{R}}(n),\ \ \boldsymbol{q}_{\mathrm{I}}(n),\ \ \boldsymbol{q}_{\mathrm{J}}(n),\ \ \boldsymbol{q}_{\mathrm{K}}(n)\ \right), \tag{3.24}$$

obtained stacking up the real and imaginary elements of $\boldsymbol{q}(n)$, the corresponding correlation matrix is given by

$$\boldsymbol{C}_{\mathrm{RC}} = E\{\boldsymbol{q}_{\mathrm{RC}}(n)\boldsymbol{q}_{\mathrm{RC}}(n)\} = \begin{bmatrix} \mathbf{C}_{\mathrm{RC_R}} & \mathbf{C}_{\mathrm{RC_{RI}}} & \mathbf{C}_{\mathrm{RC_{RJ}}} & \mathbf{C}_{\mathrm{RC_{RK}}} \\ \mathbf{C}_{\mathrm{RC_{RI}}}^T & \mathbf{C}_{\mathrm{RC_I}} & \mathbf{C}_{\mathrm{RC_{IJ}}} & \mathbf{C}_{\mathrm{RC_{RK}}} \\ \mathbf{C}_{\mathrm{RC_{RJ}}}^T & \mathbf{C}_{\mathrm{RC_{IJ}}}^T & \mathbf{C}_{\mathrm{RC_J}} & \mathbf{C}_{\mathrm{RC_{JK}}} \\ \mathbf{C}_{\mathrm{RC_{RK}}}^T & \mathbf{C}_{\mathrm{RC_{IK}}}^T & \mathbf{C}_{\mathrm{RC_{JK}}}^T & \mathbf{C}_{\mathrm{RC_K}} \end{bmatrix}, \tag{3.25}$$

where

$$\begin{aligned} \boldsymbol{C}_{\mathrm{RC_R}} &= E\{\boldsymbol{q}_{\mathrm{R}}(n)\boldsymbol{q}_{\mathrm{R}}^T(n)\}, & \boldsymbol{C}_{\mathrm{RC_I}} &= E\{\boldsymbol{q}_{\mathrm{I}}(n)\boldsymbol{q}_{\mathrm{I}}^T(n)\}, \\ \boldsymbol{C}_{\mathrm{RC_J}} &= E\{\boldsymbol{q}_{\mathrm{J}}(n)\boldsymbol{q}_{\mathrm{J}}^T(n)\}, & \boldsymbol{C}_{\mathrm{RC_K}} &= E\{\boldsymbol{q}_{\mathrm{K}}(n)\boldsymbol{q}_{\mathrm{K}}^T(n)\}, \\ \boldsymbol{C}_{\mathrm{RC_{RI}}} &= E\{\boldsymbol{q}_{\mathrm{R}}(n)\boldsymbol{q}_{\mathrm{I}}^T(n)\}, & \boldsymbol{C}_{\mathrm{RC_{JK}}} &= E\{\boldsymbol{q}_{\mathrm{J}}(n)\boldsymbol{q}_{\mathrm{K}}^T(n)\}, \\ \boldsymbol{C}_{\mathrm{RC_{RJ}}} &= E\{\boldsymbol{q}_{\mathrm{R}}(n)\boldsymbol{q}_{\mathrm{J}}^T(n)\}, & \boldsymbol{C}_{\mathrm{RC_{IK}}} &= E\{\boldsymbol{q}_{\mathrm{I}}(n)\boldsymbol{q}_{\mathrm{K}}^T(n)\}, \\ \boldsymbol{C}_{\mathrm{RC_{RK}}} &= E\{\boldsymbol{q}_{\mathrm{R}}(n)\boldsymbol{q}_{\mathrm{K}}^T(n)\}, & \boldsymbol{C}_{\mathrm{RC_{IJ}}} &= E\{\boldsymbol{q}_{\mathrm{I}}(n)\boldsymbol{q}_{\mathrm{J}}^T(n)\}. \end{aligned} \tag{3.26}$$

[3] We drop time coefficients to shorten the notation.

and $\boldsymbol{C}_{\mathrm{RC}}$ is real and contains full second-order statistics of $\boldsymbol{q}(n)$.

Using $\boldsymbol{q}_{\mathrm{RC}}(n)$ to design quaternion WL algorithms, we do not loose information, and we also obtain techniques for which the real regressor allows the substitution of many quaternion-quaternion operations by real-quaternion computations, less costly to compute. We exploit it in Section 3.3.2 to develop low-cost WL algorithms.

3.3 Quaternion gradients

In this section, we propose a general definition for quaternion derivatives, which is used to study the convergence of QLMS-based algorithms and to obtain the fastest-converging WL-QLMS algorithm in two situations: 1) when at most two of the quaternion elements in the regressor are correlated; and 2) when the WL algorithms use the real regressor vector $\boldsymbol{q}_{\mathrm{RC}}(n)$. We also use the general gradient description to develop mean and mean-square analyses for QLMS-based algorithms. This approach is later particularized to real-regressor vector WL-QLMS algorithms, leading to accurate tools to design the algorithms and evaluate their performance.

Throughout this section, we do not add indexes to identify SL and WL quantities, since the derivation is applicable to both approaches. We use $\boldsymbol{\varphi}(n)$ to define a general quaternion regressor vector with dimension M, where $M = N$ in the SL approach and $M = 4N$ in the WL case.

To obtain a general quaternion gradient, start with the definition of the cost function

$$f(\boldsymbol{w}) = e(n)e^*(n),$$

where $e(n) = d(n) - \boldsymbol{w}^H(n)\boldsymbol{\varphi}(n)$. Regardless of the method to compute the gradient, and based on the isomorphism between $\mathbb{Q}$ and $\mathbb{R}^4$, the quaternion gradients proposed in the literature have the general form [59, 60]

$$\nabla_{\boldsymbol{w}} f = a\frac{\partial f}{\partial \boldsymbol{w}_{\mathrm{R}}} + \boldsymbol{i}b\frac{\partial f}{\partial \boldsymbol{w}_{\mathrm{I}}} + \boldsymbol{j}c\frac{\partial f}{\partial \boldsymbol{w}_{\mathrm{J}}} + \boldsymbol{k}d\frac{\partial f}{\partial \boldsymbol{w}_{\mathrm{K}}}, \tag{3.27}$$

for real $\{a, b, c, d\}$, and where

$$\frac{\partial f}{\partial \boldsymbol{w}_\alpha} = \frac{\partial e(n)}{\partial \boldsymbol{w}_\alpha}e^*(n) + e(n)\frac{\partial e^*(n)}{\partial \boldsymbol{w}_\alpha}, \tag{3.28}$$

and $\alpha \in \{R, I, J, K\}$. Using eq. (3.28) in (3.27), and defining

$$\begin{aligned} g &= (a + b + c + d) \\ h &= (a - b - c - d) \end{aligned} \tag{3.29}$$

eq. (3.27) becomes

$$\nabla_{\boldsymbol{w}} f = -g\boldsymbol{\varphi}(n)e^*_{\mathrm{WL}}(n) - he_{\mathrm{WL}}(n)\boldsymbol{\varphi}^*(n). \tag{3.30}$$

From (3.30), one can check that the quaternion gradient of [59] is obtained when $g = 1/2$ and $h = -1/4$, and that the i-gradient of [42] appears when $g = 3/4$ and $h = 0$. We also note that the recently proposed quaternion gradient [60] is obtained when $g = 2$ and $h = 0$. Considering eq. (3.29), it is easy to note that the same g and h can be obtained for different values of a, b, c and d. Moreover, note that gradient-based adaptive algorithms have a general update law given by [32]

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \mu\nabla_{\boldsymbol{w}} f, \tag{3.31}$$

where μ is the step size. In this case, from the point of view of the algorithm, it is not important if the gradient uses g and h or scaled versions of them – as long as they are both scaled by the same value – since the scale factor can be absorbed by μ. This fact emphasizes that different gradients and quaternion derivatives can be applied to define the same QLMS-based algorithm.

Substituting eq. (3.30) in (3.31), we define the general form of a QLMS-like algorithm, that is

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu\left[g\boldsymbol{\varphi}(n)e^*(n) + he(n)\boldsymbol{\varphi}^*(n)\right]. \tag{3.32}$$

Note that using $g = 1/2$ and $h = -1/4$, the update equation of the WL-QLMS algorithm is obtained from (3.32) when $\boldsymbol{\varphi}(n) = \boldsymbol{q}_{\mathrm{WL}}(n)$ (see Table 7). When $g = 3/4$ and $h = 0$, we derive from (3.32) the update equation of WL-iQLMS (see Table 8).

Table 7: WL-QLMS algorithm

$$\begin{array}{|l|}\hline \hat{d}_{\mathrm{WL}}(n) = \boldsymbol{w}^H_{\mathrm{WL}}(n)\boldsymbol{q}_{\mathrm{WL}}(n) \\ e_{\mathrm{WL}}(n) = d(n) - \hat{d}_{\mathrm{WL}}(n) \\ \boldsymbol{w}_{\mathrm{WL}}(n+1) = \boldsymbol{w}_{\mathrm{WL}}(n) + \mu_{\mathrm{WL}}\left[\boldsymbol{q}_{\mathrm{WL}}(n)\dfrac{e^*_{\mathrm{WL}}(n)}{2} - \dfrac{e_{\mathrm{WL}}(n)}{4}\boldsymbol{q}^*_{\mathrm{WL}}(n)\right] \\ \hline \end{array}$$

Recalling that

$$g\boldsymbol{\varphi}(n)e^*(n) + he(n)\boldsymbol{\varphi}^*(n) = (g+h)\boldsymbol{\varphi}(n)e^*(n) - 2h\mathcal{I}m\{\boldsymbol{\varphi}(n)e^*(n)\}, \tag{3.33}$$

Table 8: WL-iQLMS algorithm

$$\boxed{\begin{array}{l}\hat{d}_{\mathrm{iQ}}(n) = \boldsymbol{w}_{\mathrm{iQ}}^H(n)\boldsymbol{q}_{\mathrm{WL}}(n) \\ e_{\mathrm{iQ}}(n) = d(n) - \hat{d}_{\mathrm{iQ}}(n) \\ \boldsymbol{w}_{\mathrm{iQ}}(n+1) = \boldsymbol{w}_{\mathrm{iQ}}(n) + \dfrac{3}{4}\mu_{\mathrm{iQ}}\boldsymbol{q}_{\mathrm{WL}}(n)e_{\mathrm{iQ}}^*(n)\end{array}}$$

and using eq. (3.33) in (3.32), one gets

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) + \mu\left[(g+h)\boldsymbol{\varphi}(n)e^*(n) - 2h\mathcal{I}m\{\boldsymbol{\varphi}(n)e^*(n)\}\right]. \tag{3.34}$$

In order to study the mean behavior of (3.34), define the optimum set of coefficients $\boldsymbol{w}_{\mathrm{opt}}$, and assume that $d(n)$ can be modeled as

$$d(n) = \boldsymbol{w}_{\mathrm{opt}}^H\boldsymbol{\varphi}(n) + v(n), \tag{3.35}$$

where the elements of $v(n)$ are i.i.d., independent of $\boldsymbol{\varphi}(n)$, $E\{v(n)\} = 0$, and $E\{|v(n)|^2\} = \sigma_v^2$. Subtracting (3.34) from $\boldsymbol{w}_{\mathrm{opt}}$, one gets

$$\tilde{\boldsymbol{w}}(n+1) = \tilde{\boldsymbol{w}}(n) - \mu[(g+h)\boldsymbol{\varphi}(n)e^*(n) - 2h\mathcal{I}m\{\boldsymbol{\varphi}(n)e^*(n)\}], \tag{3.36}$$

where

$$\tilde{\boldsymbol{w}}(n) = \boldsymbol{w}_{\mathrm{opt}} - \boldsymbol{w}(n). \tag{3.37}$$

Using (3.35) in (3.36), we obtain

$$\begin{aligned}\tilde{\boldsymbol{w}}(n+1) =& \tilde{\boldsymbol{w}}(n) - \mu(g+h)\left[\boldsymbol{\varphi}(n)\boldsymbol{\varphi}^H(n)\tilde{\boldsymbol{w}}(n) + \boldsymbol{\varphi}(n)v^*(n)\right] \\ &+ 2\mu h\mathcal{I}m\left\{\boldsymbol{\varphi}(n)\boldsymbol{\varphi}^H(n)\tilde{\boldsymbol{w}}(n) + \boldsymbol{\varphi}(n)v^*(n)\right\}.\end{aligned} \tag{3.38}$$

In terms of $\tilde{\mathbf{w}}(n)$, this is a nonlinear recursion (due to the $\mathcal{I}m\{\cdot\}$ operator). In order to obtain a linear recursion, we rewrite the recursion separating the real and the imaginary parts of $\boldsymbol{\varphi}(n)$, defining a set of extended variables. Define the extended $4M \times 1$ $\tilde{\boldsymbol{w}}(n)$

$$\tilde{\boldsymbol{w}}_{\mathrm{ext}}(n) = \mathrm{col}(\tilde{\boldsymbol{w}}_{\mathrm{R}}(n),\ \tilde{\boldsymbol{w}}_{\mathrm{I}}(n),\ \tilde{\boldsymbol{w}}_{\mathrm{J}}(n),\ \tilde{\boldsymbol{w}}_{\mathrm{K}}(n))$$

and the extended version of $\boldsymbol{\varphi}(n)\boldsymbol{\varphi}^H(n)\tilde{\boldsymbol{w}}(n)$, which is given by $\tilde{\boldsymbol{C}}_{\mathrm{ext}}\tilde{\boldsymbol{w}}_{\mathrm{ext}}(n)$, with

$$\tilde{\boldsymbol{C}}_{\mathrm{ext}} = \begin{bmatrix} \tilde{\boldsymbol{\mathcal{R}}} & \tilde{\boldsymbol{\mathcal{I}}}^T & \tilde{\boldsymbol{\mathcal{J}}}^T & \tilde{\boldsymbol{\mathcal{K}}}^T \\ \tilde{\boldsymbol{\mathcal{I}}} & \tilde{\boldsymbol{\mathcal{R}}} & \tilde{\boldsymbol{\mathcal{K}}}^T & \tilde{\boldsymbol{\mathcal{J}}} \\ \tilde{\boldsymbol{\mathcal{J}}} & \tilde{\boldsymbol{\mathcal{K}}} & \tilde{\boldsymbol{\mathcal{R}}} & \tilde{\boldsymbol{\mathcal{I}}}^T \\ \tilde{\boldsymbol{\mathcal{K}}} & \tilde{\boldsymbol{\mathcal{J}}}^T & \tilde{\boldsymbol{\mathcal{I}}} & \tilde{\boldsymbol{\mathcal{R}}} \end{bmatrix} \in \mathbb{R}^{4M}. \tag{3.39}$$

Matrix $\tilde{\boldsymbol{\mathcal{R}}}$ contains the real elements of $\boldsymbol{\varphi}(n)\boldsymbol{\varphi}^H(n)$, while $\tilde{\boldsymbol{\mathcal{I}}}$, $\tilde{\boldsymbol{\mathcal{J}}}$ and $\tilde{\boldsymbol{\mathcal{K}}}$ are the imaginary

parts for $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$, respectively. $\tilde{\boldsymbol{\mathcal{R}}}$ is a symmetric matrix, and $\tilde{\boldsymbol{\mathcal{I}}} = -\tilde{\boldsymbol{\mathcal{I}}}^T$, $\tilde{\boldsymbol{\mathcal{J}}} = -\tilde{\boldsymbol{\mathcal{J}}}^T$ and $\tilde{\boldsymbol{\mathcal{K}}} = -\tilde{\boldsymbol{\mathcal{K}}}^T$. These matrices are given by

$$\begin{aligned}
\tilde{\boldsymbol{\mathcal{R}}} &= \boldsymbol{\varphi}_{\mathrm{R}}\boldsymbol{\varphi}_{\mathrm{R}}^T + \boldsymbol{\varphi}_{\mathrm{I}}\boldsymbol{\varphi}_{\mathrm{I}}^T + \boldsymbol{\varphi}_{\mathrm{J}}\boldsymbol{\varphi}_{\mathrm{J}}^T + \boldsymbol{\varphi}_{\mathrm{K}}\boldsymbol{\varphi}_{\mathrm{K}}^T, \\
\tilde{\boldsymbol{\mathcal{I}}} &= -\boldsymbol{\varphi}_{\mathrm{R}}\boldsymbol{\varphi}_{\mathrm{I}}^T + \boldsymbol{\varphi}_{\mathrm{I}}\boldsymbol{\varphi}_{\mathrm{R}}^T - \boldsymbol{\varphi}_{\mathrm{J}}\boldsymbol{\varphi}_{\mathrm{K}}^T + \boldsymbol{\varphi}_{\mathrm{K}}\boldsymbol{\varphi}_{\mathrm{J}}^T, \\
\tilde{\boldsymbol{\mathcal{J}}} &= -\boldsymbol{\varphi}_{\mathrm{R}}\boldsymbol{\varphi}_{\mathrm{J}}^T + \boldsymbol{\varphi}_{\mathrm{J}}\boldsymbol{\varphi}_{\mathrm{R}}^T + \boldsymbol{\varphi}_{\mathrm{I}}\boldsymbol{\varphi}_{\mathrm{K}}^T - \boldsymbol{\varphi}_{\mathrm{K}}\boldsymbol{\varphi}_{\mathrm{I}}^T, \\
\tilde{\boldsymbol{\mathcal{K}}} &= -\boldsymbol{\varphi}_{\mathrm{R}}\boldsymbol{\varphi}_{\mathrm{K}}^T + \boldsymbol{\varphi}_{\mathrm{K}}\boldsymbol{\varphi}_{\mathrm{R}}^T - \boldsymbol{\varphi}_{\mathrm{I}}\boldsymbol{\varphi}_{\mathrm{J}}^T + \boldsymbol{\varphi}_{\mathrm{J}}\boldsymbol{\varphi}_{\mathrm{I}}^T,
\end{aligned}$$

Using the extended entities, $\mathcal{I}m\{\boldsymbol{\varphi}(n)\boldsymbol{\varphi}^H(n)\tilde{\boldsymbol{w}}(n)\}$ is replaced by $\tilde{\boldsymbol{C}}_{\mathrm{ext}}^{\mathrm{Im}}\tilde{\boldsymbol{w}}(n)$, where,

$$\tilde{\boldsymbol{C}}_{\mathrm{ext}}^{\mathrm{Im}} = [\mathrm{diag}(0,\ 1,\ 1,\ 1) \otimes \mathbf{I}_M]\, \tilde{\boldsymbol{C}}_{\mathrm{ext}}. \tag{3.40}$$

The extended version of $\boldsymbol{\varphi}(n)v^*(n)$ is given by

$$\tilde{\boldsymbol{\eta}}_{\mathrm{ext}} = \tilde{\boldsymbol{\eta}}_{\mathrm{ext}_1} + \tilde{\boldsymbol{\eta}}_{\mathrm{ext}_2} + \tilde{\boldsymbol{\eta}}_{\mathrm{ext}_3} + \tilde{\boldsymbol{\eta}}_{\mathrm{ext}_4}, \tag{3.41}$$

where

$$\begin{aligned}
\tilde{\boldsymbol{\eta}}_{\mathrm{ext}_1} &= \mathrm{col}(\ v_{\mathrm{R}}(n),\ \ -v_{\mathrm{I}}(n),\ \ -v_{\mathrm{J}}(n),\ \ -v_{\mathrm{K}}(n)\ ) \otimes \boldsymbol{\varphi}_{\mathrm{R}}(n) \\
\tilde{\boldsymbol{\eta}}_{\mathrm{ext}_2} &= \mathrm{col}(\ v_{\mathrm{I}}(n),\ \ v_{\mathrm{R}}(n),\ \ v_{\mathrm{K}}(n),\ \ -v_{\mathrm{J}}(n)\ ) \otimes \boldsymbol{\varphi}_{\mathrm{I}}(n) \\
\tilde{\boldsymbol{\eta}}_{\mathrm{ext}_3} &= \mathrm{col}(\ v_{\mathrm{J}}(n),\ \ -v_{\mathrm{K}}(n),\ \ v_{\mathrm{R}}(n),\ \ v_{\mathrm{I}}(n)\ ) \otimes \boldsymbol{\varphi}_{\mathrm{J}}(n) \\
\tilde{\boldsymbol{\eta}}_{\mathrm{ext}_4} &= \mathrm{col}(\ v_{\mathrm{K}}(n),\ \ v_{\mathrm{J}}(n),\ \ v_{\mathrm{I}}(n),\ \ -v_{\mathrm{R}}(n)\ ) \otimes \boldsymbol{\varphi}_{\mathrm{K}}(n),
\end{aligned} \tag{3.42}$$

(3.43)

and the extended vector of $\mathcal{I}m\{\boldsymbol{\varphi}v^*(n)\}$ is

$$\tilde{\boldsymbol{\eta}}_{\mathrm{ext}}^{\mathrm{Im}} = [\mathrm{diag}(0,\ 1,\ 1,\ 1) \otimes \boldsymbol{I}_M]\tilde{\boldsymbol{\eta}}_{\mathrm{ext}}. \tag{3.44}$$

Using eqs. (3.39)–(3.44), the extended version of (3.38) is given by

$$\tilde{\boldsymbol{w}}_{\mathrm{ext}}(n+1) = \tilde{\boldsymbol{w}}_{\mathrm{ext}}(n) - \mu(g+h)[\tilde{\boldsymbol{C}}_{\mathrm{ext}}\tilde{\boldsymbol{w}}_{\mathrm{ext}}(n) + \tilde{\boldsymbol{\eta}}_{\mathrm{ext}}] + 2\mu h[\tilde{\boldsymbol{C}}_{\mathrm{ext}}^{\mathrm{Im}}\tilde{\boldsymbol{w}}_{\mathrm{ext}}(n) + \tilde{\boldsymbol{\eta}}_{\mathrm{ext}}^{\mathrm{Im}}]. \tag{3.45}$$

Considering the structure of $\tilde{\boldsymbol{C}}_{\mathrm{ext}}$ and $\tilde{\boldsymbol{C}}_{\mathrm{ext}}^{\mathrm{Im}}$ (and also of $\tilde{\boldsymbol{\eta}}_{\mathrm{ext}}$ and $\tilde{\boldsymbol{\eta}}_{\mathrm{ext}}^{\mathrm{Im}}$) this equation can be simplified to

$$\tilde{\boldsymbol{w}}_{\mathrm{ext}}(n+1) = \tilde{\boldsymbol{w}}_{\mathrm{ext}}(n) - \mu\boldsymbol{G}_{\mathrm{ext}}[\tilde{\boldsymbol{C}}_{\mathrm{ext}}\tilde{\boldsymbol{w}}_{\mathrm{ext}}(n) + \tilde{\boldsymbol{\eta}}_{\mathrm{ext}}], \tag{3.46}$$

where

$$\boldsymbol{G}_{\mathrm{ext}} = \boldsymbol{G} \otimes \boldsymbol{I}_M \tag{3.47}$$

and

$$\boldsymbol{G} = \text{diag}(\,(g+h),(g-h),(g-h),(g-h)\,).$$

Define $\bar{\boldsymbol{w}}_{\text{ext}}(n) = E\{\tilde{\boldsymbol{w}}_{\text{ext}}(n)\}$. Taking the expectation of (3.46), one gets

$$\bar{\boldsymbol{w}}_{\text{ext}}(n+1) = \bar{\boldsymbol{w}}_{\text{ext}}(n) - \mu \boldsymbol{G}_{\text{ext}} E\{\tilde{\boldsymbol{C}}_{\text{ext}} \tilde{\boldsymbol{w}}_{\text{ext}}(n)\}, \tag{3.48}$$

where we used the assumptions for $v(n)$ to compute $E\{\tilde{\boldsymbol{\eta}}_{\text{ext}}\} = \boldsymbol{0}$.

To evaluate $E\{\tilde{\boldsymbol{C}}_{\text{ext}} \tilde{\boldsymbol{w}}_{\text{ext}}(n)\}$, we recall the independence approximation usually applied to study LMS-like algorithms [32],

$$E\{\boldsymbol{\varphi}(n)\boldsymbol{\varphi}^H(n)\tilde{\boldsymbol{w}}(n)\} \approx E\{\boldsymbol{\varphi}(n)\boldsymbol{\varphi}^H(n)\}\bar{\boldsymbol{w}}(n).$$

For the extended version of $E\{\boldsymbol{\varphi}(n)\boldsymbol{\varphi}^H(n)\tilde{\boldsymbol{w}}(n)\}$, it leads to

$$E\{\tilde{\boldsymbol{C}}_{\text{ext}} \tilde{\boldsymbol{w}}_{\text{ext}}(n)\} \approx \boldsymbol{C}_{\text{ext}} \bar{\boldsymbol{w}}_{\text{ext}}(n), \tag{3.49}$$

where

$$\boldsymbol{C}_{\text{ext}} = E\{\tilde{\boldsymbol{C}}_{\text{ext}}\} = \begin{bmatrix} \boldsymbol{\mathcal{R}} & \boldsymbol{\mathcal{I}}^T & \boldsymbol{\mathcal{J}}^T & \boldsymbol{\mathcal{K}}^T \\ \boldsymbol{\mathcal{I}} & \boldsymbol{\mathcal{R}} & \boldsymbol{\mathcal{K}}^T & \boldsymbol{\mathcal{J}} \\ \boldsymbol{\mathcal{J}} & \boldsymbol{\mathcal{K}} & \boldsymbol{\mathcal{R}} & \boldsymbol{\mathcal{I}}^T \\ \boldsymbol{\mathcal{K}} & \boldsymbol{\mathcal{J}}^T & \boldsymbol{\mathcal{I}} & \boldsymbol{\mathcal{R}} \end{bmatrix}. \tag{3.50}$$

and $\boldsymbol{\mathcal{R}} = E\{\tilde{\boldsymbol{\mathcal{R}}}\}$, $\boldsymbol{\mathcal{I}} = E\{\tilde{\boldsymbol{\mathcal{I}}}\}$, $\boldsymbol{\mathcal{J}} = E\{\tilde{\boldsymbol{\mathcal{J}}}\}$ and $\boldsymbol{\mathcal{K}} = E\{\tilde{\boldsymbol{\mathcal{K}}}\}$ (see (3.39)). Using (3.49) in (3.48), one gets

$$\bar{\boldsymbol{w}}_{\text{ext}}(n+1) = (\boldsymbol{I}_M - \mu \boldsymbol{G}_{\text{ext}} \boldsymbol{C}_{\text{ext}})\, \bar{\boldsymbol{w}}_{\text{ext}}(n). \tag{3.51}$$

The product $\boldsymbol{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}}$ is the extended quaternion autocorrelation matrix, which is obtained when the quaternion entries of (3.38) are mapped to the real field. $\boldsymbol{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}}$ can be used to study the convergence of the algorithms, which is assessed through its eigenvalue spread. Note that $\boldsymbol{C}_{\text{ext}}$ is a non-negative definite matrix, as we prove in Appendix A (see page 79). We also show in the appendix that the entries of $\boldsymbol{G}_{\text{ext}}$ must be non-negative for the recursion (3.38) to be stable.

In the next section, the convergence of (3.51) is studied in two situations, where we prove that a gradient with $h = 0$ provides the fastest-converging algorithms. We start with the case when at most two quaternion elements are correlated, and then we study quaternion algorithms using a real regressor vector.

3.3.1 Case one: signals with correlation between at most two quaternion elements

Consider the general expression (3.51), and define $\alpha = (g+h)/(g-h)$. Assume that[4] $\boldsymbol{\mathcal{J}} = \boldsymbol{\mathcal{K}} = \mathbf{0}$, such that $\mathbf{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}}$ is written as

$$(g-h)\begin{bmatrix} \mathbf{A}(\alpha) & \mathbf{0} \\ \mathbf{0} & \mathbf{B} \end{bmatrix} = (g-h)\boldsymbol{C}_{\text{ext}}(\alpha), \tag{3.52}$$

where

$$\mathbf{A}(\alpha) = \begin{bmatrix} \alpha\boldsymbol{\mathcal{R}} & \alpha\boldsymbol{\mathcal{I}}^T \\ \boldsymbol{\mathcal{I}} & \boldsymbol{\mathcal{R}} \end{bmatrix}$$

and $\mathbf{B} = \mathbf{A}(1)$. In addition, recall that the spreading factor of a non-negative definite matrix is given by

$$SF = \frac{\lambda_{\text{Max}}}{\lambda_{\text{Min}}}, \tag{3.53}$$

where λ_{Max} and λ_{Min} are the largest and the smallest eigenvalues, respectively. Define the spreading factor of $\mathbf{A}(\alpha)$, $\mathbf{B}$ and $\boldsymbol{C}_{\text{ext}}(\alpha)$ as $SF_{\mathbf{A}}$, $SF_{\mathbf{B}}$ and $SF_{\boldsymbol{C}_{\text{ext}}}$. We want to show that the smallest condition number of (3.52) is obtained when $\alpha = 1$. For this purpose, we first consider $\mathbf{A}(\alpha)$ to show that it has the minimum condition number when $\alpha = 1$. Then, we show that the smallest condition number of $\boldsymbol{C}_{\text{ext}}(\alpha)$ is also obtained when $\alpha = 1$.

First, consider $\mathbf{A}(\alpha)$. $\mathbf{A}(\alpha)$ corresponds to a row scaling of $\mathbf{B}$, since it can be written as

$$\mathbf{A}(\alpha) = \mathbf{D}(\alpha)\mathbf{B},$$

where $\mathbf{D}(\alpha)$ is the scaling matrix, given by $\mathbf{D}(\alpha) = \text{diag}(\alpha,\ 1)\otimes\mathbf{I}_M$. In [64], the problem of determining the row scaling of a matrix which minimizes the Euclidean condition number is addressed, and it is shown that this problem is convex. With this information, we know that there is a value of α which minimizes the condition number, which we now find out.

Studying matrix $\mathbf{A}(\alpha)$, one can show that the condition number of $\mathbf{A}(\alpha)$ is equal to that of $\mathbf{A}(1/\alpha)$. For this purpose, we use an unitary transformation [46] – which does not change the eigenvalues of a matrix – to show that the spreading factors of $\mathbf{A}(\alpha)$ and $\mathbf{A}(1/\alpha)$ have the same value, since

$$\begin{bmatrix} \mathbf{0} & \mathbf{I}_{2M} \\ -\mathbf{I}_{2M} & \mathbf{0} \end{bmatrix} \mathbf{A}(\alpha) \begin{bmatrix} \mathbf{0} & -\mathbf{I}_{2M} \\ \mathbf{I}_{2M} & \mathbf{0} \end{bmatrix} = \alpha\mathbf{A}(1/\alpha),$$

where we also have used the fact that $\boldsymbol{\mathcal{I}} = -\boldsymbol{\mathcal{I}}^T$. Notice that when $\alpha \to \infty$ or when $\alpha \to 0$,

[4]The results would be similar if any two terms in $(\boldsymbol{\mathcal{I}}, \boldsymbol{\mathcal{J}}, \boldsymbol{\mathcal{K}})$ were zero.

the condition numbers of $\mathbf{A}(\alpha)$ and $\mathbf{A}(1/\alpha)$ both go to ∞. Since $SF_{\mathbf{A}}(\alpha) = SF_{\mathbf{A}}(1/\alpha)$, and $SF_{\mathbf{A}}(\alpha)$ is a convex function, the minimum condition number must be at the point which defines the axis of symmetry of the problem. This point corresponds to $\alpha = 1$, where $SF_{\mathbf{A}} = SF_{\mathbf{B}}$. For different values of α, $SF_{\mathbf{A}} > SF_{\mathbf{B}}$.

Using the result for $\mathbf{A}(\alpha)$, we can evaluate the condition number of matrix $\boldsymbol{C}_{\text{ext}}(\alpha)$ in eq. (3.52). $SF_{\boldsymbol{C}_{\text{ext}}}$ is given by[5]

$$SF_{\boldsymbol{C}_{\text{ext}}} = \frac{\max\{\lambda_{\text{Max}}(\mathbf{A}(\alpha)), \lambda_{\text{Max}}(\mathbf{B})\}}{\min\{\lambda_{\text{Min}}(\mathbf{A}(\alpha)), \lambda_{\text{Min}}(\mathbf{B})\}}, \tag{3.54}$$

where $\lambda_{\text{Max}}(\mathbf{A}(\alpha))$ and $\lambda_{\text{Max}}(\mathbf{B})$ are the maximum eigenvalues of $\mathbf{A}(\alpha)$ and $\mathbf{B}$, respectively, and $\lambda_{\text{Min}}(\mathbf{A}(\alpha))$ and $\lambda_{\text{Min}}(\mathbf{B})$ are the minimum eigenvalues. Using (3.54), it is possible to list the cases which can appear in the computation of $SF_{\boldsymbol{C}_{\text{ext}}}$, i.e.,

1. $\lambda_{\text{Max}}(\mathbf{A}(\alpha)) > \lambda_{\text{Max}}(\mathbf{B})$ and $\lambda_{\text{Min}}(\mathbf{A}(\alpha)) \leq \lambda_{\text{Min}}(\mathbf{B})$.
   In this case, $SF_{\boldsymbol{C}_{\text{ext}}} = \lambda_{\text{Max}}(\mathbf{A}(\alpha))/\lambda_{\text{Min}}(\mathbf{A}(\alpha)) > SF_{\mathbf{B}}$.

2. $\lambda_{\text{Max}}(\mathbf{A}(\alpha)) > \lambda_{\text{Max}}(\mathbf{B})$ and $\lambda_{\text{Min}}(\mathbf{A}(\alpha)) \geq \lambda_{\text{Min}}(\mathbf{B})$.
   In this case, $SF_{\boldsymbol{C}_{\text{ext}}} = \lambda_{\text{Max}}(\mathbf{A}(\alpha))/\lambda_{\text{Min}}(\mathbf{B}) > SF_{\mathbf{B}}$.

3. $\lambda_{\text{Max}}(\mathbf{A}(\alpha)) \leq \lambda_{\text{Max}}(\mathbf{B})$ and $\lambda_{\text{Min}}(\mathbf{A}(\alpha)) < \lambda_{\text{Min}}(\mathbf{B})$.
   In this case, $SF_{\boldsymbol{C}_{\text{ext}}} = \lambda_{\text{Max}}(\mathbf{B})/\lambda_{\text{Min}}(\mathbf{A}(\alpha)) > SF_{\mathbf{B}}$.

4. $\lambda_{\text{Max}}(\mathbf{A}(\alpha)) = \lambda_{\text{Max}}(\mathbf{B})$ and $\lambda_{\text{Min}}(\mathbf{A}(\alpha)) = \lambda_{\text{Min}}(\mathbf{B})$.
   In this case, $SF_{\boldsymbol{C}_{\text{ext}}} = SF_{\mathbf{B}}$.

Notice that the condition $\lambda_{\text{Max}}(\mathbf{A}(\alpha)) < \lambda_{\text{Max}}(\mathbf{B})$ and $\lambda_{\text{Min}}(\mathbf{A}(\alpha)) > \lambda_{\text{Min}}(\mathbf{B})$ is not possible, since we have shown that $SF_{\mathbf{A}} \geq SF_B$ always.

For the three first conditions, the eigenvalue spread is always increased, while the last possibility reveals that the minimum value of $SF_{\boldsymbol{C}_{\text{ext}}}$ occurs when $\mathbf{A}(\alpha) = \mathbf{B}$, for $\alpha = 1$. Recalling that $\alpha = (g+h)/(g-h)$, we conclude that h must be zero, showing that the minimum eigenvalue spread is obtained with a gradient similar to that of [42] or [60], for any QLMS-based algorithm.

[5]The constant $(g-h)$ does not affect the eigenvalue spread of eq. (3.52), since it multiplies all the eigenvalues.

3.3.2 Case two: Widely-linear algorithms using real data vector

Consider a WL quaternion algorithm, implemented with the real-regressor vector $\boldsymbol{q}_{\mathrm{RC}}(n)$. In this case, the general expression applied to the mean analysis in (3.51) can be particularized to an equation easier to manipulate, since $\tilde{\boldsymbol{\eta}}_{\mathrm{ext}}$ (see eq. (3.41)) and $\tilde{\boldsymbol{C}}_{\mathrm{ext}}$ (see eq. (3.39)) have simplified expressions. Below, we show how the expressions of $\boldsymbol{\eta}_{\mathrm{ext}}$ and $\tilde{\boldsymbol{C}}_{\mathrm{ext}}$ are modified when $\boldsymbol{q}_{\mathrm{RC}}(n)$ is employed, and how the study of $\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{ext}}$ benefits from this approach.

When the regressor vector is $\boldsymbol{q}_{\mathrm{RC}}(n)$, $\tilde{\boldsymbol{\eta}}_{\mathrm{ext}}$ is simplified to

$$\tilde{\boldsymbol{\eta}}_{\mathrm{ext}} = \mathrm{col}(\ v_{\mathrm{R}},\ \ -v_{\mathrm{I}},\ \ -v_{\mathrm{J}}\ \ -v_{\mathrm{K}}\ ) \otimes \boldsymbol{q}_{\mathrm{RC}}(n), \tag{3.55}$$

since $\tilde{\boldsymbol{\eta}}_{\mathrm{ext}_2}$, $\tilde{\boldsymbol{\eta}}_{\mathrm{ext}_3}$ and $\tilde{\boldsymbol{\eta}}_{\mathrm{ext}_4}$ of eq. (3.42) vanish, and $\boldsymbol{\varphi}_{\mathrm{R}}(n) = \boldsymbol{q}_{\mathrm{RC}}(n)$. It still holds that $E\{\tilde{\boldsymbol{\eta}}_{\mathrm{ext}}\} = \boldsymbol{0}$.

$\tilde{\boldsymbol{C}}_{\mathrm{ext}}$ is real-valued for this approach, given by $\tilde{\boldsymbol{C}}_{\mathrm{ext}} = \boldsymbol{I}_4 \otimes \tilde{\boldsymbol{C}}_{\mathrm{RC}}$, so that the extended autocorrelation matrix reduces to

$$\boldsymbol{C}_{\mathrm{RC_{ext}}} = \boldsymbol{I}_4 \otimes \boldsymbol{C}_{\mathrm{RC}}, \tag{3.56}$$

where $\boldsymbol{C}_{\mathrm{RC}}$ is computed as presented in (3.25).

Replacing $\boldsymbol{C}_{\mathrm{ext}}$ by (3.56) in equation (3.51), we obtain

$$\bar{\boldsymbol{w}}_{\mathrm{ext}}(n+1) = (\boldsymbol{I}_{4N} - \mu_{\mathrm{WL}}\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}})\,\bar{\boldsymbol{w}}_{\mathrm{ext}}(n). \tag{3.57}$$

so that the convergence in the mean can be studied through matrix

$$\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}} = (\boldsymbol{G} \otimes \boldsymbol{I}_{4N})(\boldsymbol{I}_4 \otimes \boldsymbol{C}_{\mathrm{RC}}) = \boldsymbol{G} \otimes \boldsymbol{C}_{\mathrm{RC}},$$

which has a known structure that can be exploited in the analysis.

The product $\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}}$ is a block-diagonal matrix, in which one block-matrix is $(g+h)\boldsymbol{C}_{\mathrm{RC}}$ and the other three blocks are $(g-h)\boldsymbol{C}_{\mathrm{RC}}$. N eigenvalues of $\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}}$ correspond to the eigenvalues of $\boldsymbol{C}_{\mathrm{RC}}$, multiplied by $(g+h)$. The other $3N$ eigenvalues correspond to the eigenvalues of $\boldsymbol{C}_{\mathrm{RC}}$, multiplied by $(g-h)$ and with multiplicity 3 each one. The minimum and the maximum eigenvalues of $\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}}$ are

$$\lambda_{\mathrm{Max}}(\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}}) = \max\{(g+h),(g-h)\}\lambda_{\mathrm{Max}}(\boldsymbol{C}_{\mathrm{RC}}) \tag{3.58}$$

and

$$\lambda_{\mathrm{Min}}(\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}}) = \min\{(g-h),(g+h)\}\lambda_{\mathrm{Min}}(\boldsymbol{C}_{\mathrm{RC}}), \tag{3.59}$$

where $\lambda_{\mathrm{Max}}(\boldsymbol{C}_{\mathrm{RC}})$ and $\lambda_{\mathrm{Min}}(\boldsymbol{C}_{\mathrm{RC}})$ are the largest and the smallest eigenvalues of $\boldsymbol{C}_{\mathrm{RC}}$, respectively. The spreading factor is then computed with

$$SF_{\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}}} = \frac{\max\{(g+h),(g-h)\}\lambda_{\mathrm{Max}}(\boldsymbol{C}_{\mathrm{RC}})}{\min\{(g-h),(g+h)\}\lambda_{\mathrm{Min}}(\boldsymbol{C}_{\mathrm{RC}})}, \tag{3.60}$$

that can be simplified to

$$SF_{\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}}} = \max\left\{\frac{(g+h)}{(g-h)}, \frac{(g-h)}{(g+h)}\right\}\frac{\lambda_{\mathrm{Max}}(\boldsymbol{C}_{\mathrm{RC}})}{\lambda_{\mathrm{Min}}(\boldsymbol{C}_{\mathrm{RC}})}. \tag{3.61}$$

The best choice for g and h is again obtained when $(g+h)/(g-h) = 1$, so that $h = 0$ and $a = b + c + d$.

Based on this result, in the next section we propose the fastest-converging WL-QLMS algorithm with real-input, which is also a low-cost version of WL-iQLMS: the reduced-complexity WL-iQLMS algorithm.

3.3.2.1 The RC-WL-iQLMS algorithm

The reduced-complexity WL-iQLMS algorithm, presented in Table 9, is obtained with[6] $g = 3/4$ and $h = 0$ in eq. (3.32), therefore guaranteeing the minimum eigenvalue spread in $\mathbf{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}}$ and the fastest-converging performance among the WL-QLMS algorithms with real input. It has exactly the same performance of WL-iQLMS, which can be verified by obtaining a transformation to map the entries of $\boldsymbol{q}_{\mathrm{RC}}(n)$ in $\boldsymbol{q}_{\mathrm{WL}}(n)$, and then using it to compare the equations of the techniques. For this purpose, recall the WL-iQLMS (see Table 8, page 54). Define

$$\boldsymbol{F} = \begin{bmatrix} 1 & i & j & k \\ 1 & i & -j & -k \\ 1 & -i & j & -k \\ 1 & -i & -j & k \end{bmatrix} \otimes \boldsymbol{I}_N,$$

and note that $\boldsymbol{F}\boldsymbol{F}^H/4 = \boldsymbol{F}^H\boldsymbol{F}/4 = \boldsymbol{I}_{4N}$. One can show that

$$\boldsymbol{q}_{\mathrm{WL}}(n) = \boldsymbol{F}\boldsymbol{q}_{\mathrm{RC}}(n). \tag{3.62}$$

[6]Other values to g could have been used, since the constant that appears in the second term of the right-hand side of the update equation can be absorbed by the step size. We choose this value to make easier the comparison with WL-iQLMS of Table 8

The system of equations which defines the optimum solution $\mathbf{w}_{\mathrm{iQ,opt}}$ of the WL-iQLMS algorithm is given by

$$E\{\boldsymbol{q}_{\mathrm{WL}}(n)\boldsymbol{q}_{\mathrm{WL}}^{H}(n)\}\boldsymbol{w}_{\mathrm{iQ,opt}} = E\{\boldsymbol{q}_{\mathrm{WL}}(n)d^{*}(n)\}.$$

Left-multiplying both sides by $\boldsymbol{F}^{H}/4$ and using (3.62), one gets

$$\underbrace{\frac{\boldsymbol{F}^{H}}{4}\boldsymbol{F}}_{\boldsymbol{I}_{4N}} E\{\boldsymbol{q}_{\mathrm{RC}}(n)\boldsymbol{q}_{\mathrm{RC}}^{T}(n)\}\boldsymbol{F}^{H}\boldsymbol{w}_{\mathrm{iQ,opt}} = \underbrace{\frac{\boldsymbol{F}^{H}}{4}\boldsymbol{F}}_{\boldsymbol{I}_{4N}} E\{\boldsymbol{q}_{\mathrm{RC}}(n)d^{*}(n)\}.$$

Recognizing $E\{\boldsymbol{q}_{\mathrm{RC}}(n)\boldsymbol{q}_{\mathrm{RC}}^{T}(n)\}$ and $E\{\boldsymbol{q}_{\mathrm{RC}}(n)d^{*}(n)\}$ as the autocorrelation matrix and cross-correlation vector of the algorithm proposed in Table 9, then

$$\boldsymbol{w}_{\mathrm{RC_{iQ},opt}} = \boldsymbol{F}^{H}\boldsymbol{w}_{\mathrm{iQ,opt}} \;\text{ or }\; \boldsymbol{w}_{\mathrm{iQ,opt}} = \boldsymbol{F}\boldsymbol{w}_{\mathrm{RC_{iQ},opt}}/4, \tag{3.63}$$

where we used subscript $\mathrm{RC_{iQ}}$ to identify quantities related to RC-WL-iQLMS.

Based on (3.63), define

$$\boldsymbol{w}_{\mathrm{iQ}}(n) = \boldsymbol{F}\boldsymbol{w}_{\mathrm{RC_{iQ}}}(n)/4$$

to show that

$$\hat{d}_{\mathrm{iQ}}(n) = \boldsymbol{w}_{\mathrm{iQ}}^{H}(n)\frac{\boldsymbol{F}^{H}\boldsymbol{F}}{4}\boldsymbol{q}_{\mathrm{WL}}(n) = \boldsymbol{w}_{\mathrm{RC_{iQ}}}^{H}(n)\boldsymbol{q}_{\mathrm{RC}}(n) = \hat{d}_{\mathrm{RC_{iQ}}}(n),$$

and notice that $e_{\mathrm{iQ}}(n) = e_{\mathrm{RC_{iQ}}}(n)$. Finally, left multiplying the update equation of WL-iQLMS by $\boldsymbol{F}^{H}$, we obtain

$$\underbrace{\boldsymbol{F}^{H}\boldsymbol{w}_{\mathrm{iQ}}(n+1)}_{\boldsymbol{w}_{\mathrm{RC_{iQ}}}(n+1)} = \underbrace{\boldsymbol{F}^{H}\boldsymbol{w}_{\mathrm{iQ}}(n)}_{\boldsymbol{w}_{\mathrm{RC_{iQ}}}(n)} + \frac{3}{4}\mu_{\mathrm{iQ}}\boldsymbol{F}^{H}\underbrace{\boldsymbol{q}_{\mathrm{WL}}(n)}_{\boldsymbol{F}\boldsymbol{q}_{\mathrm{RC}}(n)}\underbrace{e_{\mathrm{iQ}}^{*}(n)}_{e_{\mathrm{RC_{iQ}}}^{*}(n)}.$$

Defining $\mu_{\mathrm{RC_{iQ}}} = 4\mu_{\mathrm{iQ}}$, we show that the new technique and WL-iQLMS have exactly the same performance. The new technique is low-cost since it applies the real input, which replaces many quaternion-quaternion operations by real-quaternion calculations in the update equation, and also in the computation of the estimate of $d(n)$. Since each quaternion-quaternion multiplication requires 16 real multiplications and 12 real additions, while a real-quaternion product uses only 4 real multiplications, the RC approach can lead to a considerable reduction in the number of computations (see Table 10).

Note that another low-cost technique based on real-input data was proposed in our initial research on WL-QLMS-based the algorithms. In [18], we developed the RC-WL-QLMS algorithm, which is obtained choosing $g = 1/2$ and $h = -1/4$. The resulting update equation is similar to that of WL-QLMS. Since $h \neq 0$, RC-WL-QLMS is outperformed

by RC-WL-iQLMS in the convergence rate. RC-WL-QLMS is also more costly, since its update law has an additional term. For these reasons, in this text we concentrate only on RC-WL-iQLMS, which is a better alternative.

Table 9: RC-WL-iQLMS algorithm

$$\begin{array}{|l|}\hline \hat{d}_{\mathrm{RC_{iQ}}}(n) = \boldsymbol{w}^H_{\mathrm{RC_{iQ}}}(n)\boldsymbol{q}_{\mathrm{RC}}(n) \\ e_{\mathrm{RC_{iQ}}}(n) = d(n) - \hat{d}_{\mathrm{RC_{iQ}}}(n) \\ \boldsymbol{w}_{\mathrm{RC_{iQ}}}(n+1) = \boldsymbol{w}_{\mathrm{RC_{iQ}}}(n) + \dfrac{3}{4}\mu_{\mathrm{RC_{iQ}}}\boldsymbol{q}_{\mathrm{RC}}(n)e^*_{\mathrm{RC_{iQ}}}(n) \\ \hline \end{array}$$

3.3.2.2 Comparing the RC-WL-iQLMS with the 4-Ch-LMS algorithm

To show that the RC-WL-iQLMS algorithm corresponds to the 4-Ch-LMS algorithm written in the quaternion domain, we must rewrite the equations of Table 9 to separate the imaginary numbers $\boldsymbol{i}$, $\boldsymbol{j}$ and $\boldsymbol{k}$ from the real terms.

Define the extended vectors

$$\begin{aligned}\boldsymbol{d}_{\mathrm{ext}}(n) &= \mathrm{col}\left(d_{\mathrm{R}}(n),\ d_{\mathrm{I}}(n),\ d_{\mathrm{J}}(n),\ d_{\mathrm{K}}(n)\right), \\ \boldsymbol{v}'_{\mathrm{ext}}(n) &= \mathrm{col}\left(v_{\mathrm{R}}(n),\ v_{\mathrm{I}}(n),\ v_{\mathrm{J}}(n),\ v_{\mathrm{K}}(n)\right)\end{aligned}$$

and note that $d(n) = \boldsymbol{t}^T\boldsymbol{d}_{\mathrm{ext}}(n)$ and $v(n) = \boldsymbol{t}^T\boldsymbol{v}'_{\mathrm{ext}}(n)$, where

$$\boldsymbol{t} = \mathrm{col}\left(1,\ \ \boldsymbol{i},\ \ \boldsymbol{j},\ \ \boldsymbol{k}\right). \tag{3.64}$$

Assume that $d(n)$ is modeled as

$$d(n) = \boldsymbol{w}^H_{\mathrm{RC_{iQ},opt}}\boldsymbol{q}_{\mathrm{RC}}(n) + v(n),$$

which can be expressed as

$$\boldsymbol{t}^T\boldsymbol{d}_{\mathrm{ext}}(n) = \boldsymbol{t}^T\left(\boldsymbol{W}^T_{\mathrm{RC_{iQ},opt}}\boldsymbol{q}_{\mathrm{RC}}(n) + \boldsymbol{v}'_{\mathrm{ext}}(n)\right).$$

$\boldsymbol{W}_{\mathrm{RC_{iQ},opt}}$ is the $4N \times 4$ real matrix, given by

$$\boldsymbol{W}_{\mathrm{RC_{iQ},opt}} = \left[\boldsymbol{w}_{\mathrm{RC_{iQ},opt_R}}\ -\boldsymbol{w}_{\mathrm{RC_{iQ},opt_I}}\ -\boldsymbol{w}_{\mathrm{RC_{iQ},opt_J}}\ -\boldsymbol{w}_{\mathrm{RC_{iQ},opt_K}}\right].$$

Define the extended vectors

$$\boldsymbol{e}_{\mathrm{RC}i\mathrm{Q}_{\mathrm{ext}}}(n) = \mathrm{col}\left(\ e_{\mathrm{RC}i\mathrm{Q}_{\mathrm{R}}}(n),\ \ e_{\mathrm{RC}i\mathrm{Q}_{\mathrm{I}}}(n),\ \ e_{\mathrm{RC}i\mathrm{Q}_{\mathrm{J}}}(n),\ \ e_{\mathrm{RC}i\mathrm{Q}_{\mathrm{K}}}(n)\ \right), \tag{3.65}$$

$$\hat{\boldsymbol{d}}_{\mathrm{RC}i\mathrm{Q}_{\mathrm{ext}}}(n) = \mathrm{col}\left(\ \hat{d}_{\mathrm{RC}i\mathrm{Q}_{\mathrm{R}}}(n),\ \ \hat{d}_{\mathrm{RC}i\mathrm{Q}_{\mathrm{I}}}(n),\ \ \hat{d}_{\mathrm{RC}i\mathrm{Q}_{\mathrm{J}}}(n),\ \ \hat{d}_{\mathrm{RC}i\mathrm{Q}_{\mathrm{K}}}(n)\ \right). \tag{3.66}$$

Using (3.64), (3.65) and (3.66) in the equations of Table 9, one gets

$$\boldsymbol{W}(n+1)\boldsymbol{t}^{*}=\left(\boldsymbol{W}(n)+\frac{3}{4}\mu_{\mathrm{RC}iQ}\boldsymbol{q}_{\mathrm{RC}}(n)\boldsymbol{e}^{T}_{\mathrm{RC}iQ_{\mathrm{ext}}}(n)\right)\boldsymbol{t}^{*}, \tag{3.67}$$

$$\boldsymbol{t}^{T}\boldsymbol{e}_{\mathrm{RC}iQ_{\mathrm{ext}}}(n)=\boldsymbol{t}^{T}\left(\boldsymbol{d}_{\mathrm{ext}}(n)-\hat{\boldsymbol{d}}_{\mathrm{RC}iQ_{\mathrm{ext}}}(n)\right) \tag{3.68}$$

and

$$\boldsymbol{t}^{T}\hat{\boldsymbol{d}}_{\mathrm{RC}iQ_{\mathrm{ext}}}(n)=\boldsymbol{t}^{T}\left(\boldsymbol{W}^{T}(n)\boldsymbol{q}_{\mathrm{RC}}(n)\right), \tag{3.69}$$

where

$$\boldsymbol{W}(n)=\left[\begin{array}{cccc}\boldsymbol{w}_{\mathrm{RC}iQ_{\mathrm{R}}}(n) & -\boldsymbol{w}_{\mathrm{RC}iQ_{\mathrm{I}}}(n) & -\boldsymbol{w}_{\mathrm{RC}iQ_{\mathrm{J}}}(n) & -\boldsymbol{w}_{\mathrm{RC}iQ_{\mathrm{K}}}(n)\end{array}\right].$$

Removing the multiplication by $\boldsymbol{t}$ in (3.67), (3.68) and (3.69), we obtain the 4-Ch-LMS algorithm. Thus the proposed algorithm is a rewriting (in the quaternion domain) of the 4-Ch-QLMS algorithm, just as the RC-WL-LMS of [38] is a rewriting of the 2-Ch-LMS algorithm [24].

In Table 10, we present the computational complexity of some QLMS-based techniques proposed in the literature and the 4-Ch-LMS algorithm. The WL-QLMS and WL-iQLMS algorithms are about 4 times more costly than their SL counterparts (QLMS and iQLMS, respectively). The RC-WL-LMS technique is less costly than WL-QLMS and WL-iQLMS, and has almost the complexity of iQLMS. The 4-Ch-LMS, iQLMS and the RC-WL-iQLMS algorithms have the same complexity, and they are the less-costly techniques in the table.

Table 10: Computational complexity in terms of real operations per iteration (N is the length of the SL data vector)

Algorithm	$+$	$\times$
QLMS	$48N$	$48N+9$
WL-QLMS	$192N$	$192N+9$
RC-WL-QLMS	$32N+4$	$32N+8$
iQLMS	$32N$	$32N+4$
WL-iQLMS	$128N$	$128N+4$
RC-WL-iQLMS	$32N$	$32N+4$
4-Ch-LMS	$32N$	$32N+4$

3.4 Analysis of QLMS-based algorithms

In this section, we study the convergence of WL quaternion algorithms based on the QLMS technique. Using our general description of the quaternion gradient, we obtain

mean and mean-square analyses results for any QLMS-based technique, which we later particularize for the case of real-input algorithms.

3.4.1 Designing the step size to guarantee the convergence in the mean of real-regressor vector techniques

Similar to the analysis applied to study the LMS algorithm [32], we can use $\mathbf{G}_{\text{ext}}\boldsymbol{C}_{\text{RC}_{\text{ext}}}$ (see eq. (3.57)) to define bounds for the step size which guarantee the convergence in the mean. From (3.57), we can study the eigenvalues of $\boldsymbol{I}_{4N} - \mu_{\text{WL}}\boldsymbol{G}_{\text{ext}}\boldsymbol{C}_{\text{RC}_{\text{ext}}}$. To guarantee the convergence in the mean, the absolute value of each eigenvalue must be less than 1. Applying this condition, we obtain

$$0 < \mu < \frac{2}{\lambda_{\text{Max}}(\mathbf{G}_{\text{ext}}\boldsymbol{C}_{\text{RC}_{\text{ext}}})} = \frac{2}{\psi(g,h)\lambda_{\text{Max}}(\boldsymbol{C}_{\text{RC}})},$$

where $\psi(g,h) = \max\{(g+h),(g-h)\}$. For RC-WL-iQLMS, the bounds are

$$0 < \mu_{\text{RC}_{\text{iQ}}} < 8/[3\lambda_{\text{Max}}(\boldsymbol{C}_{\text{RC}})],$$

and the spreading factor is given by

$$SF_{\text{RC}_{\text{iQ}}} = \lambda_{\text{Max}}(\boldsymbol{C}_{\text{RC}})/\lambda_{\text{Min}}(\boldsymbol{C}_{\text{RC}}). \tag{3.70}$$

3.4.2 Mean-square analysis of QLMS-based algorithms

In order to perform a general second-order analysis, suitable for any QLMS-based algorithm, we use a general equation, where $\boldsymbol{\varphi}(n)$ is applied to represent any quaternion regressor vector. We particularize the results for real-input WL-QLMS-based techniques in the end of the section.

To start the analysis, recall eq. (3.46) (see page 56). Similarly to the traditional analysis of the LMS algorithm [45], multiply this equation by its transpose and take the

expectation, to obtain

$$\begin{aligned}
&\underbrace{E\{\tilde{\boldsymbol{w}}_{\text{ext}}(n+1)\tilde{\boldsymbol{w}}_{\text{ext}}^T(n+1)\}}_{\boldsymbol{S}(n+1)} = \underbrace{E\{\tilde{\boldsymbol{w}}_{\text{ext}}(n)\tilde{\boldsymbol{w}}_{\text{ext}}^T(n)\}}_{\boldsymbol{S}(n)} \\
&-\mu\underbrace{E\{\tilde{\boldsymbol{w}}_{\text{ext}}(n)\tilde{\boldsymbol{w}}_{\text{ext}}^T(n)\tilde{\boldsymbol{C}}_{\text{ext}}\boldsymbol{G}_{\text{ext}}\}}_{\boldsymbol{\mathcal{A}}} - \mu\underbrace{E\{\boldsymbol{G}_{\text{ext}}\tilde{\boldsymbol{C}}_{\text{ext}}\tilde{\boldsymbol{w}}_{\text{ext}}(n)\tilde{\boldsymbol{w}}_{\text{ext}}^T(n)\}}_{\boldsymbol{\mathcal{B}}} \\
&+\mu^2\underbrace{E\{\boldsymbol{G}_{\text{ext}}\tilde{\boldsymbol{C}}_{\text{ext}}\tilde{\boldsymbol{w}}_{\text{ext}}\tilde{\boldsymbol{w}}_{\text{ext}}^T(n)\tilde{\boldsymbol{C}}_{\text{ext}}\boldsymbol{G}_{\text{ext}}\}}_{\boldsymbol{\mathcal{C}}} + \mu^2\underbrace{E\{\boldsymbol{G}_{\text{ext}}\boldsymbol{\eta}_{\text{ext}}\boldsymbol{\eta}_{\text{ext}}^T\boldsymbol{G}_{\text{ext}}\}}_{\boldsymbol{\mathcal{D}}} \\
&-\mu\underbrace{E\{\boldsymbol{G}_{\text{ext}}\boldsymbol{\eta}_{\text{ext}}(n)\tilde{\boldsymbol{w}}_{\text{ext}}^T(n)\}}_{\boldsymbol{\mathcal{E}}} - \mu\underbrace{E\{\tilde{\boldsymbol{w}}_{\text{ext}}(n)\boldsymbol{\eta}_{\text{ext}}^T(n)\boldsymbol{G}_{\text{ext}}\}}_{\boldsymbol{\mathcal{F}}} \\
&+\mu^2\underbrace{E\{\boldsymbol{G}_{\text{ext}}\boldsymbol{\eta}_{\text{ext}}(n)\tilde{\boldsymbol{w}}_{\text{ext}}(n)\tilde{\boldsymbol{C}}_{\text{ext}}\boldsymbol{G}_{\text{ext}}\}}_{\boldsymbol{\mathcal{G}}} + \mu^2\underbrace{E\{\boldsymbol{G}_{\text{ext}}\tilde{\boldsymbol{C}}_{\text{ext}}\boldsymbol{w}_{\text{ext}}(n)\boldsymbol{\eta}_{\text{ext}}^T\boldsymbol{G}_{\text{ext}}(n)\}}_{\boldsymbol{\mathcal{H}}},
\end{aligned} \tag{3.71}$$

where we define $\boldsymbol{S}(n) = E\{\tilde{\boldsymbol{w}}_{\text{ext}}(n)\tilde{\boldsymbol{w}}_{\text{ext}}^T(n)\}$ to simplify the notation. From eq. (3.71) the second-order model for small step-sizes is derived. However, some approximations are required to proceed with the analysis, which are presented in the Assumption I next.

Assumption I: Assume that the sequence $\{\boldsymbol{\varphi}(n)\}$ is Gaussian, stationary and zero-mean, and that $\{\boldsymbol{\varphi}(n)\}$ and $\{v(n)\}$ are independent from each other. Additionally, assume that $E\{\boldsymbol{v}_{\text{ext}}(n)\boldsymbol{v}_{\text{ext}}^T(n)\} = \sigma_v^2\boldsymbol{I}_4$ and that μ is small enough such that $\boldsymbol{\varphi}(n)$ and $\boldsymbol{w}(n)$ are approximately independent.

Based on Assumption I, the terms of eq. (3.71) are studied as follows[7]:

1. Term $\boldsymbol{\mathcal{A}}$ – This term can be approximated as

$$\begin{aligned}
\boldsymbol{\mathcal{A}} &= E\{\tilde{\boldsymbol{w}}_{\text{ext}}\tilde{\boldsymbol{w}}_{\text{ext}}^T\tilde{\boldsymbol{C}}_{\text{ext}}|\boldsymbol{\varphi}\}\boldsymbol{G}_{\text{ext}} \\
&\approx E\{E\{\tilde{\boldsymbol{w}}_{\text{ext}}\tilde{\boldsymbol{w}}_{\text{ext}}^T\}\tilde{\boldsymbol{C}}_{\text{ext}}\}\boldsymbol{G}_{\text{ext}} \\
&= \boldsymbol{S}(n)\boldsymbol{C}_{\text{ext}}\boldsymbol{G}_{\text{ext}},
\end{aligned} \tag{3.72}$$

   where we used $E\{\tilde{\boldsymbol{C}}_{\text{ext}}\} = \boldsymbol{C}_{\text{ext}}$.

2. Term $\boldsymbol{\mathcal{B}}$ – Similarly to $\boldsymbol{\mathcal{A}}$, term $\boldsymbol{\mathcal{B}}$ reduces to

$$\boldsymbol{\mathcal{B}} = \boldsymbol{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}}\boldsymbol{S}(n).$$

3. Term $\boldsymbol{\mathcal{C}}$ – This term can be rewritten as

$$\begin{aligned}
\boldsymbol{\mathcal{C}} &= \boldsymbol{G}_{\text{ext}}E\{\tilde{\boldsymbol{C}}_{\text{ext}}\tilde{\boldsymbol{w}}_{\text{ext}}\tilde{\boldsymbol{w}}_{\text{ext}}^T\tilde{\boldsymbol{C}}_{\text{ext}}\}\boldsymbol{G}_{\text{ext}} \\
&\cong \boldsymbol{G}_{\text{ext}}E\{\tilde{\boldsymbol{C}}_{\text{ext}}\boldsymbol{S}(n)\tilde{\boldsymbol{C}}_{\text{ext}}\}\boldsymbol{G}_{\text{ext}}.
\end{aligned} \tag{3.73}$$

[7]Note that we drop the time coefficient to simplify the notation.

4. Term $\boldsymbol{\mathcal{D}}$ – Recalling equations (3.41) and (3.42) (see p. 56), $\boldsymbol{\mathcal{D}}$ is given by

$$\begin{aligned}\boldsymbol{\mathcal{D}} &= \boldsymbol{G}_{\text{ext}} E\{\boldsymbol{\eta}_{\text{ext}}(n)\boldsymbol{\eta}_{\text{ext}}^T(n)\}\boldsymbol{G}_{\text{ext}} \\ &= \boldsymbol{G}_{\text{ext}} E\{\sum_{m=1}^{4}\sum_{\ell=1}^{4}\boldsymbol{\eta}_{\text{ext}_\text{m}}(n)\boldsymbol{\eta}_{\text{ext}_\ell}^T(n)\}\boldsymbol{G}_{\text{ext}}.\end{aligned} \tag{3.74}$$

   For a general case, $\boldsymbol{\mathcal{D}}$ is not structured, such that a simple analytical expression is difficult to obtain. However, with knowledge on the statistics of the noise and of the quaternion vector $\boldsymbol{\varphi}(n)$, $\boldsymbol{\mathcal{D}}$ can be computed with a computer program. When the input is $\boldsymbol{q}_{\text{RC}}(n)$, a simple expression can be obtained for $\boldsymbol{\mathcal{D}}$. We present this expression later, when we particularize the analysis for real-input WL-QLMS algorithms.

5. Terms $\boldsymbol{\mathcal{E}}$, $\boldsymbol{\mathcal{F}}$, $\boldsymbol{\mathcal{G}}$ and $\boldsymbol{\mathcal{H}}$ – From Assumption I, the elements of $\boldsymbol{v}_{\text{ext}}$ are zero-mean random variables and are independent of $\boldsymbol{\varphi}(n)$ and $\tilde{\boldsymbol{w}}_{\text{ext}}$. Using this argument, $\boldsymbol{\mathcal{E}}$, $\boldsymbol{\mathcal{F}}$, $\boldsymbol{\mathcal{G}}$ and $\boldsymbol{\mathcal{H}}$ result in $4M \times 4M$ null matrices.

With these approximations, the recursion for $\boldsymbol{S}(n)$ results

$$\boldsymbol{S}(n+1) \approx \boldsymbol{S}(n) + \mu^2\boldsymbol{\mathcal{C}} - \mu\left[\boldsymbol{S}(n)\boldsymbol{C}_{\text{ext}}\boldsymbol{G}_{\text{ext}} + \boldsymbol{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}}\boldsymbol{S}(n)\right] + \mu^2\boldsymbol{\mathcal{D}}. \tag{3.75}$$

Note that the three first terms on the right-hand side are linear in $\boldsymbol{S}(n)$. Assuming that the step size is small enough so that $\mu^2\boldsymbol{\mathcal{C}}$ can be neglected, we obtain the simplified model

$$\boldsymbol{S}(n+1) \approx \boldsymbol{S}(n) - \mu\boldsymbol{S}(n)\boldsymbol{C}_{\text{ext}}\boldsymbol{G}_{\text{ext}} - \mu\boldsymbol{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}}\boldsymbol{S}(n) + \mu^2\boldsymbol{\mathcal{D}}, \tag{3.76}$$

where the initialization corresponds to $\boldsymbol{S}(0) = \tilde{\boldsymbol{w}}_{\text{ext}}(0)\tilde{\boldsymbol{w}}_{\text{ext}}^T(0)$.

Using (3.76), two theoretical quantities can be calculated at each time instant: the excess mean-square error (EMSE) [32],

$$\zeta(n) = E\{|e(n)|^2\} = \text{Tr}(\boldsymbol{S}(n)\boldsymbol{C}_{\text{ext}}),$$

and the mean-square deviation (MSD) [32]

$$\chi(n) = E\{||\tilde{\boldsymbol{w}}_{\text{ext}}(n)||_2^2\} = \text{Tr}(\boldsymbol{S}(n)).$$

From the small step-size model, we can also deduce the steady-state EMSE for general case. The steady-state MSD cannot be computed in our general approach, since $\text{Tr}(\boldsymbol{S}(\infty))$ cannot be isolated in the general expression. However, it can be computed for real-input WL-QLMS-based techniques, as we will show in Section 3.4.3.

To compute the steady-state EMSE, assume that for $n \to \infty$, $\boldsymbol{S}(n+1) \approx \boldsymbol{S}(n) = \boldsymbol{S}(\infty)$, such that eq. (3.76) reduces to

$$\boldsymbol{S}(\infty)\boldsymbol{C}_{\text{ext}}\boldsymbol{G}_{\text{ext}} + \boldsymbol{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}}\boldsymbol{S}(\infty) \approx \mu\boldsymbol{\mathcal{D}}. \tag{3.77}$$

Multiply eq. (3.77) by $\boldsymbol{G}_{\text{ext}}^{-1}$ from the right and take the trace. Using the trace property $\text{Tr}(\boldsymbol{AB}) = \text{Tr}(\boldsymbol{BA})$, we obtain

$$\zeta(\infty) \approx \mu\text{Tr}(\boldsymbol{\mathcal{D}}\boldsymbol{G}_{\text{ext}}^{-1})/2, \tag{3.78}$$

which only depends on $\boldsymbol{\mathcal{D}}$ (that captures the relation among the noise and the input) and $\boldsymbol{G}_{\text{ext}}$, that depends on the algorithm that is applied.

Using our general approach to study QLMS-based algorithms, in this section we obtained a small step-size model to iteratively compute the MSD and the EMSE of any QLMS-algorithm. We also derived an analytical expression for the steady-state EMSE. In the next section, we particularize these results for real-regressor WL-QLMS-based techniques.

3.4.3 Mean-square analysis of WL-QLMS algorithms with real input

When the input of a WL-QLMS-based algorithm is a real vector, the analysis presented in Section 3.4.2 can be simplified, leading to expressions easier to manipulate and interpret. In this section, we exploit it to obtain an accurate model and relations to help the design of the algorithms.

For this purpose, initially consider the expectation of the terms used to compute $\boldsymbol{S}(n)$ in (3.71). Note that when the input vector is given by $\boldsymbol{q}_{\text{RC}}(n)$, the extended autocorrelation matrix $\boldsymbol{C}_{\text{ext}}$ is replaced by $\boldsymbol{C}_{\text{RC}_{\text{ext}}}$ in $\boldsymbol{\mathcal{A}}$, $\boldsymbol{\mathcal{B}}$ and $\boldsymbol{\mathcal{C}}$. Additionally, recall that when the regressor vector is real, $\boldsymbol{\eta}_{\text{ext}}$ simplifies to (3.55) (see p. 60,) and term $\boldsymbol{\mathcal{D}}$ is given by

$$\begin{aligned}\boldsymbol{\mathcal{D}} &= \boldsymbol{G}_{\text{ext}}E\{\boldsymbol{\eta}_{\text{ext}}(n)\boldsymbol{\eta}_{\text{ext}}^T(n)\}\boldsymbol{G}_{\text{ext}} \\ &= \boldsymbol{G}_{\text{ext}}E\{(\boldsymbol{v}_{\text{ext}} \otimes \boldsymbol{q}_{\text{RC}})\left(\boldsymbol{v}_{\text{ext}}^T \otimes \boldsymbol{q}_{\text{RC}}^T\right)\}\boldsymbol{G}_{\text{ext}}.\end{aligned} \tag{3.79}$$

Using property 3 of Kronecker products and Assumption I, we simplify eq. (3.79) to

$$\begin{aligned}\boldsymbol{\mathcal{D}} &= \boldsymbol{G}_{\text{ext}}\left(E\{\boldsymbol{v}_{\text{ext}}\boldsymbol{v}_{\text{ext}}^T\} \otimes E\{\boldsymbol{q}_{\text{RC}}\boldsymbol{q}_{\text{RC}}^T\}\right)\boldsymbol{G}_{\text{ext}} \\ &= \sigma_v^2\boldsymbol{G}_{\text{ext}}\left(\boldsymbol{I}_4 \otimes E\{\boldsymbol{q}_{\text{RC}}\boldsymbol{q}_{\text{RC}}^T\}\right)\boldsymbol{G}_{\text{ext}} \\ &= \sigma_v^2\boldsymbol{G}_{\text{ext}}\boldsymbol{C}_{\text{RC}_{\text{ext}}}\boldsymbol{G}_{\text{ext}},\end{aligned}$$

where we have used (3.56) to identify $\boldsymbol{C}_{\mathrm{RC_{ext}}}$. The other terms $\boldsymbol{\mathcal{E}}$, $\boldsymbol{\mathcal{F}}$, $\boldsymbol{\mathcal{G}}$ and $\boldsymbol{\mathcal{H}}$ are equal to zero matrices.

Note that all the results obtained in the analysis for any QLMS-based technique still hold when the input is $\boldsymbol{q}_{\mathrm{RC}}(n)$. Using (3.4.3), the steady-state EMSE expression, simplifies to

$$\zeta(\infty) \approx \mu\sigma_v^2 \mathrm{Tr}(\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}}\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{G}_{\mathrm{ext}}^{-1})/2 = \mu\sigma_v^2 \mathrm{Tr}(\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}})/2. \tag{3.80}$$

Using (3.47) and (3.56) to write $\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}} = \boldsymbol{G} \otimes \boldsymbol{C}_{\mathrm{RC}}$, and applying the trace property of the Kronecker product, one gets

$$\zeta(\infty) \approx \frac{\mu\sigma_v^2 \mathrm{Tr}(\boldsymbol{G})\mathrm{Tr}(\boldsymbol{C}_{\mathrm{RC}})}{2} = (2g-h)\mu\sigma_v^2 \mathrm{Tr}(\boldsymbol{C}_{\mathrm{RC}}).$$

The steady-state MSD is computed by right-multiplying both sides of eq. (3.77) by $(\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{C}_{\mathrm{RC_{ext}}})^{-1}$ and taking the trace to obtain

$$\begin{aligned}\chi(\infty) &\approx \mu\sigma_v^2 \mathrm{Tr}(\boldsymbol{G}_{\mathrm{ext}})/2 \\ &= 4N(2g-h)\mu\sigma_v^2.\end{aligned}$$

In this case, the steady-state MSD can be easily obtained, since the structure of $\boldsymbol{C}_{\mathrm{RC_{ext}}}$ and $\boldsymbol{G}_{\mathrm{ext}}$ helps us to isolate $\mathrm{Tr}(\boldsymbol{S}(\infty))$ in equation (3.77). For other matrices, a closed expression may not be obtained.

Notice that we obtained simple equations to calculate the EMSE and the MSD for small step size, which depend only on N and on matrix $\boldsymbol{C}_{\mathrm{RC}}$, similarly to the LMS steady-state equations. In the next section, we improve the second-order model and obtain a range of values for μ which guarantee the convergence in the variance.

3.4.3.1 Choosing the step-size

Recalling eq. (3.75) and assuming that all variables are Gaussian, we can use properties of fourth-order Gaussian vectors [32] to obtain an approximation for $\boldsymbol{\mathcal{C}}$ (see eq.(3.73)) and improve the accuracy of the proposed model when the input vector is real-valued. For this purpose, assume that $\boldsymbol{q}_{\mathrm{RC}}(n)$ is a correlated Gaussian vector, and recall that the autocorrelation matrix $\boldsymbol{C}_{\mathrm{RC}}$ is symmetric and non-negative definite. This fact implies that there exists an orthogonal matrix $\boldsymbol{Z}$, such that

$$\boldsymbol{Z}\boldsymbol{Z}^T = \boldsymbol{Z}^T\boldsymbol{Z} = \boldsymbol{I}_{4N},$$

which diagonalizes $\boldsymbol{C}_{\mathrm{RC}}$, as given by

$$\boldsymbol{C}_{\mathrm{RC}} = \boldsymbol{Z}\boldsymbol{\Lambda}\boldsymbol{Z}^T, \tag{3.81}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix, $\mathrm{diag}(\boldsymbol{\Lambda}) = [\lambda_1\ \lambda_2 \ldots \lambda_{4N}]^T$, and $\lambda_i \geq 0$, $i = 1, 2, \ldots, 4N$ are the eigenvalues of $\boldsymbol{C}_{\mathrm{RC}}$. Defining $\boldsymbol{q}'_{\mathrm{RC}}(n) = \boldsymbol{Z}^T\boldsymbol{q}_{\mathrm{RC}}(n)$, then

$$E\{\boldsymbol{q}'_{\mathrm{RC}}(n)\boldsymbol{q}'^T_{\mathrm{RC}}(n)\} = E\{\boldsymbol{Z}^T\boldsymbol{q}_{\mathrm{RC}}(n)\boldsymbol{q}^T_{\mathrm{RC}}(n)\boldsymbol{Z}\} = \boldsymbol{Z}^T\boldsymbol{C}_{\mathrm{RC}}\boldsymbol{Z} = \boldsymbol{\Lambda}.$$

Since a linear transformation of a Gaussian vector is also Gaussian, $\boldsymbol{q}'_{\mathrm{RC}}(n)$ is a Gaussian vector whose elements are independent from each other. Define the extended matrix $\boldsymbol{Z}_{\mathrm{ext}} = \boldsymbol{I}_4 \otimes \boldsymbol{Z}$. Multiplying $\boldsymbol{\mathcal{C}}$ by $\boldsymbol{Z}^T_{\mathrm{ext}}$ on the left and by $\boldsymbol{Z}_{\mathrm{ext}}$ on the right, we obtain

$$\boldsymbol{\mathcal{C}}' = \boldsymbol{Z}^T_{\mathrm{ext}}\boldsymbol{\mathcal{C}}\boldsymbol{Z}_{\mathrm{ext}} = \boldsymbol{Z}^T_{\mathrm{ext}}\boldsymbol{G}_{\mathrm{ext}}E\{\left(\boldsymbol{I}_4 \otimes \boldsymbol{q}_{\mathrm{RC}}\boldsymbol{q}^T_{\mathrm{RC}}\right)\boldsymbol{S}(n)\left(\boldsymbol{I}_4 \otimes \boldsymbol{q}_{\mathrm{RC}}\boldsymbol{q}^T_{\mathrm{RC}}\right)\}\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{Z}_{\mathrm{ext}} \tag{3.82}$$

Defining $\boldsymbol{S}'(n) = \boldsymbol{Z}^T\boldsymbol{S}(n)\boldsymbol{Z}$ and noting that

$$\boldsymbol{Z}^T_{\mathrm{ext}}\boldsymbol{G}_{\mathrm{ext}}\boldsymbol{Z}_{\mathrm{ext}} = (\boldsymbol{I}_4 \otimes \boldsymbol{Z}^T)(\boldsymbol{G} \otimes \boldsymbol{I}_{4N})(\boldsymbol{I}_4 \otimes \boldsymbol{Z}) = \boldsymbol{G}_{\mathrm{ext}}$$

and

$$\begin{aligned}\boldsymbol{Z}^T_{\mathrm{ext}}\left(\boldsymbol{I}_4 \otimes \boldsymbol{q}_{\mathrm{RC}}\boldsymbol{q}^T_{\mathrm{RC}}\right)\boldsymbol{Z}_{\mathrm{ext}} &= (\boldsymbol{I}_4 \otimes \boldsymbol{Z})^T(\boldsymbol{I}_4 \otimes \boldsymbol{q}_{\mathrm{RC}}\boldsymbol{q}^T_{\mathrm{RC}})(\boldsymbol{I}_4 \otimes \boldsymbol{Z}) \\ &= \boldsymbol{I}_4 \otimes \boldsymbol{Z}^T\boldsymbol{q}_{\mathrm{RC}}\boldsymbol{q}^T_{\mathrm{RC}}\boldsymbol{Z} \\ &= \boldsymbol{I}_4 \otimes \boldsymbol{q}'_{\mathrm{RC}}\boldsymbol{q}'^T_{\mathrm{RC}},\end{aligned}$$

we rewrite (3.82) as

$$\boldsymbol{\mathcal{C}}' = \boldsymbol{G}_{\mathrm{ext}}E\left\{\left(\boldsymbol{I}_4 \otimes \boldsymbol{q}'_{\mathrm{RC}}\boldsymbol{q}'^T_{\mathrm{RC}}\right)\boldsymbol{S}'(n)\left(\boldsymbol{I}_4 \otimes \boldsymbol{q}'_{\mathrm{RC}}\boldsymbol{q}'^T_{\mathrm{RC}}\right)\right\}\boldsymbol{G}_{\mathrm{ext}}. \tag{3.83}$$

Each element of the matrix in the argument of (3.83) can be expressed as sums of terms

$$E\{q'_{\mathrm{RC}m_1}(n)q'_{\mathrm{RC}m_2}(n)q'_{\mathrm{RC}m_3}(n)q'_{\mathrm{RC}m_4}(n)\}s'_{ij}(n), \tag{3.84}$$

where the $q'_{\mathrm{RC}m}(n)$ represent the elements of $\boldsymbol{q}'_{\mathrm{RC}}(n)$. $s'_{ij}(n)$ are the entries of $\boldsymbol{S}'(n)$. Since the elements of $\boldsymbol{q}'_{\mathrm{RC}}(n)$ are independent and zero-mean, eq. (3.84) is different from zero only if $m_1 = m_2 = m_3 = m_4$, or $m_1 = m_2$ and $m_3 = m_4$, or $m_1 = m_3$ and $m_2 = m_4$, or $m_1 = m_4$ and $m_2 = m_3$. Using this result and eq. (3.81), after some algebra, eq. (3.83) reduces to

$$\boldsymbol{\mathcal{C}}' = \boldsymbol{G}_{\mathrm{ext}}\left[\boldsymbol{\Lambda}_{\mathrm{ext}}\mathrm{Tr}\left(\boldsymbol{S}'(n)\boldsymbol{\Lambda}_{\mathrm{ext}}\right) + 2\boldsymbol{\Lambda}_{\mathrm{ext}}\boldsymbol{S}'(n)\boldsymbol{\Lambda}_{\mathrm{ext}}\right]\boldsymbol{G}_{\mathrm{ext}}.$$

where $\boldsymbol{\Lambda}_{\text{ext}} = \boldsymbol{I}_4 \otimes \boldsymbol{\Lambda}$. Multiplying (3.75) by $\boldsymbol{Z}_{\text{ext}}^T$ on the right and by $\boldsymbol{Z}_{\text{ext}}$ on the left, one gets

$$\begin{aligned}\boldsymbol{S}'(n+1) \approx & \boldsymbol{S}'(n) - \mu\left[\boldsymbol{S}'(n)\boldsymbol{\Lambda}_{\text{ext}}\boldsymbol{G}_{\text{ext}} + \boldsymbol{G}_{\text{ext}}\boldsymbol{\Lambda}_{\text{ext}}\boldsymbol{S}'(n)\right] + \mu^2\sigma_v^2\boldsymbol{G}_{\text{ext}}\boldsymbol{\Lambda}_{\text{ext}}\boldsymbol{G}_{\text{ext}} \\ & + \mu^2\boldsymbol{G}_{\text{ext}}\left[\boldsymbol{\Lambda}_{\text{ext}}\text{Tr}(\boldsymbol{S}'(n)\boldsymbol{\Lambda}_{\text{ext}}) + 2\boldsymbol{\Lambda}_{\text{ext}}\boldsymbol{S}'(n)\boldsymbol{\Lambda}_{\text{ext}}\right]\boldsymbol{G}_{\text{ext}},\end{aligned} \tag{3.85}$$

with initialization $\boldsymbol{S}'(0) = \boldsymbol{Z}_{\text{ext}}^T\boldsymbol{w}_{\text{ext}}(0)\boldsymbol{w}_{\text{ext}}^T(0)\boldsymbol{Z}_{\text{ext}}$. Taking only the diagonal $\boldsymbol{s}'(n) = \text{diag}(\boldsymbol{S}'(n))$ in eq. (3.85), we obtain a simplified recursion

$$\boldsymbol{s}'(n+1) = \left[\boldsymbol{I}_{16N} - 2\mu\boldsymbol{\Lambda}_{\text{ext}}\boldsymbol{G}_{\text{ext}} + \mu^2\boldsymbol{G}_{\text{ext}}^2\boldsymbol{\ell}_{\text{ext}}\boldsymbol{\ell}_{\text{ext}}^T + 2\mu^2\boldsymbol{G}_{\text{ext}}^2\boldsymbol{\Lambda}_{\text{ext}}^2\right]\boldsymbol{s}'(n) + \mu^2\sigma_v^2\boldsymbol{G}_{\text{ext}}^2\boldsymbol{\ell}_{\text{ext}}, \tag{3.86}$$

where

$$\boldsymbol{\ell}_{\text{ext}} = \text{col}(1,\ 1,\ 1,\ 1) \otimes \text{diag}(\boldsymbol{\Lambda})$$

and $\boldsymbol{s}'(0) = \text{diag}(\boldsymbol{S}'(0))$.

We can study the system matrix of eq. (3.86), i.e.,

$$\begin{aligned}\boldsymbol{\Gamma} &= \boldsymbol{I}_{16N} - 2\mu\boldsymbol{\Lambda}_{\text{ext}}\boldsymbol{G}_{\text{ext}} + \mu^2\boldsymbol{G}_{\text{ext}}^2\boldsymbol{\ell}_{\text{ext}}\boldsymbol{\ell}_{\text{ext}}^T + 2\mu^2\boldsymbol{G}_{\text{ext}}^2\boldsymbol{\Lambda}_{\text{ext}}^2 \\ &= (\boldsymbol{I}_{16N} - \mu\boldsymbol{\Lambda}_{\text{ext}}\boldsymbol{G}_{\text{ext}})^2 + \mu^2\boldsymbol{G}_{\text{ext}}^2\boldsymbol{\ell}_{\text{ext}}\boldsymbol{\ell}_{\text{ext}}^T + \mu^2\boldsymbol{G}_{\text{ext}}^2\boldsymbol{\Lambda}_{\text{ext}}^2\end{aligned}$$

to define a bound for the step size which guarantees the stability in the variance. Using the ℓ_1-norm [46], we can find an upper bound for the largest eigenvalue λ_{Γ_l} of $\boldsymbol{\Gamma}$, i.e.,

$$\max_{1\le l\le 16N}|\lambda_{\Gamma_l}| \le ||\boldsymbol{\Gamma}||_1 = \max_{1\le l\le 16N}\sum_{\text{m}=1}^{16N}|\gamma_{l\text{m}}|,$$

where $\gamma_{l\text{m}}$ are the elements of $\boldsymbol{\Gamma}$. A conservative range of values for μ, which guarantees stability, requires that $||\boldsymbol{\Gamma}||_1 < 1$. Observe that the l-th column of $\boldsymbol{\Gamma}$ has entries

$$\begin{cases}\mu^2 g_{\text{ext}_{l,l}}^2\lambda_{\text{ext}_l}\lambda_{\text{ext}_m}, & \text{if } l \ne m \\ (1-\mu g_{\text{ext}_{l,l}}\lambda_{\text{ext}_l})^2 + \mu^2 g_{\text{ext}_{l,l}}^2\lambda_{\text{ext}_l}^2 + \mu^2 g_{\text{ext}_{l,l}}^2\lambda_{\text{ext}_l}\sum_{m=1}^{16N}\lambda_{\text{ext}_m}, & \text{if } l = m,\end{cases} \tag{3.87}$$

where $g_{\text{ext}_{l,l}}$ are the diagonal elements of $\boldsymbol{G}_{\text{ext}}$. Note that

$$\begin{aligned}\sum_{\text{m}=1}^{16N}|\gamma_{l\text{m}}| &= (1-\mu g_{\text{ext}_{l,l}}\lambda_{\text{ext}_l})^2 + \mu^2 g_{\text{ext}_{l,l}}^2\left(\lambda_{\text{ext}_l}^2 + \lambda_{\text{ext}_l}\sum_{\text{m}=1}^{16N}\lambda_{\text{ext}_\text{m}}\right) \\ &= (1-\mu g_{\text{ext}_{l,l}}\lambda_{\text{ext}_l})^2 + \mu^2 g_{\text{ext}_{l,l}}^2\left(\lambda_{\text{ext}_l}^2 + \lambda_{\text{ext}_l}\text{Tr}(\boldsymbol{\Lambda}_{\text{ext}})\right).\end{aligned} \tag{3.88}$$

Thus, the recursion in eq. (3.86) is stable if

$$(1-\mu g_{\text{ext}_{l,l}}\lambda_{\text{ext}_l})^2 + \mu^2 g_{\text{ext}_{l,l}}^2\left(\lambda_{\text{ext}_l}^2 + \lambda_{\text{ext}_l}\text{Tr}(\boldsymbol{\Lambda}_{\text{ext}})\right) < 1,$$

for $1 \leq l \leq 16N$. Recalling that

$$\mathrm{Tr}(\boldsymbol{C}_{\mathrm{RC_{ext}}}) = \mathrm{Tr}(\boldsymbol{\Lambda}_{\mathrm{ext}}) = \mathrm{Tr}(\boldsymbol{I}_4 \otimes \boldsymbol{C}_{\mathrm{RC}}) = 4\mathrm{Tr}(\boldsymbol{C}_{\mathrm{RC}})$$

and after some manipulation, the condition simplifies to

$$\mu < 1/g_{\mathrm{ext}_{l,l}}(\lambda_{\mathrm{ext}_l} + 2\mathrm{Tr}(\boldsymbol{C}_{\mathrm{RC}})), \quad 1 \leq l \leq 16N.$$

The smallest bound occurs for $g_{\mathrm{ext}_{l,l}} = \max\{(g+h),(g-h)\}$ and $\lambda_{\mathrm{ext}_l} = \lambda_{\mathrm{ext}_{\max}}$:

$$\mu < 1/(\max\{(g+h),(g-h)\}(\lambda_{\mathrm{ext}_{\max}} + 2\mathrm{Tr}(\boldsymbol{C}_{\mathrm{RC}}))). \tag{3.89}$$

Replacing $\lambda_{\mathrm{ext}_{\max}}$ by $\mathrm{Tr}(\boldsymbol{C}_{\mathrm{RC}})$ we obtain a simpler but more conservative condition for stability,

$$0 < \mu < 1/(3\mathrm{Tr}(\boldsymbol{C}_{\mathrm{RC}})\max\{(g+h),(g-h)\}),$$

which guarantees stability in the variance. For RC-WL-iQLMS, the step-size selection must respect

$$0 < \mu_{\mathrm{RC_{iQ}}} < 4/9\mathrm{Tr}(\boldsymbol{C}_{\mathrm{RC}}), \tag{3.90}$$

since $\max\{(g+h),(g-h)\} = 3/4$.

Note that the analysis proposed in this section is valid for uncorrelated and correlated input. The results presented here can be extended to other real-regressor quaternion algorithms.

3.5 Simulations

To compare the algorithms and show the accuracy of the proposed model, we performed some simulations using $\mathbb{Q}$-improper processes. For this purpose, we define the elements of $\boldsymbol{q}(n)$ as given by

$$\begin{aligned}
\boldsymbol{q}_{\mathrm{R}}(n) &= (\sqrt{0.58}\boldsymbol{\rho}_1(n) + 0.5\boldsymbol{\rho}_2(n) + 0.4\boldsymbol{\rho}_3(n) + 0.1\boldsymbol{\rho}_4(n))/4 \\
\boldsymbol{q}_{\mathrm{I}}(n) &= (0.2\boldsymbol{\rho}_1(n) + 0.9\boldsymbol{\rho}_2(n) + 0.3\boldsymbol{\rho}_3(n) + \sqrt{0.06}\boldsymbol{\rho}_4(n))/4 \\
\boldsymbol{q}_{\mathrm{J}}(n) &= (0.4\boldsymbol{\rho}_1(n) + 0.5\boldsymbol{\rho}_2(n) + 0.7\boldsymbol{\rho}_3(n) + \sqrt{0.1}\boldsymbol{\rho}_4(n))/4 \\
\boldsymbol{q}_{\mathrm{K}}(n) &= (0.3\boldsymbol{\rho}_1(n) + 0.1\boldsymbol{\rho}_2(n) + 0.2\boldsymbol{\rho}_3(n) + \sqrt{0.86}\boldsymbol{\rho}_4(n))/4,
\end{aligned}$$

where each $\boldsymbol{\rho}_l(n)$ is a 4×1 vector. The elements of each $\boldsymbol{\rho}_l(n)$ are zero-mean, Gaussian, i.i.d. and with unitary variance. $\boldsymbol{\rho}_l(n)$ and $\boldsymbol{\rho}_m(n)$ are independent from each other $\forall l \neq m$. Using these values for the input, the components of each quaternion in $\boldsymbol{q}(n)$ are

correlated, but different quaternions are uncorrelated. One can check that for this input, $\boldsymbol{C}_{\boldsymbol{q}\boldsymbol{q}^i} = \boldsymbol{C}_{\boldsymbol{q}\boldsymbol{q}^j} = \boldsymbol{C}_{\boldsymbol{q}^i\boldsymbol{q}^j} \neq \boldsymbol{0}$, such that $\boldsymbol{q}(n)$ is $\mathbb{Q}$-improper. We also define $d(n)$ so that $d(n)$ and $\boldsymbol{q}(n)$ are jointly $\mathbb{Q}$-improper processes. $d(n)$ is obtained with (3.35), and to compute the 16-entry column quaternion vector $\boldsymbol{w}_{\text{WL,opt}} = \text{col}(\boldsymbol{w}_{\text{opt,1}},\ \boldsymbol{w}_{\text{opt,2}},\ \boldsymbol{w}_{\text{opt,3}},\ \boldsymbol{w}_{\text{opt,4}})$, we define

$$f(l, b, c) = 1 - \cos(2\pi(l - b)/4c), \tag{3.91}$$

to obtain

$$\begin{aligned}
\boldsymbol{w}_{\text{opt,1}_l} &= \beta_1(f(l,3,1) - \boldsymbol{i}f(l,3,3) - \boldsymbol{j}f(l,3,5) - \boldsymbol{k}f(l,3,7)) \\
\boldsymbol{w}_{\text{opt,2}_l} &= \beta_2(f(l,5,1) - \boldsymbol{i}f(l,5,3) - \boldsymbol{j}f(l,5,5) - \boldsymbol{k}f(l,5,7)) \\
\boldsymbol{w}_{\text{opt,3}_l} &= \beta_3(f(l,7,1) - \boldsymbol{i}f(l,7,3) - \boldsymbol{j}f(l,7,5) - \boldsymbol{k}f(l,7,7)) \\
\boldsymbol{w}_{\text{opt,4}_l} &= \beta_4(f(l,11,1) - \boldsymbol{i}f(l,11,3) - \boldsymbol{j}f(l,11,5) - \boldsymbol{k}f(l,11,7)),
\end{aligned}$$

where $\boldsymbol{w}_{\text{opt},m_l}$ is the l-th ($l = 1, \ldots, 4$) entry of $\boldsymbol{w}_{\text{opt},m}$. $\beta_1 = 1/||\boldsymbol{w}_{\text{opt,1}}||_2$ and $\beta_m = 0.1/||\boldsymbol{w}_{\text{opt},m}||_2$, $m = 2, 3, 4$. Note that we choose these values for the entries of $\boldsymbol{w}_{\text{WL,opt}}$, since this results in an example for which the SL filter still has reasonable performance.

The quaternion elements of $v(n)$ are zero-mean, Gaussian and i.i.d, with equal variance $\sigma^2_{v_\alpha} = 10^{-4}/4$, $\alpha \in \{\text{R, I, J, K}\}$. We perform 200 simulations to observe the EMSE and the MSD. We compare QLMS, WL-QLMS, WL-iQLMS, RC-WL-QLMS, RC-WL-iQLMS and 4-Ch-LMS, and we plot our model for RC-WL-iQLMS and RC-WL-QLMS. We perform two simulations. In the first, we adjust the WL-iQLMS and the QLMS algorithms to have the same convergence rate, and adjust the other WL-algorithms (and the 4-Ch-LMS) to achieve the same steady-state EMSE achieved by WL-iQLMS. In the second simulation, we adjust all the WL techniques (and 4-Ch-LMS) to achieve the same steady-state MSD performance. Table 11 shows the step sizes used in our simulations.

Table 11: Step-sizes used in the simulations ($\mu = 10^{-3}$)

Algorithm	QLMS	WL-QLMS	WL-iQLMS	RC-WL-QLMS	RC-WL-iQLMS	4-Ch-LMS
Step size (1^{th} case)	μ	2.4μ	2μ	9.6μ	8μ	6μ
Step size (2^{nd} case)	μ	9.6μ	2μ	9.6μ	8μ	6μ

Figures 4 and 5 show the results for the first scenario, while figures 6 and 7 present the results in the second approach.

Note that in both scenarios the WL algorithms achieve lower EMSE and MSD performances than QLMS, as expected, and that the WL-iQLMS, 4-Ch-LMS and RC-WL-

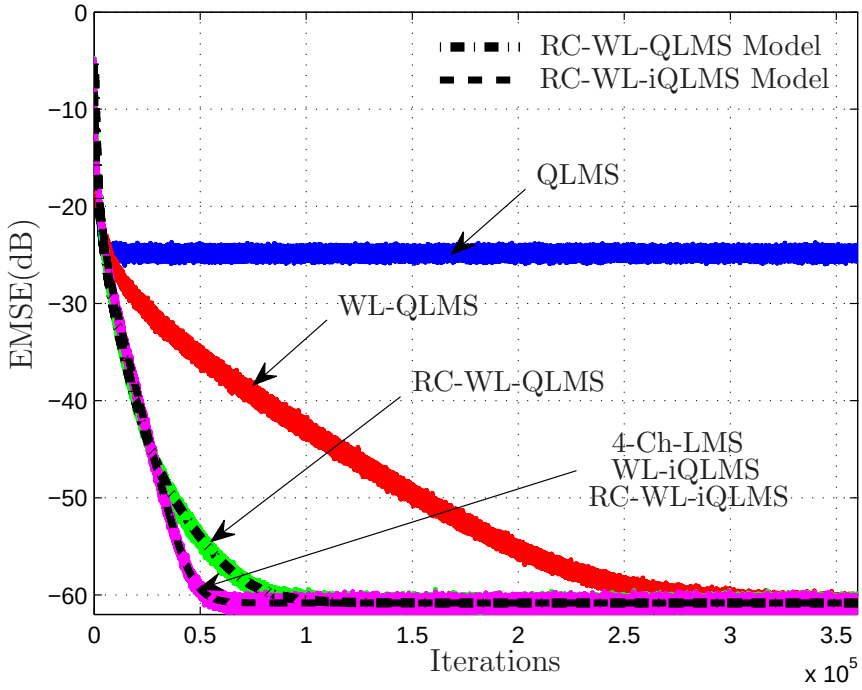

Figure 4: EMSE performance comparing QLMS, WL-QLMS, RC-WL-QLMS, RC-WL-iQLMS, WL-iQLMS, 4-Ch-LMS and the proposed second-order model, when the WL techniques are adjusted to achieve the same steady-state EMSE. The steady-state EMSE corresponds to -65 dB for RC-WL-QLMS and RC-WL-iQLMS. – Average of 200 simulations.

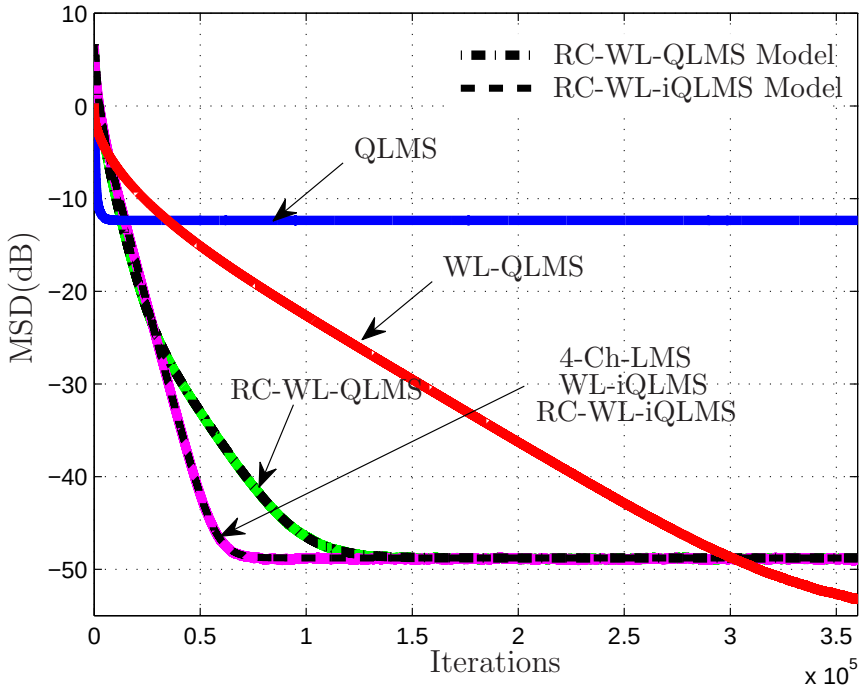

Figure 5: MSD performance comparing QLMS, WL-QLMS, RC-WL-QLMS, RC-WL-iQLMS, WL-iQLMS, 4-Ch-LMS and the proposed second-order model, when the WL techniques are adjusted to achieve the same steady-state MSD. The steady-state MS corresponds to -47dB for RC-WL-QLMS and RC-WL-iQLMS. – Average of 200 simulations.

iQLMS have the same convergence rate. In the first scenario (see figures 4 and 5), when all the WL techniques are adjusted to achieve the same EMSE steady-state, RC-WL-iQLMS is the fastest-converging technique, but in terms of MSD steady-state performance, it is outperformed by WL-QLMS. In the second scenario, (see figures 6 and 7), the step sizes of the WL algorithms are adjusted to achieve the same steady-state MSD. The RC-WL-iQLMS algorithm again is the fastest-converging technique in both figures, but the WL-QLMS algorithm has a convergence rate much closer to that of RC-WL-iQLMS. However, the steady-state EMSE performance of WL-QLMS is poor in this case. The proposed model (presented in the figures as a black dashed line) is accurate to describe the performance behavior of both the RC-WL-QLMS and the RC-WL-iQLMS algorithms. We can also verify that the model is accurate to predict the faster convergence rate of the RC-WL-iQLMS algorithm when compared to RC-WL-QLMS.

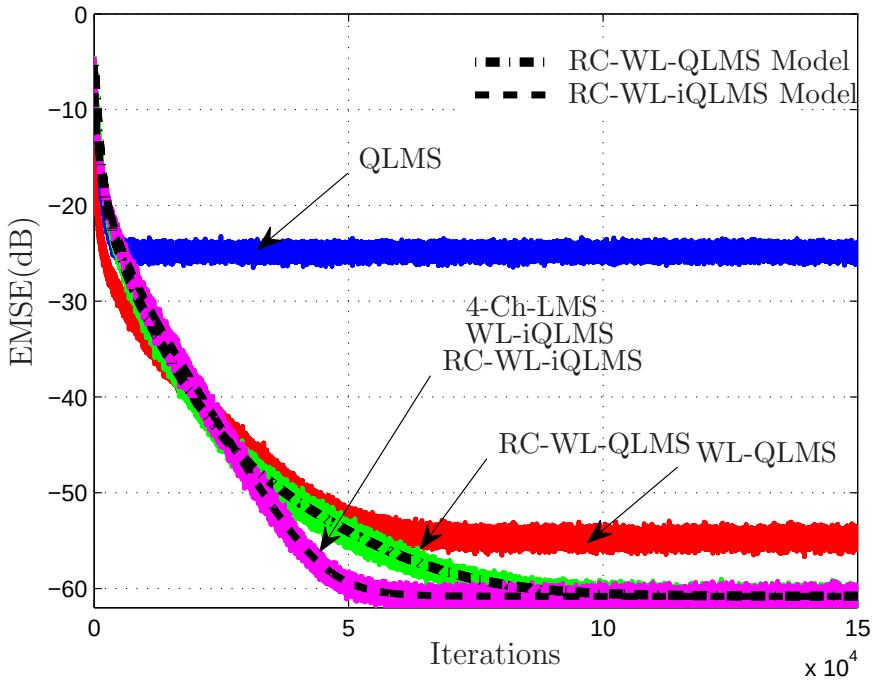

Figure 6: EMSE performance comparing QLMS, WL-QLMS, RC-WL-QLMS, RC-WL-iQLMS, WL-iQLMS, 4-Ch-LMS and the proposed second-order model, when the WL techniques are adjusted to achieve the same steady-state MSD. The steady-state EMSE corresponds to -65 dB for RC-WL-QLMS and RC-WL-iQLMS. – Average of 200 simulations.

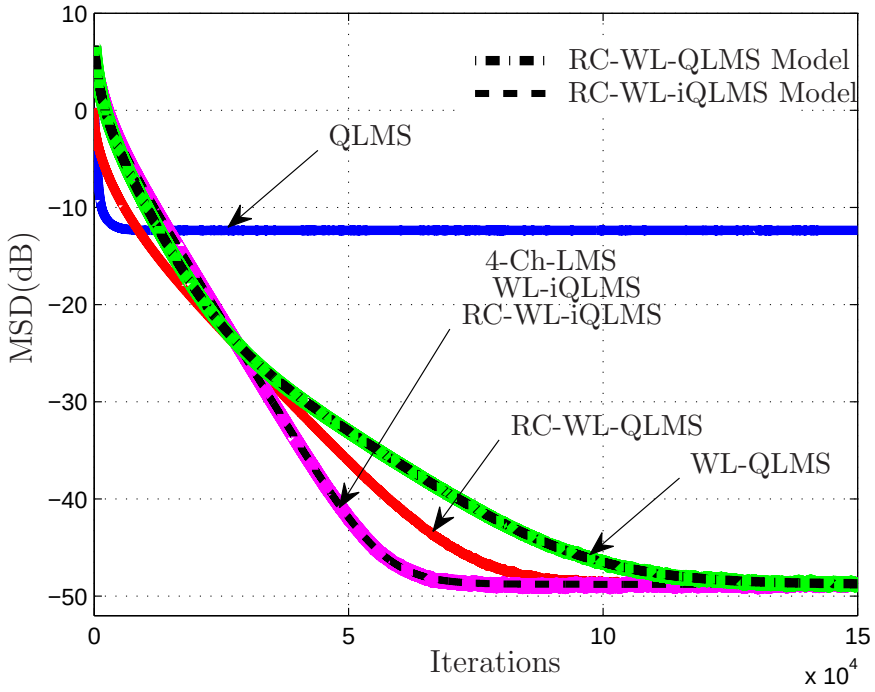

Figure 7: MSD performance comparing QLMS, WL-QLMS, RC-WL-QLMS, RC-WL-iQLMS, WL-iQLMS, 4-Ch-LMS and the proposed second-order model, when the WL techniques are adjusted to achieve the same steady-state MSD. The steady-state MS corresponds to -47dB for RC-WL-QLMS and RC-WL-iQLMS. – Average of 200 simulations.

3.6 Conclusions

In this chapter, we developed a general representation to quaternion gradients and we used it to prove that different gradients can be applied to obtain the same QLMS-like algorithm.

We showed that the class of gradients to which the i-gradient belongs part provides the fastest-converging WL algorithms when there is correlation only in two axis, or when the WL regressor vector is real. Using the general gradient, we devised the fastest-converging WL-QLMS algorithm with a real regressor vector, and we showed that it corresponds to a RC version of WL-iQLMS and also to the 4-Ch-LMS algorithm written in the quaternion domain. The RC-WL-iQMS technique has one-fourth of the complexity of WL-iQLMS, and has the same cost of the 4-Ch-LMS algorithm

We developed a mean and a mean-square analysis for any QLMS-based algorithm, and we applied it to study WL-QLMS algorithms using real data vector. The proposed analysis is suitable for correlated and uncorrelated input. We obtained simple equations to compute the steady-state EMSE and MSD, and a range of step sizes for mean and mean-

squared convergence. We also proposed a model to study the convergence performance of any WL-QLMS-based algorithm using real regressor vector, which was particularized for the RC-WL-iQLMS technique. Simulations comparing the algorithms and the model showed the accuracy of the analysis.

Appendix A: Conditions to guarantee the positive semi-definiteness of $\mathbf{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}}$

In this appendix we show that $\boldsymbol{C}_{\text{ext}}$ is a positive semi-definite matrix. Then, we use this to prove that the diagonal entries of $\mathbf{G}_{\text{ext}}$ must be non-negative to make $\mathbf{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}}$ always positive semi-definite .

Initially, notice that for any quaternion vector $\boldsymbol{\chi}$ with the proper dimension,

$$\boldsymbol{\chi}^H E\{\boldsymbol{\varphi}(n)\boldsymbol{\varphi}^H(n)\}\boldsymbol{\chi} = E\{|\boldsymbol{\varphi}^H(n)\boldsymbol{\chi}|^2\} \geq 0, \ \ \forall \boldsymbol{\chi}. \tag{3.92}$$

Define vector $\boldsymbol{\chi}_{\text{ext}} = \text{col}(\ \boldsymbol{\chi}_{\text{R}}, \ \ \boldsymbol{\chi}_{\text{I}}, \ \ \boldsymbol{\chi}_{\text{J}}, \ \ \boldsymbol{\chi}_{\text{K}}\ )$, and consider the product $\boldsymbol{\chi}_{\text{ext}}^T\boldsymbol{C}_{\text{ext}}\boldsymbol{\chi}_{\text{ext}}$, for $\boldsymbol{\chi}_{\text{ext}} \in \mathbb{R}^{4M}$. It then holds

$$\boldsymbol{\chi}_{\text{ext}}^T\boldsymbol{C}_{\text{ext}}\boldsymbol{\chi}_{\text{ext}} = \boldsymbol{\chi}^H E\{\boldsymbol{\varphi}(n)\boldsymbol{\varphi}^H(n)\}\boldsymbol{\chi} \geq 0.$$

To define the conditions on $\mathbf{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}}$ which lead to stable (3.51), we must find values for which the diagonal entries of $\mathbf{G}_{\text{ext}}$ make the product of matrices positive semi-definite. First, note that using the similarity transformation [46]

$$\mathbf{G}_{\text{ext}}^{-1/2}(\mathbf{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}})\mathbf{G}_{\text{ext}}^{1/2} = \mathbf{G}_{\text{ext}}^{1/2}\boldsymbol{C}_{\text{ext}}\mathbf{G}_{\text{ext}}^{1/2}, \tag{3.93}$$

the eigenvalues of $\mathbf{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}}$ are preserved in

$$\mathbf{G}_{\text{ext}}^{1/2}\boldsymbol{C}_{\text{ext}}\mathbf{G}_{\text{ext}}^{1/2}. \tag{3.94}$$

In this case, we just need to define the conditions on which (3.94) is positive semi-definite to show that the elements of $\mathbf{G}_{\text{ext}}$ should be non-negative. For this purpose, we use $\boldsymbol{\chi}_{\text{ext}}$ to check on which cases

$$\boldsymbol{\chi}_{\text{ext}}^H\mathbf{G}_{\text{ext}}^{1/2}\boldsymbol{C}_{\text{ext}}\mathbf{G}_{\text{ext}}^{1/2}\boldsymbol{\chi}_{\text{ext}} \geq 0.$$

We start showing that if $\mathbf{G}_{\text{ext}}$ has at least one negative diagonal entry, then it is possible to find one vector for which $\boldsymbol{\chi}_{\text{ext}}^H\mathbf{G}_{\text{ext}}^{1/2}\boldsymbol{C}_{\text{ext}}\mathbf{G}_{\text{ext}}^{1/2}\boldsymbol{\chi}_{\text{ext}} < 0$. For this purpose, assume that only the m-th entry $g_{\text{ext}_{m,m}}$ of $\mathbf{G}_{\text{ext}}$ is negative, such that

$$\mathbf{G}_{\text{ext}}^{1/2} = \text{diag}(\sqrt{g_{\text{ext}_{1,1}}}, \ \ldots, \ \boldsymbol{j}\sqrt{|g_{\text{ext}_{m,m}}|}, \ \ldots, \ \sqrt{g_{\text{ext}_{4M,4M}}}\ ),$$

where $\boldsymbol{j} = \sqrt{-1}$. Using $\boldsymbol{\chi}_{\text{ext}} = \boldsymbol{\epsilon}_m$, where $\epsilon_m = 1$ and the other entries are set to zero, one can easily show

$$\boldsymbol{\epsilon}_m^T\mathbf{G}_{\text{ext}}^{1/2}\boldsymbol{C}_{\text{ext}}\mathbf{G}_{\text{ext}}^{1/2}\boldsymbol{\epsilon}_m = g_{\text{ext}_{m,m}}c_{\text{ext}_{m,m}} < 0$$

so that (3.94) is not positive semi-definite. When $\mathbf{G}_{\text{ext}}$ has more negative diagonal entries,

the proof that (3.94) is non-positive is straightforward. In this case, all $g_{\text{ext}_{m,m}}$ must be non negative, $1 \leq m \leq 4M$, to guarantee $\mathbf{G}_{\text{ext}}\boldsymbol{C}_{\text{ext}}$ positive semi-definite always. The condition for this is $(g + h) \geq 0$ and $(g - h) \geq 0$.

PART II

LOW-COMPLEXITY TECHNIQUES USING THE DCD ALGORITHM

4 LOW-COMPLEXITY WIDELY-LINEAR ADAPTIVE FILTERS USING THE DICHOTOMOUS COORDINATE DESCENT (DCD) ALGORITHM

It is well-known that the RLS algorithm [32] converges faster than the LMS algorithm [32], and that RLS is also less sensitive to the eigenvalue spread in the autocorrelation matrix. However, while the LMS has linear computational cost, traditional implementations of the RLS (for instance [32], p.201) have quadratic computational complexity with the regressor vector length N ($\boldsymbol{\mathcal{O}}(N^2)$). Additionally, the RLS algorithm suffers with numerical instability [24], requiring a careful implementation to avoid divergence.

There are faster ways to implement RLS, such as the Fast Transversal Filter (FTF) [6, 65], lattice approaches [65] and techniques using the QR decomposition [65, 66]. However, even these methods have limitations. The FTF is the less costly technique (its computational cost is proportional to $7N$), but it is numerically unstable. The lattice approaches (with complexity about $13N$) and the QR-based techniques (which can be implemented at cost $\boldsymbol{\mathcal{O}}(N)$ if the input is a tap delay-line) can be implemented in stable ways, but they require a careful implementation. Additionally, lattice algorithms compute only the error $e(n)$: they do not explicitly compute the weight vector $\boldsymbol{w}(n)$, such that they cannot be applied to problems which require $\boldsymbol{w}(n)$.

A low-cost and stable alternative to these techniques was proposed in [25], based on the dichotomous coordinate descent algorithm (DCD) [23, 28]. The DCD is an iterative method to solve a system of equations given by

$$\boldsymbol{R}\boldsymbol{w} = \boldsymbol{p}, \tag{4.1}$$

where $\boldsymbol{R}$ is a positive definite matrix, $\boldsymbol{p}$ is vector, and $\boldsymbol{w}$ is unknown. It is designed to avoid multiplications and divisions, costly to compute. These operations are replaced by additions, comparisons and bit-shifts, thus DCD is suitable for fixed-point implementation in hardware, such as in FPGAs [25]. For the RLS problem, it can be directly applied to

solve the normal equations at each time instant n. However, this approach is costly, since many DCD iterations are required to approximate the solution, and is avoided. To solve the RLS problem, the DCD-RLS technique [25] uses the solution obtained in the previous instant $n-1$ as the initial condition at instant n, such that only a low-cost update to the previous solution can be computed with a reduced number of DCD iterations. Avoiding costly computations, the DCD-RLS can be implemented with a cost linear in N. Additionally, it does not propagate the inverse autocorrelation matrix, thus avoiding the numerical instability of the traditional RLS algorithm.

Prior work also reports the use of DCD iterations to obtain a low-complexity affine projection (AP) algorithm [27], which incorporates the DCD to update the filter weights, resulting in a method less costly to implement than NLMS [45]. In [67], the DCD is used to implement the MVDR beamformer, and in [68], a DCD-homotopy technique based on the algorithm of [69] is also proposed to recover sparse signals, in system identification problems. All these results from the literature motivated us to study and propose low-cost techniques using DCD iterations, as presented in [29, 40].

In this chapter, we use DCD iterations to further reduce the complexity of the RC-WL-RLS, presented in Chapter 2. In that chapter, we used a linear transformation to rewrite the complex WL input data vector as a real-valued vector, which helped us to avoid redundance in the the autocorrelation matrix. Using this approach, we arrived at algorithms with the same MSE performance as that obtained with the original WL techniques, but less costly. Despite the complexity reduction achieved by RC-WL-RLS, it is still costly, since it uses $\mathcal{O}(N^2)$ computations (see Table 6, in page 40). Additionally, the numerical instability problem of WL-RLS remained unsolved for RC-WL-RLS. To improve this technique, we modify it to use DCD iterations, and a lower-complexity and also numerically stable method for widely-linear processing is obtained. The new technique is adequate for cases where the real and the imaginary entries of the input vector are tap-delay lines, so that a low-cost update of the estimated correlation matrix can be applied. We compare the proposed technique (the DCD-WL-RLS algorithm) with RC-WL-RLS, and show that the former can be designed to achieve almost the same performance MSE of RC-WL-RLS, but with complexity proportional to N.

The main contributions in this chapter are summarized as follows:

1. We extend the DCD algorithm so it can be applied to WL-RLS algorithms.
2. We develop a $\mathcal{O}(N)$ numerically stable version of the RC-WL-RLS algorithm.

The chapter is organized as follows. In Section 4.1 we present the DCD algorithm. Based on the RC-WL-RLS algorithm, in Section 4.2 we obtain the DCD-WL-RLS algorithm, and we also evaluate its computational complexity. Simulations are presented in Section 4.3, and the results of this chapter are summarized in Section 4.4.

4.1 The DCD algorithm

The DCD method has the goal of minimizing the multi-variable cost function [24]

$$J(\boldsymbol{w}) = \frac{1}{2}\boldsymbol{w}^T\boldsymbol{R}\boldsymbol{w} - \boldsymbol{p}^T\boldsymbol{w} + \boldsymbol{c}, \tag{4.2}$$

where $\boldsymbol{R}$, $\boldsymbol{p}$ and $\boldsymbol{c}$ are assumed constant. $\boldsymbol{R}$ is a positive definite matrix, such that $J(\boldsymbol{w})$ is a convex function.

Assume that an initial approximation to the solution $\hat{\boldsymbol{w}}(0)$ is available, and define $\hat{\boldsymbol{w}}^{(0)}(0) = \hat{\boldsymbol{w}}(0)$. DCD searches for a better nearby approximation, updating one entry of $\hat{\boldsymbol{w}}$ at a time. The first step is to evaluate

$$\Delta J_+ = J(\hat{\boldsymbol{w}}^{(0)}(0) + \boldsymbol{\epsilon}_1 H) - J(\hat{\boldsymbol{w}}^{(0)}(0)) \tag{4.3}$$

and

$$\Delta J_- = J(\hat{\boldsymbol{w}}^{(0)}(0) - \boldsymbol{\epsilon}_1 H) - J(\hat{\boldsymbol{w}}^{(0)}(0)), \tag{4.4}$$

where $H > 0$ is a step size and $\boldsymbol{\epsilon}_k$ is an $N \times 1$ vector, where all entries are 0, except for the element in position k, which is equal to 1. Denote $\hat{\boldsymbol{w}}^{(1)}(0)$ the improved estimate after $\hat{\boldsymbol{w}}^{(0)}(0)$. The update is obtained as follows:

1. If $\Delta J_+ < 0$, then the new estimate is $\hat{\boldsymbol{w}}^{(1)}(0) = \hat{\boldsymbol{w}}^{(0)}(0) + \boldsymbol{\epsilon}_1 H$.
2. If $\Delta J_- < 0$, then the new estimate is $\hat{\boldsymbol{w}}^{(1)}(0) = \hat{\boldsymbol{w}}^{(0)}(0) - \boldsymbol{\epsilon}_1 H$.
3. When ΔJ_+ and ΔJ_- are both positive, then $\hat{\boldsymbol{w}}^{(1)}(0) = \hat{\boldsymbol{w}}^{(0)}(0)$ and the algorithm moves to the next entry of $\hat{\boldsymbol{w}}^{(1)}(0)$, to verify if it needs to be updated.[1]

Figure 8 shows the first iteration when $\boldsymbol{w}$ is a scalar and (4.2) reduces to finding the minimum of a parabola.

Once $\hat{\boldsymbol{w}}^{(1)}(0)$ is computed, $\hat{\boldsymbol{w}}^{(2)}(0)$ can be obtained, checking $J(\hat{\boldsymbol{w}}^{(1)}(0) \pm \boldsymbol{\epsilon}_2 H) - J(\hat{\boldsymbol{w}}^{(1)}(0))$. After that, one can use the same steps to calculate $\hat{\boldsymbol{w}}^{(3)}(0)$ from $\hat{\boldsymbol{w}}^{(2)}(0)$,

[1]Note that ΔJ_+ and ΔJ_- cannot be simultaneously negative, since $J(\boldsymbol{w})$ is a convex function.

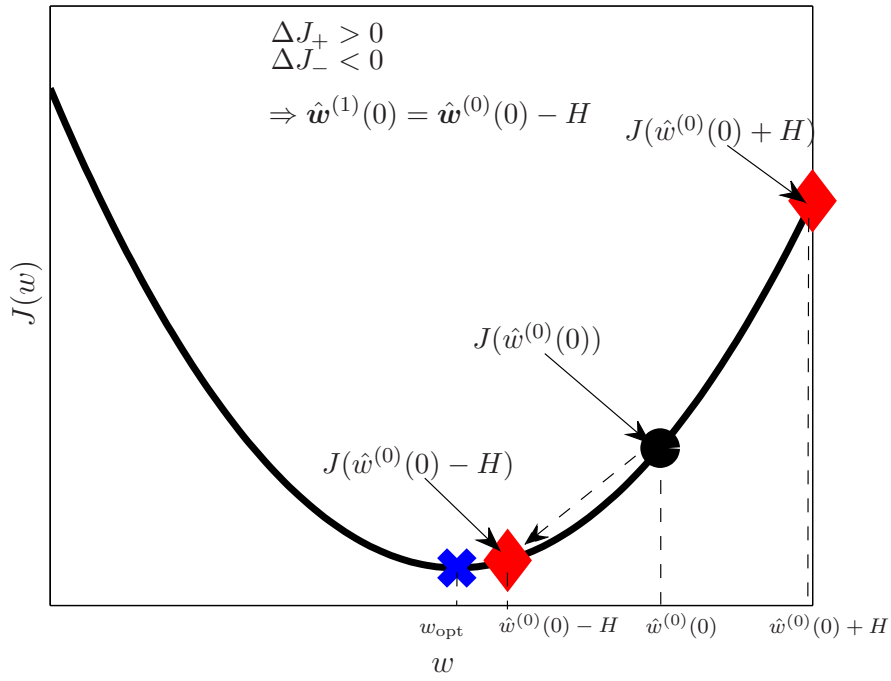

Figure 8: First DCD iteration when $\boldsymbol{w}$ has only one element and the minimization problem reduces to obtaining the minimum of a parabola. The algorithm starts with $\hat{w}^{(0)}(0)$ and computes $\Delta J_+ = J(\hat{w}^{(0)}(0) + H) - J(\hat{w}^{(0)}(0))$ and $\Delta J_- = J(\hat{w}^{(0)}(0) - H) - J(\hat{w}^{(0)}(0))$. Since $\Delta J_+ > 0$ and $\Delta J_- < 0$, $\hat{w}^{(0)}(0) - H$ is chosen as the improved solution $\hat{w}^{(1)}(0)$.

and so on. The algorithm continues the computation of $\hat{\boldsymbol{w}}^{(k)}(0)$ (for $k = 1, 2, \cdots N$) until $\hat{\boldsymbol{w}}^{(N)}(0)$ is obtained. In this case, we let $\hat{\boldsymbol{w}}^{(0)}(1) = \hat{\boldsymbol{w}}^{(N)}(0)$, and repeat the procedure. If at least one update occurred, the step size is kept with the same value. Otherwise, H is reduced to $H/2$, and the update continues.

Note that the computation of ΔJ_+ and ΔJ_- in each step is costly, since to obtain the i-th update $\hat{\boldsymbol{w}}^{(k+1)}(i)$ we must compute[2]

$$\begin{aligned}\Delta J_+ &= \frac{1}{2}\left(\hat{\boldsymbol{w}}^{(k)}(i) + H\boldsymbol{\epsilon}_{k+1}\right)^T \boldsymbol{R}\left(\hat{\boldsymbol{w}}^{(k)}(i) + H\boldsymbol{\epsilon}_{k+1}\right) - \boldsymbol{p}^T\left(\hat{\boldsymbol{w}}^{(k)}(i) + H\boldsymbol{\epsilon}_{k+1}\right) \\ &\quad - \left(\frac{1}{2}\hat{\boldsymbol{w}}^{(k)T}(i)\boldsymbol{R}\hat{\boldsymbol{w}}^{(k)}(i) - \boldsymbol{p}^T\hat{\boldsymbol{w}}^{(k)}(i)\right) \\ &= H\boldsymbol{\epsilon}_{k+1}^T\boldsymbol{R}\hat{\boldsymbol{w}}^{(k)}(i) + \frac{1}{2}H^2\boldsymbol{\epsilon}_{k+1}^T\boldsymbol{R}\boldsymbol{\epsilon}_{k+1} - H\boldsymbol{p}^T\boldsymbol{\epsilon}_{k+1},\end{aligned} \tag{4.5}$$

and, similarly,

$$\Delta J_- = -H\boldsymbol{\epsilon}_{k+1}^T\boldsymbol{R}\hat{\boldsymbol{w}}^{(k)}(i) + \frac{1}{2}H^2\boldsymbol{\epsilon}_{k+1}^T\boldsymbol{R}\boldsymbol{\epsilon}_{k+1} + H\boldsymbol{p}^T\boldsymbol{\epsilon}_{k+1}. \tag{4.6}$$

For ΔJ_+, the term $\boldsymbol{\epsilon}_{k+1}^T\boldsymbol{R}\hat{\boldsymbol{w}}^{(k)}(i)$ is computed with N multiplications and $N-1$ additions,

[2]We assume that $\boldsymbol{R}$ and $\boldsymbol{p}$ are real quantities in the following equations. The extension to the complex field uses a similar argument.

since it is equivalent to the product of the $k+1$-th row of $\boldsymbol{R}$ by $\hat{\boldsymbol{w}}^{(k)}(i)$. We use one addition and one multiplication to obtain $H\left(\boldsymbol{\epsilon}_{k+1}^T\boldsymbol{R}\hat{\boldsymbol{w}}^{(k)}(i) - \boldsymbol{p}^T\boldsymbol{\epsilon}_{k+1}\right)$, and one more addition and one multiplication to add $(H^2/2)\boldsymbol{\epsilon}_{k+1}^T\boldsymbol{R}\boldsymbol{\epsilon}_{k+1} = (H^2/2){R_{k+1,k+1}}^3$ to the result, where we assume that $(H^2/2)$ is pre-computed, and $R_{k+1,k+1}$ is the element of $\boldsymbol{R}$ in the $(k+1)$-th column and $(k+1)$-th row.

To compute ΔJ_+ and ΔJ_- with low cost, we define a residue vector $\boldsymbol{r}$, that is updated at each step. The initial residue is given by

$$\boldsymbol{r}^{(0)}(0) = \boldsymbol{p} - \boldsymbol{R}\hat{\boldsymbol{w}}^{(0)}(0), \tag{4.7}$$

and we compute

$$\begin{aligned}\boldsymbol{r}^{(1)}(0) &= \boldsymbol{p} - \boldsymbol{R}\hat{\boldsymbol{w}}^{(1)}(0)\\ &= \begin{cases} \boldsymbol{p} - \boldsymbol{R}\left(\hat{\boldsymbol{w}}^{(0)}(0) + H\boldsymbol{\epsilon}_1\right) = \boldsymbol{r}^{(0)}(0) - H\boldsymbol{R}\boldsymbol{\epsilon}_1, & \text{if } \Delta J_+ < 0\\ \boldsymbol{p} - \boldsymbol{R}\left(\hat{\boldsymbol{w}}^{(0)}(0) - H\boldsymbol{\epsilon}_1\right) = \boldsymbol{r}^{(0)}(0) + H\boldsymbol{R}\boldsymbol{\epsilon}_1, & \text{if } \Delta J_- < 0\\ \boldsymbol{r}^{(0)}(0), & \text{otherwise.}\end{cases}\end{aligned} \tag{4.8}$$

Note that to obtain $\boldsymbol{r}^{(1)}(0)$, we still need N multiplications and N additions. However, if we use a fixed-point implementation and choose H to be a power of two, we can exploit hardware properties to use only N bit-shifts and N additions. In this case, we reduce the complexity, since multiplications are more costly than bit-shifts. But there is a better way to obtain the estimation, which avoids the direct evaluation of (4.5) and (4.6).

Recall equations (4.5) and (4.6). Using the residue definition, one can write

$$\Delta J_+ = -H\boldsymbol{\epsilon}_{k+1}^T\boldsymbol{r}^{(k)}(i) + \frac{1}{2}H^2\boldsymbol{\epsilon}_{k+1}^T\boldsymbol{R}\boldsymbol{\epsilon}_{k+1} = -Hr_{k+1}^{(k)}(i) + \frac{1}{2}H^2R_{k+1,k+1} \tag{4.9}$$

and

$$\Delta J_- = H\boldsymbol{\epsilon}_{k+1}^T\boldsymbol{r}^{(k)}(i) + \frac{1}{2}H^2\boldsymbol{\epsilon}_{k+1}^T\boldsymbol{R}\boldsymbol{\epsilon}_{k+1} = Hr_{k+1}^{(k)}(i) + \frac{1}{2}H^2R_{k+1,k+1}, \tag{4.10}$$

where $r_{k+1}^{(k)}(i)$ is the $(k+1)$-th entry of $\boldsymbol{r}^{(k)}(i)$.

Using (4.9) and recalling that $H > 0$ and that $\boldsymbol{R}$ is positive-definite, $\Delta J_+ < 0$ only if

$$r_{k+1}^{(k)}(i) > \frac{1}{2}HR_{k+1,k+1} \geq 0. \tag{4.11}$$

Similarly, $\Delta J_- < 0$ only if

$$-r_{k+1}^{(k)}(i) > \frac{1}{2}HR_{k+1,k+1}. \tag{4.12}$$

[3]To denote the $k+1$-th diagonal element of $\boldsymbol{R}$, we use $R_{k+1,k+1}$ instead of $r_{k+1,k+1}$ to avoid confusion with the entries of $\boldsymbol{r}$, which is the residue.

Note that we can define a unique condition which covers both (4.11) and (4.12), i.e.,

$$|r_{k+1}^{(k)}(i)| > \frac{1}{2} H R_{k+1,k+1}. \tag{4.13}$$

Using the condition (4.13), we update the estimate and the residue with

$$\hat{\boldsymbol{w}}^{(k+1)}(i) = \hat{\boldsymbol{w}}^{(k)}(i) + H\text{sign}(r_{k+1}^{(k)}(i))\boldsymbol{\epsilon}_{k+1} \tag{4.14}$$

and

$$\boldsymbol{r}^{(k+1)}(i) = \boldsymbol{r}^{(k)}(i) - H\text{sign}(r_{k+1}^{(k)}(i))\boldsymbol{R}\boldsymbol{\epsilon}_{k+1}, \tag{4.15}$$

where sign(α) is equal to 1 if $\alpha \geq 0$ and -1 otherwise.

Using equations (4.13), (4.14) and (4.15), the update requires only one comparison, $N+1$ additions and $N+1$ shifts. Choosing H as a power of two, further reduction of complexity can be achieved, since many multiplications are replaced by shifts and additions, less costly to compute. Since the algorithm cyclically updates the solution coefficients, it is namely the real-valued cyclic DCD, and it is presented in Table 12[4], where $\boldsymbol{R}^{(1:N,1)}$ denotes the first column of $\boldsymbol{R}$ and $\hat{w}_k$ is the k-th element of $\hat{\boldsymbol{w}}$.

The DCD performs a bitwise update of the $\hat{\boldsymbol{w}}$ entries, which we assume that are represented with M_b bits. The update is performed from the most to the least significant bits. When the condition (4.13) is not fulfilled, there is no update. In this case, we say that the iteration was unsuccessful and the steps 4 to 6 (see Table 12) are avoided. When the algorithm updates $\hat{\boldsymbol{w}}$, we have a successful iteration. The technique uses a maximum number of successful iterations – given by N_u – that determines when the algorithm stops.

The computational complexity of the real-valued cyclic-DCD algorithm is a random variable, which motivates the computation of the worst case complexity. In the worst case (which occurs when the *for* loops are executed until $m = M_b$ and $k = N$, and the condition in step 3 is met N_u times), step 1 updates α M_b times, using shifts. Step 2 can be disregarded, since it just attributes a value to a flag. Step 3 uses one comparison (expressed by the operator $>$) and one shift to compute $(\alpha/2)R_{k,k}$, assuming that both α and $R_{k,k}$ are represented as a power of 2. When both *for* loops are completely executed, the total cost of step 3 is $2NM_b$. In the worst scenario, steps 4, 5, 6 and 7 are computed N_u times (these steps are executed only N_u times, since the algorithm stops when $q = N_u$ in step 7). Step 4 computes only one addition per iteration, since sign(r_k)α is just a sign change. Step 5 uses N shifts to obtain the term $\alpha\boldsymbol{R}^{(1:N,1)}$ and N additions to update $\boldsymbol{r}$

[4]The number of multiplications is omitted since the algorithm only uses additions, comparisons and shifts.

Table 12: Real-valued cyclic-DCD algorithm

Step	Equation	Number of additions, shifts and comparisons
	Initialization: $\hat{\boldsymbol{w}} = \mathbf{0}_{N\times 1}$, $\boldsymbol{r} = \boldsymbol{p}$ $\alpha = H$, $q = 0$	
	for $m = 1, \ldots, M_b$	
1	$\alpha = \alpha/2$	1
2	flag $= 0$	0
	for $k = 1, \ldots, N$	
3	if $\lvert r_k\rvert > (\alpha/2)R_{k,k}$ then	2
4	$\hat{w}_k = \hat{w}_k + \text{sign}(r_k)\alpha$	1
5	$\boldsymbol{r} = \boldsymbol{r} - \text{sign}(r_k)\alpha\boldsymbol{R}^{(1:N,1)}$	$2N$
6	$q = q + 1$, flag $= 1$	1
7	if $q > N_u$, the algorithm stops	1
8	if flag $= 1$, then repeat step 2	1
	Total: (worst case)	$< (2N + 2)M_b + N_u(2N + 3)$

at each iteration. Step 6 only requires one addition to update q, while step 7 uses one comparison to verify if the algorithm stops. Step 8 uses one comparison, and in the worst case is repeated N times. The total computational cost for the worst case corresponds to $P_m = 0$ multiplications and $P_a = (2N + 2)M_b + N_u(2M + 3)$ additions, comparisons and shifts, which are presented together since all these operations require approximately the same of the hardware. This complexity is a pessimistic bound, since in general this upper bound is not reached.

A complex-valued extension of the cyclic-DCD was proposed in [70] (see Table 13). Different from the real-valued algorithm, the complex-valued cyclic-DCD has two values to be updated at every iteration, which correspond to the real and the imaginary parts of the k-th entry of $\hat{\boldsymbol{w}}$. The algorithm starts with the real part of an entry $\hat{w}_k$, and then proceeds to update $\mathcal{I}m\{\hat{w}_k\}$. To verify if $\mathcal{R}e\{\hat{w}_k\}$ must be updated, a condition based on the real part of the k-th entry of $\boldsymbol{r}$ is verified (see step 3 of Table 13). If it holds, then $\mathcal{R}e\{\hat{w}_k\}$ is updated, following the same approach used in the real-valued cyclic-DCD. If the criterion is not met for the real part of $\hat{w}_k$, then the algorithm proceeds with the update of the imaginary part. After updating the real and imaginary parts of $\hat{w}_k$, the technique proceeds to the next entry $\hat{w}_{k+1}$, and it continues with the update until some stopping criterion holds. Only a few modifications are required to obtain the code of the complex-valued technique, as presented in Table 13 (see the first line of step 3 and step 8). The conditions added to account for the imaginary parts of the solution increase the computational complexity of the technique. However, only additions,

comparison and shifts are added to the code, so that the complex-valued algorithm can also be implemented at low-cost.

Table 13: Complex-valued cyclic-DCD algorithm

Step	Equation	Number of additions, shifts and comparisons
	Initialization: $\hat{\boldsymbol{w}} = \mathbf{0}_{N\times 1}$, $\boldsymbol{r} = \boldsymbol{p}$ $\alpha = H$, $q = 0$, $s = 1$	
	for $m = 1, \ldots, M_b$	
1	$\alpha = \alpha/2$	1
2	flag $= 0$	0
	for $k = 1, \ldots, N$	
3	if $s = 1$ then $r_{tmp} = \mathcal{R}e\{r_k\}$, else $r_{tmp} = \mathcal{I}m\{r_k\}$ if $\lvert r_{tmp}\rvert > (\alpha/2)R_{k,k}$ then	3
4	$\hat{w}_k = \hat{w}_k + \text{sign}(r_{tmp})s\alpha$	1
5	$\boldsymbol{r} = \boldsymbol{r} - \text{sign}(r_{tmp})s\alpha\boldsymbol{R}^{(1:N,1)}$	$2N$
6	$q = q + 1$, flag $= 1$	1
7	if $q > N_u$, the algorithm stops	1
8	if $s = 1$, then $s = \boldsymbol{j}$, go to step 3; else $s = 1$	2
9	if flag $= 1$, then repeat step 2	1
	Total: (worst case)	$< M_b(8N + 2)$ $+N_u(2N + 3)$

The cyclic-DCD algorithm updates the entries $\hat{w}_k$ cyclically, from $k = 1, \ldots, N$. Since the number of successful updates is determined by N_u, which is generally chosen low ($N_u \ll N$), the first elements of $\hat{\boldsymbol{w}}$ are more likely to be updated. However, these terms are not necessarily the terms which are related to the main features of the residue, such that these update choices may not be the better ones. This fact motivated the development of the leading-element DCD algorithm [26], which modifies the criterion to choose the entry of $\hat{\boldsymbol{w}}$ which will be updated.

The complex DCD with a leading-element (presented in Table 14) first updates the entries of $\hat{\boldsymbol{w}}$ which make the residue higher. For this purpose, the criterion

$$[k, s] = \arg\max_{p=1,\ldots,N}\{|\mathcal{R}e\{r_p\}|, |\mathcal{I}m\{r_p\}|\} \tag{4.16}$$

is applied at each iteration to determine the index p of the real or imaginary element of $\boldsymbol{r}$ with the highest value (which is stored in variable k). Updating $\hat{w}_k$, we improve the estimation of $\hat{\boldsymbol{w}}$ by reducing the error in the direction that contributes the most to the residue. s is used to determine if the real ($s = 1$) or the imaginary ($s = \boldsymbol{j}$, where $\boldsymbol{j} = \sqrt{-1}$) part of the k-th entry of $\hat{\boldsymbol{w}}$ will be updated.

This version of DCD is advantageous when the number of successful iterations N_u is low, since it concentrates on reducing the main features of the residue. For larger values of N_u, the solution computed by both techniques is expected to have similar values for the entries. When $N_u \to \infty$, all entries of $\hat{\boldsymbol{w}}$ will be updated in both versions of the algorithms, and the same $\hat{\boldsymbol{w}}$ will be obtained.

Table 14: Complex-valued leading-element DCD algorithm

Step	Equation	Number of additions, shifts and comparisons
	Initialization: $\hat{\boldsymbol{w}} = \boldsymbol{0}_{N\times 1}$, $\boldsymbol{r} = \boldsymbol{p}$, $\alpha = H$, $q = 0$	
	for $m = 1, \ldots, N_u$	
1	$[k, s] = \arg\max_{p=1,\ldots,N}\{\lvert\mathcal{R}e\{r_p\}\rvert, \lvert\mathcal{I}m\{r_p\}\rvert\}$ go to step 4	$2N$
2	$q = q + 1$, $\alpha = \alpha/2$	2
3	if $q > M_b$, the algorithm stops	1
4	if $s = 1$ then $r_{tmp} = \mathcal{R}e\{r_k\}$, else $r_{tmp} = \mathcal{I}m\{r_k\}$ if $\lvert r_{tmp}\rvert \leq (\alpha/2)R_{k,k}$ then go to step 2	3
5	$\hat{w}_k = \hat{w}_k + \text{sign}(r_{tmp})s\alpha$	1
6	$\boldsymbol{r} = \boldsymbol{r} - \text{sign}(r_{tmp})s\alpha\boldsymbol{R}^{(1:N,1)}$	$2N$
	Total: (worst case)	$< 6M_b + (4N + 4)N_u$

The DCD was used in [25] to obtain a numerically stable and low-cost ($\boldsymbol{\mathcal{O}}(N)$) version of the RLS algorithm. In the next section, we adapt the technique of [25] to the RC-WL-RLS algorithm.

4.2 The DCD-WL-RLS algorithm

In this section, we modify the RC-WL-RLS to develop the DCD-WL-RLS algorithm with complexity linear on N. We start presenting a low-cost update of the estimated autocorrelation matrix, obtained when the entries of the input vector arise from a tap delay-line. Then, we adapt RC-WL-RLS to apply DCD.

Recall that the RC-WL-RLS algorithm uses a $2N \times 1$ real input vector, given by

$$\boldsymbol{x}_{\text{RC}}(n) = \begin{bmatrix} \boldsymbol{x}_{\text{R}}(n) \\ \boldsymbol{x}_{\text{I}}(n) \end{bmatrix}. \tag{4.17}$$

In the n-th iteration, RC-WL-RLS computes the solution for the normal equations

$$\boldsymbol{R}(n)\boldsymbol{w}(n) = \boldsymbol{p}(n), \tag{4.18}$$

assuming that the estimated autocorrelation matrix $\boldsymbol{R}(n)$ and the cross-correlation vector $\boldsymbol{p}(n)$ are updated as given by

$$\boldsymbol{R}(n) = \sum_{i=0}^{n} \nu^{n-i} \boldsymbol{x}_{\mathrm{RC}}(i) \boldsymbol{x}_{\mathrm{RC}}^{T}(i) = \nu \boldsymbol{R}(n-1) + \boldsymbol{x}_{\mathrm{RC}}(n) \boldsymbol{x}_{\mathrm{RC}}^{T}(n), \tag{4.19}$$

and

$$\boldsymbol{p}(n) = \sum_{i=0}^{n} \nu^{n-i} d(i) \boldsymbol{x}_{\mathrm{RC}}(i) = \nu \boldsymbol{p}(n-1) + d(n) \boldsymbol{x}_{\mathrm{RC}}(n), \tag{4.20}$$

where ν is the forgetting factor. $\boldsymbol{R}(n)$ is real and symmetric, with dimension $2N \times 2N$, and $\boldsymbol{p}(n)$ is a $2N \times 1$ vector. Note that in the general case, the estimation of $\boldsymbol{R}(n)$ requires the computation of $2N^2 - N$ entries at each iteration (the elements of the main diagonal and above it). However, when $\boldsymbol{x}_{\mathrm{R}}(n)$ and $\boldsymbol{x}_{\mathrm{I}}(n)$ have a known structure, this complexity can be reduced.

When both $\boldsymbol{x}_{\mathrm{R}}(n)$ and $\boldsymbol{x}_{\mathrm{I}}(n)$ come from tap-delay lines, i.e.,

$$\boldsymbol{x}_{\mathrm{R}}(n) = \begin{bmatrix} x_{\mathrm{R}}(n) & x_{\mathrm{R}}(n-1) & \cdots & x_{\mathrm{R}}(n-N+1) \end{bmatrix}^{T} \tag{4.21}$$

and

$$\boldsymbol{x}_{\mathrm{I}}(n) = \begin{bmatrix} x_{\mathrm{I}}(n) & x_{\mathrm{I}}(n-1) & \cdots & x_{\mathrm{I}}(n-N+1) \end{bmatrix}^{T}, \tag{4.22}$$

a low-cost update of $\boldsymbol{R}(n)$, linear on N, can be applied. To show this, rewrite $\boldsymbol{R}(n)$ to explicit the contribution of $\boldsymbol{x}_{\mathrm{R}}(n)$ and $\boldsymbol{x}_{\mathrm{I}}(n)$,

$$\boldsymbol{R}(n) = \begin{bmatrix} \boldsymbol{R}_{\mathrm{R}}(n) & \boldsymbol{R}_{\mathrm{RI}}(n) \\ \boldsymbol{R}_{\mathrm{RI}}^{T}(n) & \boldsymbol{R}_{\mathrm{I}}(n) \end{bmatrix},$$

where

$$\boldsymbol{R}_{\mathrm{R}}(n) = \sum_{i=0}^{n} \nu^{n-i} \boldsymbol{x}_{\mathrm{R}}(i) \boldsymbol{x}_{\mathrm{R}}^{T}(i) = \nu \boldsymbol{R}_{\mathrm{R}}(n-1) + \boldsymbol{x}_{\mathrm{R}}(n) \boldsymbol{x}_{\mathrm{R}}^{T}(n), \tag{4.23}$$

$$\boldsymbol{R}_{\mathrm{I}}(n) = \sum_{i=0}^{n} \nu^{n-i} \boldsymbol{x}_{\mathrm{I}}(i) \boldsymbol{x}_{\mathrm{I}}^{T}(i) = \nu \boldsymbol{R}_{\mathrm{I}}(n-1) + \boldsymbol{x}_{\mathrm{I}}(n) \boldsymbol{x}_{\mathrm{I}}^{T}(n), \tag{4.24}$$

and

$$\boldsymbol{R}_{\mathrm{RI}}(n) = \sum_{i=0}^{n} \nu^{n-i} \boldsymbol{x}_{\mathrm{R}}(i) \boldsymbol{x}_{\mathrm{I}}^{T}(i) = \nu \boldsymbol{R}_{\mathrm{RI}}(n-1) + \boldsymbol{x}_{\mathrm{R}}(n) \boldsymbol{x}_{\mathrm{I}}^{T}(n), \tag{4.25}$$

and note that $\boldsymbol{R}_{\mathrm{R}}(n)$ and $\boldsymbol{R}_{\mathrm{I}}(n)$ are symmetric matrices, but $\boldsymbol{R}_{\mathrm{RI}}(n)$ is not symmetric in general. $\boldsymbol{R}_{\mathrm{R}}(n)$ and $\boldsymbol{R}_{\mathrm{I}}(n)$ can be evaluated using a low-cost update, as employed in [25]:

$$\boldsymbol{R}_{\mathrm{R}}(n) = \begin{bmatrix} R_{\mathrm{R}}(n) & \boldsymbol{\rho}_{\mathrm{R}}^{T}(n) \\ \boldsymbol{\rho}_{\mathrm{R}}(n) & \boldsymbol{R}_{\mathrm{R}}^{(2:N,2:N)}(n) \end{bmatrix}, \tag{4.26}$$

where the notation $\boldsymbol{R}_{\mathrm{R}}^{(2:N,2:N)}(n)$ stands for a matrix with the elements of $\boldsymbol{R}_{\mathrm{R}}(n)$ from row 2 to N and from column 2 to N. $R_{\mathrm{R}}(n)$ is a real number and $\boldsymbol{\rho}_{\mathrm{R}}(n)$ is an $(N-1)\times 1$ vector. Recalling that $\boldsymbol{x}_{\mathrm{R}}^{(1:N-1)}(n-1) = \boldsymbol{x}_{\mathrm{R}}^{(2:N)}(n)$[5], the elements of (4.26) are given by

$$R_{\mathrm{R}}(n) = \sum_{i=0}^{n} \nu^{n-i} x_{\mathrm{R}}^2(i),$$

$$\boldsymbol{\rho}_{\mathrm{R}}(n) = \sum_{i=0}^{n} \nu^{n-i} x_{\mathrm{R}}(i) \boldsymbol{x}_{\mathrm{R}}^{(1:N-1)}(i-1), \tag{4.27}$$

and

$$\begin{aligned} \boldsymbol{R}_{\mathrm{R}}^{(2:N,2:N)}(n) &= \textstyle\sum_{i=0}^{n} \nu^{n-i} \boldsymbol{x}_{\mathrm{R}}^{(2:N)}(i) \boldsymbol{x}_{\mathrm{R}}^{(2:N)^T}(i) \\ &= \textstyle\sum_{i=0}^{n} \nu^{n-i} \boldsymbol{x}_{\mathrm{R}}^{(1:N-1)}(i-1) \boldsymbol{x}_{\mathrm{R}}^{(1:N-1)^T}(i-1) \\ &= \textstyle\sum_{i=0}^{n-1} \nu^{n-i-1} \boldsymbol{x}_{\mathrm{R}}^{(1:N-1)}(i) \boldsymbol{x}_{\mathrm{R}}^{(1:N-1)^T}(i) \\ &= \boldsymbol{R}_{\mathrm{R}}^{(1:N-1,1:N-1)}(n-1). \end{aligned} \tag{4.28}$$

From eq. (4.26), note that only the first column of $\boldsymbol{R}_{\mathrm{R}}(n)$ needs to be updated, since $\boldsymbol{R}_{\mathrm{R}}^{(1:N-1,1:N-1)}(n-1)$ is available from the last iteration and $\boldsymbol{R}_{\mathrm{R}}(n)$ is symmetric. In order to obtain the first row of $\boldsymbol{R}_{\mathrm{R}}(n)$ we only need to copy the values of the first column. Thus, we calculate only the first column of $\boldsymbol{R}_{\mathrm{R}}(n)$ – that is $\boldsymbol{R}_{\mathrm{R}}^{(1:N,1)}(n)$ – to update eq. (4.26), i.e.,

$$\boldsymbol{R}_{\mathrm{R}}^{(1:N,1)}(n) = \nu \boldsymbol{R}_{\mathrm{R}}^{(1:N,1)}(n-1) + x_{\mathrm{R}}(n) \boldsymbol{x}_{\mathrm{R}}(n).$$

Similarly, we update the first column of eq. (4.24) using

$$\boldsymbol{R}_{\mathrm{I}}^{(1:N,1)}(n) = \nu \boldsymbol{R}_{\mathrm{I}}^{(1:N,1)}(n-1) + x_{\mathrm{I}}(n) \boldsymbol{x}_{\mathrm{I}}(n).$$

To update $\boldsymbol{R}_{\mathrm{RI}}(n)$, write (4.25) as the matrix

$$\boldsymbol{R}_{\mathrm{RI}}(n) = \begin{bmatrix} R_{\mathrm{RI}}(n) & \boldsymbol{\rho}_{\mathrm{RI}}^T(n) \\ \boldsymbol{\rho}_{\mathrm{IR}}(n) & \boldsymbol{R}_{\mathrm{RI}}^{(2:N,2:N)}(n) \end{bmatrix}, \tag{4.29}$$

where

$$R_{\mathrm{RI}}(n) = \sum_{i=0}^{n} \nu^{n-i} x_{\mathrm{R}}(i) x_{\mathrm{I}}(i),$$

$$\boldsymbol{\rho}_{\mathrm{RI}}(n) = \sum_{i=0}^{n} \nu^{n-i} x_{\mathrm{R}}(i) \boldsymbol{x}_{\mathrm{I}}(i-1), \tag{4.30}$$

$$\boldsymbol{\rho}_{\mathrm{IR}}(n) = \sum_{i=0}^{n} \nu^{n-i} x_{\mathrm{I}}(i) \boldsymbol{x}_{\mathrm{R}}(i-1) \tag{4.31}$$

[5]Note that it also holds $\boldsymbol{x}_{\mathrm{I}}^{(1:N-1)}(n-1) = \boldsymbol{x}_{\mathrm{I}}^{(2:N)}(n)$.

and note that $\boldsymbol{R}_{\mathrm{RI}}^{(2:N,2:N)}(n)$ can be related to $\boldsymbol{R}_{\mathrm{RI}}(n-1)$ as follows

$$\begin{aligned}\boldsymbol{R}_{\mathrm{RI}}^{(2:N,2:N)}(n) &= \sum_{i=0}^{n} \nu^{n-i} \boldsymbol{x}_{\mathrm{R}}^{(2:N)}(i) \boldsymbol{x}_{\mathrm{I}}^{(2:N)^T}(i) \\ &= \sum_{i=0}^{n} \nu^{n-i} \boldsymbol{x}_{\mathrm{R}}^{(1:N-1)}(i-1) \boldsymbol{x}_{\mathrm{I}}^{(1:N-1)^T}(i-1) \\ &= \sum_{i=0}^{n-1} \nu^{n-i-1} \boldsymbol{x}_{\mathrm{R}}^{(1:N-1)}(i) \boldsymbol{x}_{\mathrm{I}}^{(1:N-1)^T}(i) \\ &= \boldsymbol{R}_{\mathrm{RI}}^{(1:N-1,1:N-1)}(n-1).\end{aligned} \tag{4.32}$$

Therefore, $\boldsymbol{R}_{\mathrm{RI}}^{(2:N,2:N)}(n)$ is a block matrix obtained from iteration $n-1$, which does not need to be updated. Equations (4.30) and (4.31) show that, in general $\boldsymbol{\rho}_{\mathrm{RI}} \neq \boldsymbol{\rho}_{\mathrm{IR}}$, which means that they need to be calculated separately. To obtain the first column of $\boldsymbol{R}_{\mathrm{RI}}(n)$, define the column vector

$$\begin{aligned}\boldsymbol{R}_{\mathrm{RI}}^{(1:N,1)}(n) &= \begin{bmatrix} R_{\mathrm{RI}}(n) & \boldsymbol{\rho}_{\mathrm{IR}}^T(n) \end{bmatrix}^T \\ &= \sum_{i=0}^{n} \nu^{n-i} x_{\mathrm{I}}(i) \boldsymbol{x}_{\mathrm{R}}(i) \\ &= \sum_{i=0}^{n-1} \nu^{n-i} x_{\mathrm{I}}(i) \boldsymbol{x}_{\mathrm{R}}(i) + x_{\mathrm{I}}(n) \boldsymbol{x}_{\mathrm{R}}(n) \\ &= \nu \boldsymbol{R}_{\mathrm{RI}}^{(1:N,1)}(n-1) + x_{\mathrm{I}}(n) \boldsymbol{x}_{\mathrm{R}}(n),\end{aligned} \tag{4.33}$$

which is recursively updated. The first row of $\boldsymbol{R}_{\mathrm{RI}}(n)$ is updated in a similar manner, i.e.,

$$\boldsymbol{R}_{\mathrm{RI}}^{(1,2:N)}(n) = \nu \boldsymbol{R}_{\mathrm{RI}}^{(1,2:N)}(n-1) + x_{\mathrm{R}}(n) \boldsymbol{x}_{\mathrm{I}}^{(2:N)^T}(n). \tag{4.34}$$

In eq. (4.34) we do not calculate the first element of the row, since it is already computed in eq. (4.33). Using this approach, the matrix $\boldsymbol{R}(n)$ is computed with the update of three $N \times N$ block-matrices. Recall that for each block-matrix, we only need to update the first row – in the case of $\boldsymbol{R}_{\mathrm{R}}(n)$ and $\boldsymbol{R}_{\mathrm{I}}(n)$– or the first row and the first column (in the case of $\boldsymbol{R}_{\mathrm{RI}}(n)$). We use this low-cost form to update matrix $\boldsymbol{R}(n)$ in the DCD-WL-RLS algorithm (see Table 15).

Assume that the matrix $\boldsymbol{R}(n)$ is updated with the procedure shown before. To obtain the DCD-WL-RLS, recall eq. (4.18), but in the instant $n-1$, i.e.,

$$\boldsymbol{R}(n-1)\boldsymbol{w}(n-1) = \boldsymbol{p}(n-1). \tag{4.35}$$

Assume that an approximate solution $\tilde{\boldsymbol{w}}(n-1)$ is available. We define the residue of the normal equations, after obtaining an approximate solution $\tilde{\boldsymbol{w}}(n-1)$, as the vector

$$\boldsymbol{\zeta}(n-1) = \boldsymbol{p}(n-1) - \boldsymbol{R}(n-1)\tilde{\boldsymbol{w}}(n-1). \tag{4.36}$$

Moreover, define

$$\Delta\boldsymbol{R}(n) = \boldsymbol{R}(n) - \boldsymbol{R}(n-1), \tag{4.37}$$

$$\Delta \boldsymbol{p}(n) = \boldsymbol{p}(n) - \boldsymbol{p}(n-1), \tag{4.38}$$

and

$$\Delta \boldsymbol{w}(n) = \boldsymbol{w}(n) - \tilde{\boldsymbol{w}}(n-1). \tag{4.39}$$

Substituting eqs. (4.37), (4.38) and (4.39) in (4.35), after some manipulation, we obtain

$$\boldsymbol{R}(n)\Delta \boldsymbol{w}(n) = \boldsymbol{\beta}_0(n), \tag{4.40}$$

where

$$\boldsymbol{\beta}_0(n) = \boldsymbol{\zeta}(n-1) + \Delta \boldsymbol{p}(n) - \Delta \boldsymbol{R}(n)\tilde{\boldsymbol{w}}(n-1). \tag{4.41}$$

The DCD is used to solve equation (4.40) instead of (4.18), which is a way of taking advantage of the previous solution $\tilde{\boldsymbol{w}}(n-1)$ to reduce the complexity of the algorithm. This is due to the iterative structure of DCD – solving (4.40) is equivalent to starting DCD with $\tilde{\boldsymbol{w}}(n-1)$ as initial condition, reducing the number of DCD iterations (and the complexity) necessary at each time instant. We apply the DCD algorithm to iteratively solve (4.40) and obtain the approximate solution

$$\tilde{\boldsymbol{w}}(n) = \tilde{\boldsymbol{w}}(n-1) + \Delta \tilde{\boldsymbol{w}}(n). \tag{4.42}$$

Equation (4.36) is also updated in terms of $\boldsymbol{\beta}_0(n)$ and $\Delta \tilde{\boldsymbol{w}}(n)$, using eqs. (4.41) and (4.42), i.e.,

$$\boldsymbol{\zeta}(n) = \boldsymbol{\beta}_0(n) - \boldsymbol{R}(n)\Delta \tilde{\boldsymbol{w}}(n).$$

Recall (4.37) and (4.38). Using the second identity of (4.19) and (4.20) in eqs. (4.37) and (4.38), respectively, we can express

$$\Delta \boldsymbol{R}(n) = (1-\nu)\boldsymbol{R}(n-1) + \boldsymbol{x}_{\mathrm{RC}}(n)\boldsymbol{x}_{\mathrm{RC}}^T(n) \tag{4.43}$$

and

$$\Delta \boldsymbol{p}(n) = (1-\nu)\boldsymbol{\beta}_0(n) + d(n)\boldsymbol{x}_{\mathrm{RC}}(n). \tag{4.44}$$

Define the filter estimative

$$y(n) = \boldsymbol{x}_{\mathrm{RC}}^T(n)\tilde{\boldsymbol{w}}(n-1).$$

Using (4.43) and (4.44) in eq. (4.36),

$$\Delta \boldsymbol{R}(n)\tilde{\boldsymbol{w}}(n-1) = (\nu-1)\left[\boldsymbol{p}(n-1) - \boldsymbol{\zeta}(n-1)\right] + \boldsymbol{x}_{\mathrm{RC}}(n)y(n), \tag{4.45}$$

which can be used in (4.41) to obtain

$$\boldsymbol{\beta}_0(n) = \nu\boldsymbol{\zeta}(n-1) + e(n)\boldsymbol{x}_{\mathrm{RC}}(n),$$

where $e(n) = d(n) - y(n)$ corresponds to the *a priori* error. The DCD-WL-RLS algorithm is summarized in Table 15, where we also provide the initialization of the algorithm. The DCD algorithm is used to solve step 9 of Table 15. Note that $\zeta(n)$ corresponds to the residue $\boldsymbol{r}$ computed by the DCD method, so that in step 9 we obtain both $\Delta\tilde{\boldsymbol{w}}$ and $\zeta(n)$.

Table 15: DCD-WL-RLS algorithm

Step	Equation
	Initialization: $\Delta\hat{\boldsymbol{w}}(0) = \boldsymbol{0}_{2N\times 1}$, $\boldsymbol{\zeta}(0) = \boldsymbol{0}_{2N\times 1}$, $\boldsymbol{R}_{\text{R}} = \delta\boldsymbol{I}_N$, $\boldsymbol{R}_{\text{I}} = \delta\boldsymbol{I}_N$, $\boldsymbol{R}_{\text{RI}} = \boldsymbol{0}_N$
	for $n = 1, 2, \ldots$
1	$\boldsymbol{R}_{\text{R}}^{(1,1:N)}(n) = \nu\boldsymbol{R}_{\text{R}}^{(1,1:N)}(n-1) + x_{\text{R}}(n)\boldsymbol{x}_{\text{R}}(n)$
2	$\boldsymbol{R}_{\text{I}}^{(1,1:N)}(n) = \nu\boldsymbol{R}_{\text{I}}^{(1,1:N)}(n-1) + x_{\text{I}}(n)\boldsymbol{x}_{\text{I}}(n)$
3	$\boldsymbol{R}_{\text{RI}}^{(1,1:N)}(n) = \nu\boldsymbol{R}_{\text{RI}}^{(1,1:N)}(n-1) + x_{\text{I}}(n)\boldsymbol{x}_{\text{R}}(n)$
4	$\boldsymbol{R}_{\text{RI}}^{(2:N,1)}(n) = \nu\boldsymbol{R}_{\text{RI}}^{(2:N,1)}(n-1) + x_{\text{R}}(n)\boldsymbol{x}_{\text{I}}^{(2:N)^T}(n)$
5	$\boldsymbol{R}_{\text{R}}(n)$, $\boldsymbol{R}_{\text{I}}(n)$, $\boldsymbol{R}_{\text{RI}}(n) \Rightarrow \boldsymbol{R}(n)$
6	$y(n) = \boldsymbol{x}_{\text{RC}}^T(n)\tilde{\boldsymbol{w}}(n-1)$
7	$e(n) = d(n) - y(n)$
8	$\boldsymbol{\beta}_0(n) = \nu\boldsymbol{\zeta}(n-1) + e(n)\boldsymbol{x}_{\text{RC}}(n)$
9	$\boldsymbol{R}(n)\Delta\tilde{\boldsymbol{w}}(n) = \boldsymbol{\beta}_0(n) \Rightarrow \Delta\tilde{\boldsymbol{w}}(n)$, $\boldsymbol{\zeta}(n)$ (solved using DCD)
10	$\tilde{\boldsymbol{w}}(n) = \tilde{\boldsymbol{w}}(n-1) + \Delta\tilde{\boldsymbol{w}}(n)$
	end

Table 16 compares the complexity of the developed algorithm with the SL, WL and RC-WL-RLS. There are two rows referring to the DCD-WL-RLS. The first presents the complexity considering that ν can be any real number such that $0 < \nu < 1$. The second assumes that $\nu = 1 - 2^{-m}$, $m \in \mathbb{N}$, which allows the substitution of multiplications by bit-shifts and sums in steps 1, 2, 3, 4 and 8 in Table 15.

Table 16: Computational complexity of the SL-RLS, WL-RLS, RC-WL-RLS and DCD-WL-RLS per iteration

Algorithm	Additions and shifts	$\times$	$\div$
SL-RLS	$6N^2 + 14N - 1$	$7N^2 + 21N + 1$	1
WL-RLS	$24N^2 + 28N - 1$	$28N^2 + 42N + 1$	1
RC-WL-RLS	$6N^2 + 11N$	$8N^2 + 14N + 1$	1
DCD-WL-RLS	$16N + P_a - 1$	$20N + P_m - 2$	-
DCD-WL-RLS ($\nu = 1 - 2^{-m}$)	$32N + P_a - 3$	$12N + P_m - 1$	-

Note that in Table 16, $P_m = 0$ and $P_a < M_b(8N + 2) + N_u(2N + 3)$, if the complex-valued cyclic DCD is applied, and $P_m = 0$ and $P_a < 6M_b + N_u(4N + 4)$ if complex-valued leading-element DCD is used.

4.3 Simulations

In this section, we compare DCD-WL-RLS using the DCD with a leading element with the RC-WL-RLS and the SL-RLS algorithms. For this purpose, we consider a system identification problem based on the approach of [38], and we assume that the input signal is given by

$$\boldsymbol{x}(n) = \sqrt{1-\tau^2}\boldsymbol{x}_{\mathrm{R}}(n) + \boldsymbol{j}\tau\boldsymbol{x}_{\mathrm{I}}(n), \tag{4.46}$$

where $\boldsymbol{x}_{\mathrm{R}}(n)$ and $\boldsymbol{x}_{\mathrm{I}}(n)$ are two real-valued uncorrelated complex-Gaussian processes, with zero-mean and variance $\sigma_{\mathrm{R}}^2 = \sigma_{\mathrm{I}}^2 = 1$. τ is used to define a non-circular input: when $\tau = 1/\sqrt{2}$, $\boldsymbol{x}(n)$ is second-order circular. Otherwise, it is non-circular.

To show the DCD-WL-RLS algorithm performance, we consider three scenarios:

1. In the first scenario, the system to be identified has 30 coefficients. The parameters are given by

   $$w_{\mathrm{opt},1,i} = \beta_1[(1+\cos(2\pi(i-3)/30) - \boldsymbol{j}(1+\cos(2\pi(i-3)/60))], \text{ for } i = 1,2,\ldots,30 \tag{4.47}$$

   where $\beta_1 = 1/||\boldsymbol{w}_{\mathrm{opt},1}||_2$ is used to normalize the $w_{\mathrm{opt},i}$. For this case, we assume that the desired signal $d(n)$ is given by

   $$d(n) = \mathcal{R}e\{\boldsymbol{w}_{\mathrm{opt},1}^H\boldsymbol{x}(n)\} + \eta(n), \tag{4.48}$$

   where $\eta(n)$ is zero-mean Gaussian noise, i.i.d, and the SNR=40dB. $\tau = 1/\sqrt{2}$, such that WL techniques are expected to achieve better MSE performance when compared to SL methods, as presented in Section 2.1.2.4.

2. In the second scenario, we assume that $\tau = 0.3$ and that

   $$d(n) = \boldsymbol{w}_{\mathrm{opt},1}^H\boldsymbol{x}(n) + \boldsymbol{w}_{\mathrm{opt},2}^H\boldsymbol{x}^*(n) + \eta(n), \tag{4.49}$$

   where

   $$w_{\mathrm{opt},2,i} = \beta_2[(1+\cos(2\pi(i-7)/30) - \boldsymbol{j}(1+\cos(2\pi(i-7)/60))], \text{ for } i = 1,2,\ldots,30 \tag{4.50}$$

   and $\beta_2 = 0.4/||\boldsymbol{w}_{\mathrm{opt},2}||_2$. Since the contribution of the real and the imaginary parts of $\boldsymbol{x}(n)$ are weighted differently, $\boldsymbol{x}(n)$ is non-circular. Recalling that $\boldsymbol{x}_{\mathrm{R}}(n)$ and $\boldsymbol{x}_{\mathrm{I}}(n)$ are uncorrelated Gaussian noise, one can verify that the autocorrelation and pseudo-correlation matrices are given by $\boldsymbol{C} = \boldsymbol{I}_{60}$ and $\boldsymbol{P} = (\sqrt{1-0.3^2} - 0.3)\boldsymbol{I}_{60}$.

Additionally, using $\boldsymbol{C}$, $\boldsymbol{P}$ and the values of $\boldsymbol{w}_{\text{opt},1}$ and $\boldsymbol{w}_{\text{opt},2}$, one can verify that $\boldsymbol{q}^* \neq \boldsymbol{0}$, such that the WL estimation is required to fully extract second-order data.

3. In the last scenario, we evaluate the tracking performance of the DCD-WL-RLS algorithm, when the set of optimum coefficients change. For this purpose, we use the same scenario of the second group of simulations, but we assume that $d(n)$ is given by (4.49) only in the first half of the iterations. After that, the coefficients change, and $d(n)$ is given by

$$d(n) = \boldsymbol{w}_{\text{opt},1}^H \boldsymbol{x}(n) + \boldsymbol{w}_{\text{opt},3}^H \boldsymbol{x}^*(n) + \eta(n), \tag{4.51}$$

where

$$w_{\text{opt},3,i} = \beta_3[(1 + \cos(2\pi(i-7)/30) - \boldsymbol{j}(1 + \cos(2\pi(i-7)/60))], \text{ for } i = 1, 2, \ldots, 30$$

and $\beta_3 = 0.6/||\boldsymbol{w}_{\text{opt},3}||_2$.

For all scenarios, we define the RC-WL-RLS forgetting factor $\nu_{\text{RC}} = 1 - 2^{-6}$ and we initialize $\boldsymbol{\Phi}_{\text{RC}}(0) = 5 \cdot 10^{-3}\boldsymbol{I}_{60}$. $\nu_{\text{DCD-RLS}}$ is chosen equal to ν_{RC}, and $H = 1$. The SL-RLS uses the same forgetting factor used by the other techniques, but it is initialized with $\boldsymbol{\Phi}_{\text{SL}}(0) = 1 \cdot 10^{-2}\boldsymbol{I}_{30}$. We compare the mean-square error (MSE) of the techniques in two situations:

1. Fixing the value of $N_u = 8$ in the DCD-WL-RLS and varying the number of bits M_b as 2, 4, 8 or 16;

2. Setting $M_b = 16$ bits and choosing N_u to be 1, 2, 4 or 8.

The curves are obtained with mean of 300 realizations. Figures 9 and 10 show the results for the first scenario, while figures 11 and 12 present the curves for the second case. The tracking performance is presented in figures 13 and 14

From figures 9 and 11, we note that for a fixed N_u, the precision of the DCD-WL-RLS algorithm increases as the number of bits M_b increases, when compared to the RC-WL-RLS implemented with Matlab precision. With this information, we can choose M_b to guarantee an specific MSE – of course bounded by the SNR – and use the algorithm with the minimum number of bits necessary for a given implementation. We also verify that the SL-RLS is outperformed by the RC-WL-RLS and by the DCD-WL-RLS algorithms, when the number of bits is 8 and 16. When M_b is low, the MSE is limited by the number

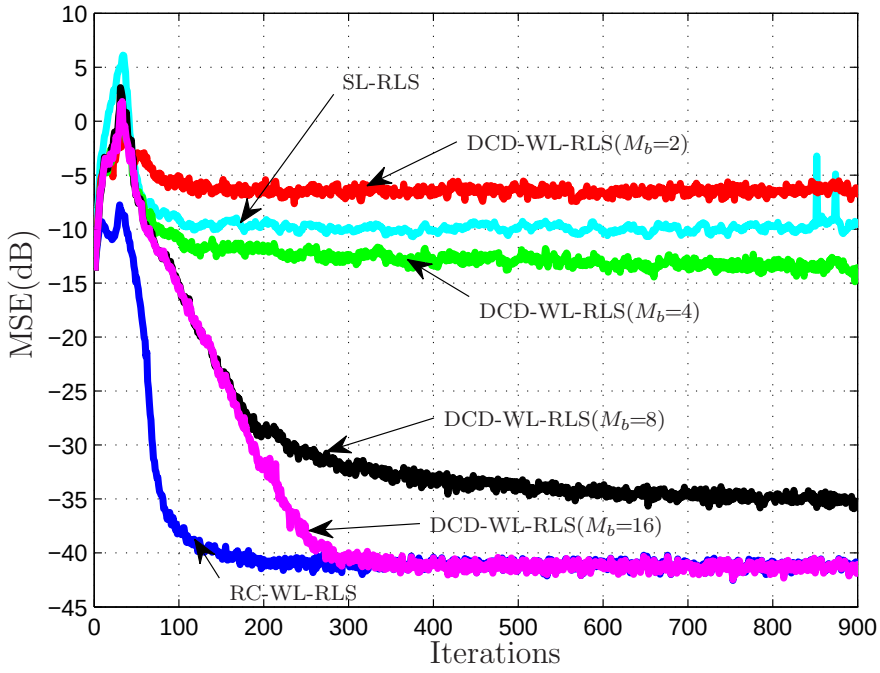

Figure 9: MSE comparison between the RC-WL-RLS, DCD-WL-RLS and SL-RLS algorithms using $N_u = 8$ and $M_b = 2$, 4, 8, 16, for the first scenario. Mean of 300 realizations.

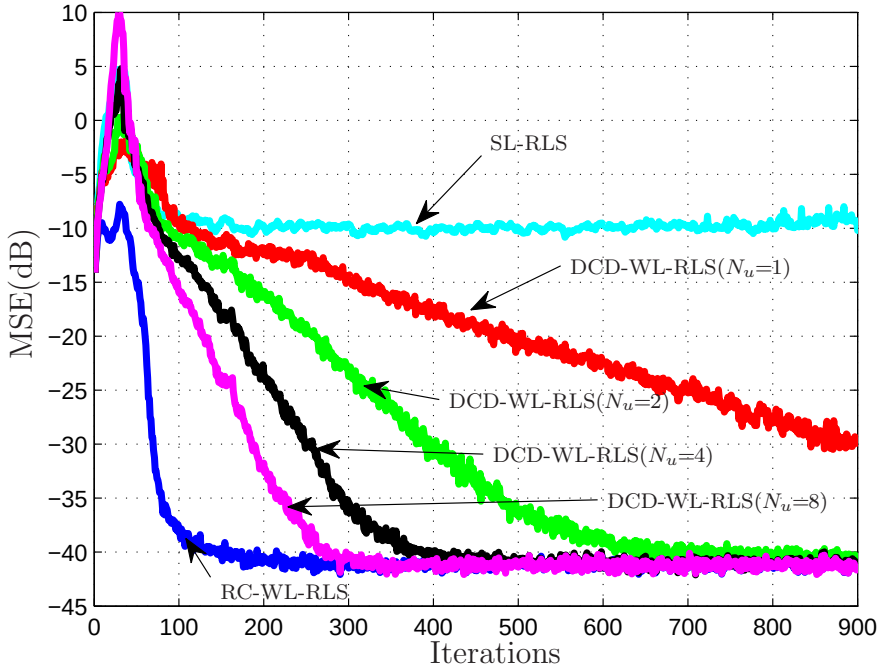

Figure 10: MSE comparison between the RC-WL-RLS, DCD-WL-RLS and SL-RLS algorithms using $M_b = 16$ and $N_u = 1$, 2, 4, 8, for the first scenario. Mean of 300 realizations.

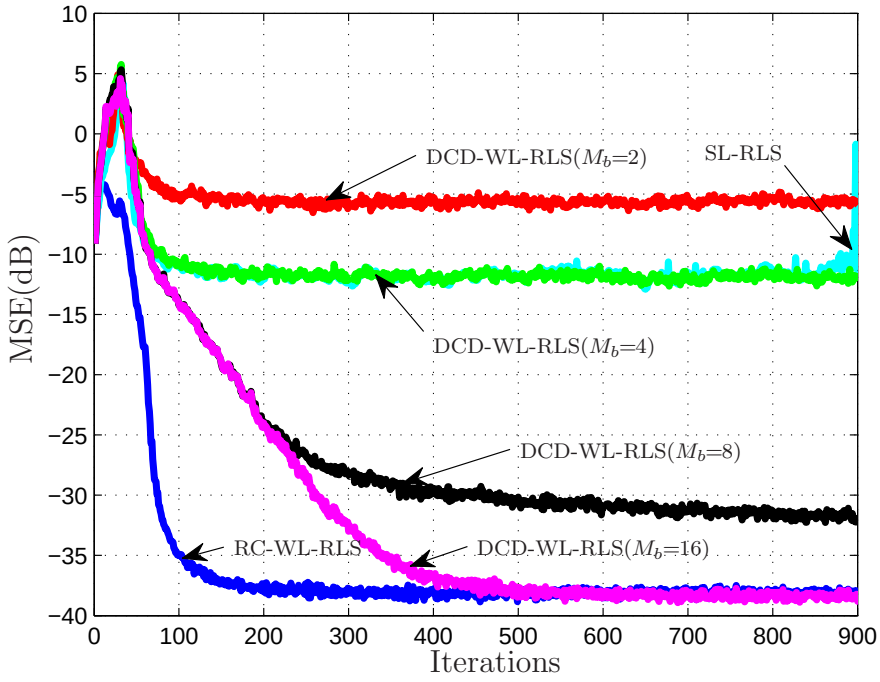

Figure 11: MSE comparison between the RC-WL-RLS, DCD-WL-RLS and SL-RLS algorithms using $N_u = 8$ and $M_b = 2,\ 4,\ 8,\ 16$, for the second scenario. Mean of 300 realizations.

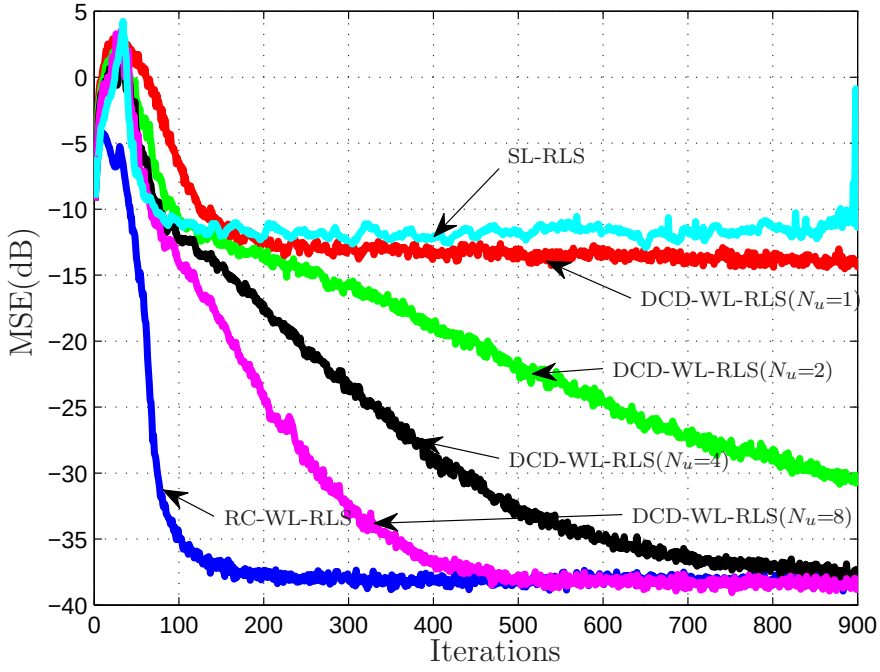

Figure 12: MSE comparison between the RC-WL-RLS, DCD-WL-RLS and SL-RLS algorithms using $M_b = 16$ and $N_u = 1,\ 2,\ 4,\ 8$, for the second scenario. Mean of 300 realizations.

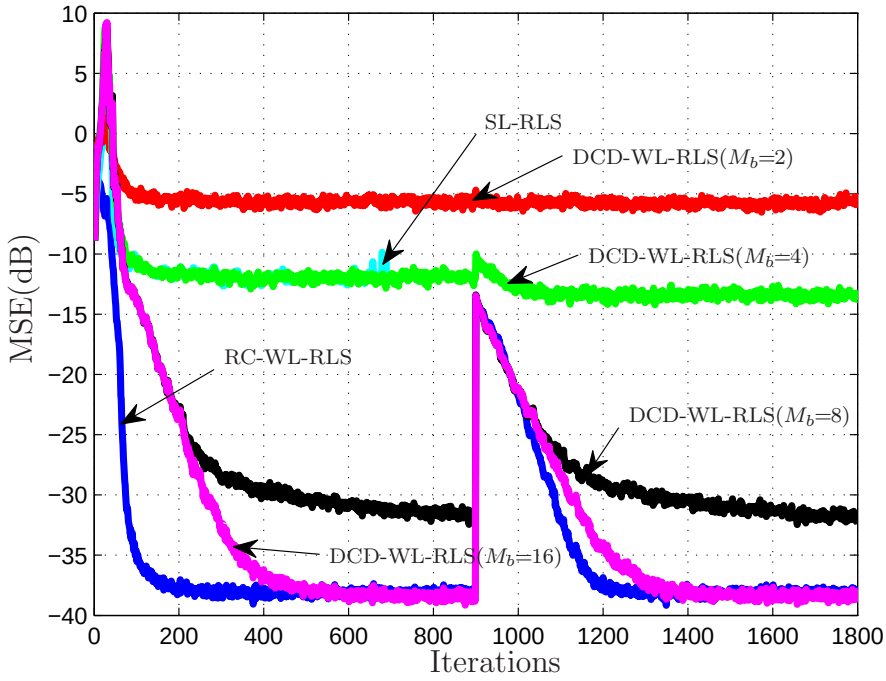

Figure 13: MSE comparison between the RC-WL-RLS, DCD-WL-RLS and SL-RLS algorithms, when the coefficients change after 900 iterations. $N_u = 8$ and $M_b = 2$, 4, 8, 16. Mean of 300 realizations.

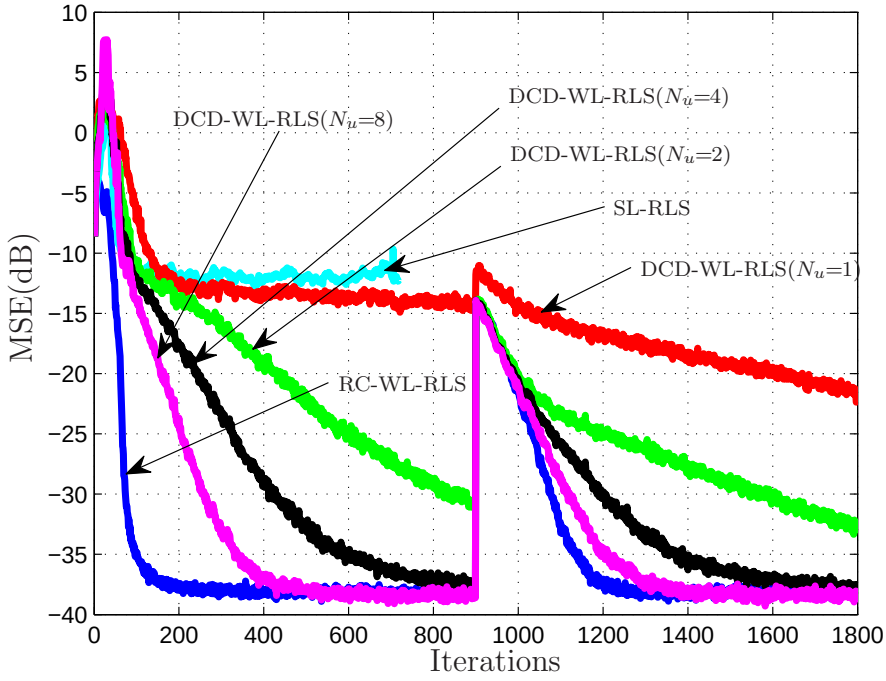

Figure 14: MSE comparison between the RC-WL-RLS, DCD-WL-RLS and SL-RLS algorithms, when the coefficients change after 900 iterations. $M_b = 16$ and $N_u = 1$, 2, 4, 8. Mean of 300 realizations.

of bits used by DCD, and the performance is poor. The SL-RLS algorithm diverges after 870 iteration, in both simulations.

In Figures 10 and 12, we see that, for a fixed number of bits, when N_u increases, the convergence of the DCD-based techniques is accelerated. We note that the convergence for $N_u = 1$ and $N_u = 2$ is slower than that of the DCD-WL-RLS algorithms, and the steady-state performance with these first two values of N_u is not presented in the figures. In general, the DCD-WL-RLS achieves the same steady-state MSE performance of the RC-WL-RLS algorithm. However, the convergence is slower than that of the RC-WL-RLS algorithm. When we apply the DCD to solve the normal equations, we reduce the complexity to $\mathcal{O}(N)$, but the trade-off for the low-cost is the additional number of iterations to achieve the steady-state MSE. Since we control the additional number of iterations by choosing a convenient N_u, this approach is very attractive for low-cost implementations. As expected, the SL-RLS is outperformed by the WL techniques in both simulations.

Figures 13 and 14 show the tracking performance of the new method when the set of optimum coefficients changes. Note that when $N_u = 8$ and M_b varies (see Fig. 13), using low values of M_b (2 and 4), the DCD-WL-RLS technique is not able to achieve the same tracking performance of the RC-WL-RLS algorithm, since the MSE performance is affected by the low number of bits. When $M_b = 8$ or 16, the DCD-WL-RLS has a tracking performance close to that obtained with RC-WL-RLS. When M_b has a fixed value and N_u varies (see Fig. 14), we notice that the tracking performance is improved when the value of N_u increases. In both simulations the SL-RLS algorithm diverges.

4.4 Conclusions

In this chapter, we presented the DCD-WL-RLS algorithm, a low-cost and numerically stable version of WL-RLS, since it uses DCD to solve the normal equation, avoiding the propagation of the inverse autocorrelation matrix. The DCD-WL-RLS is applicable for problems where the entries of the real and imaginary parts of the input are tap-delay lines, such that the autocorrelation matrix can be updated in a low-cost manner. Combining the DCD and the cheap update of the autocorrelation matrix, we obtained an algorithm with cost linear on N.

The first two scenarios considered in our simulations showed that the convergence of the proposed algorithm is affected by the values of N_u and M_b. When N_u is a large number,

the convergence is accelerated but the computational complexity is also increased. When M_b is low, the MSE is limited by the number of bits used in the representation, and the estimation error increases. The tracking performance simulations showed that the DCD-WL-RLS is able to track the non-linear perturbation caused by a change in $\boldsymbol{w}_{\text{opt}}$, even for low values of N_u. There is a relation between the value of N_u and the convergence rate after the non-linearity: the technique converges slower when N_u is low. As expected, we verified again that M_b limits the steady-state MSE, when M_b is low. From our simulation results, we conclude that a proper design of N_u and M_b (in our examples, $N_u = 8$ and $M_b = 16$ bits) is required to keep the cost low and the MSE performance close to that obtained with the RC-WL-RLS.

5 LOW-COMPLEX ADAPTIVE BEAMFORMING USING THE HOMOTOPY ALGORITHM

Adaptive beamforming techniques are used in sensor arrays to enhance the reception of a signal of interest and suppress interference [7]. They implement techniques such as the minimum variance distortionless response (MVDR) beamformer [7] using data collected from sensors, since the second-order statistics required to compute the MVDR beamformer, in general, are not available.

Although many fields apply adaptive beamforming techniques, such as radar, sonar and wireless communications [7], it is challenging to implement traditional approaches in large arrays. Techniques such as the LMS, the conjugate gradient (CG), and the RLS algorithms [7,24,45] have their convergence and tracking performances affected by the size and/or the eigenvalue spread of the input correlation matrix [45]. This performance can also degrade due to mismatch and modeling errors. Therefore, beamformers with many parameters may require many snapshots to converge, which can be incompatible with the requirements of some applications (for instance, space-time adaptive processing for airborne radar [71–74], where the amount of data involved requires a high computational cost to implement the beamformer).

As an alternative to traditional methods, robust adaptive beamformers were proposed to reduce the performance degradation caused by steering vector uncertainties and also to enhance interference cancellation. Recent advances also include techniques such as [75], [76] and [77], which use a distributed approach to compute the beamformer. Techniques such as adding a diagonal loading to the correlation matrix [78–81], the robust Capon beamformer of [33,35] (RCB), which uses the eigenvalue decomposition of the correlation matrix to compute the beamformer, and techniques based on worst-case performance optimization [34, 35] are some examples of robust beamformers. Many others can be found in the literature, (see [35] and references therein, for instance). However, most of these techniques are costly to compute (for instance, the beamformers of [33] and [81]

have cubic computational complexity in the number of sensors in the array), which make them difficult to implement.

In this chapter, we focus on arrays for which the number of signal sources is less than the number of sensors, such that the correlation matrix can become ill-conditioned when the ratio of the source to the noise power is high. We propose ℓ_1-norm regularized algorithms to regularize the matrix, and we show that this approach enhances the signal-to-interference-plus-noise ratio (SINR) performance at a low computational cost. The use of ℓ_1 regularization has the additional advantage of making the algorithm robust against sensor failure. For this purpose, we employ a modified version of the homotopy algorithm [69], which is an ℓ_1-norm regularized technique used in many applications, such as recovery of sparse signals from noisy measurements [37] and channel estimation [36]. Homotopy is generally applied to solve sparse systems of equations, and it helps the selection of the minimum amount of regularization required to compute the solution, reducing the bias. In the approach of this chapter, the homotopy strategy is not applied to sparse systems of equations, but to regularize the correlation matrix in a low-cost way. The proposed algorithms are extensions of the adaptive re-weighting homotopy (ARH) of [37] to the complex domain (C-ARH). We also develop new low-cost iterative versions of the C-ARH algorithms, suitable for adaptive beamforming. We show that we obtain techniques that can be applied at cost $\boldsymbol{\mathcal{O}}(N^2)$ (where N is the number of array elements), if the number of interferers remains constant with an increase of N. This case could model radar and sonar applications [82], assuming that the number of targets is approximately constant. In applications for which the number of interferers grows linearly with the increase of N, such as in MIMO communications [83], for which larger arrays are designed to serve a higher number of users, we show that the computational complexity increases, but our approach is still advantageous for a range of values of N.

In this text, we present the C-ARH and the multi-candidate (MC)-C-ARH algorithms, which we proposed in [40], and the iterative versions of these techniques, developed in [41]. We show that the iterative approach is doubly advantageous, since it improves the SINR performance and also reduces the computational complexity.

The contributions presented in this chapter are summarized as follows.

1. We extend the ARH algorithm of [37] to the complex domain, and propose a modification with better performance (but higher computational complexity), the multi-candidate C-ARH.

2. We devise the iterative C-ARH (It-C-ARH) and the iterative MC-C-ARH (It-MC-C-

ARH) algorithms to further improve the SINR performance with a reduced computational cost. We show that the iterative approaches outperform their non-iterative counterparts, improving the steady-state SINR.

3. We show how the DCD algorithm (presented in Chapter 4) can be used to further reduce the computational complexity of the C-ARH-based algorithms. For beamforming applications, its is shown that the iterative techniques applying DCD have computational complexity proportional to $\mathcal{O}(N^2)$, if the number of interferers does not increase linearly with N. We also show that our approach is still advantageous for a range of values of N, if the number of interferers increases linearly with the number of sensors. In this case, the computational cost is $\mathcal{O}(N^5)$, but N^5 is multiplied by a low value.

4. An analysis of properties of the proposed algorithms is presented along with an assessment of their computational complexity.

5. We present a simulation study comparing the proposed algorithms to existing robust techniques. We show that the iterative algorithms using the DCD present a small SINR performance degradation when compared to the RCB of [33], the adaptive beamformer with variable loading (VL) of [80] and the general-linear-combination-based robust Capon beamformer (GLC) [81] algorithms, but under some conditions on the relation between N and the number of interferers, the iterative algorithms require less computations, and are also robust against sensor failure, a property that these methods do not have.

This chapter is organized as follows: Section 5.1 presents the system model and the problems for which the C-ARH-based techniques are proposed. In Section 5.2, we present the C-ARH and MC-C-ARH algorithms. In Section 5.3, we propose the iterative versions of C-ARH and MC-C-ARH, while in Section 5.4 we use the DCD algorithm to obtain low-cost algorithms. Section 5.5 presents the analyses of the algorithms, and Section 5.6 shows simulation results. We conclude the chapter in Section 5.7.

5.1 System Model and Problem Statement

Consider a uniform linear array (ULA) with N sensors, and assume S signals, where one arrives from the desired direction of arrival θ_{d}, and the other $S-1$ signals are interferers, as described in Figure 15. Additionally, assume that the signal of interest and the

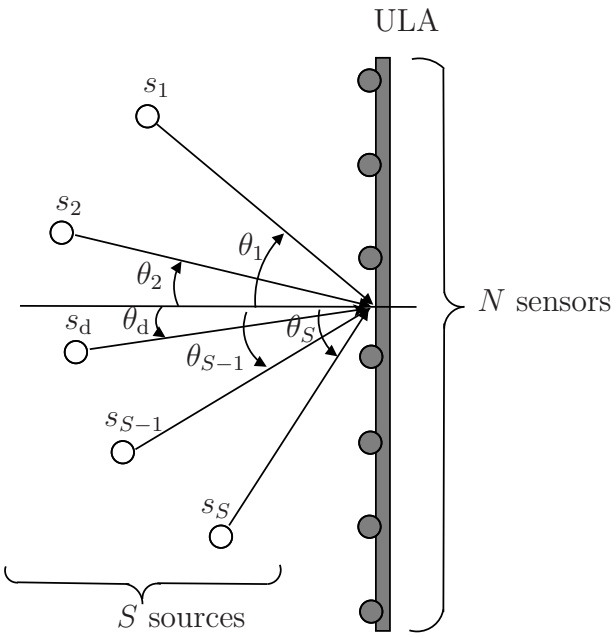

Figure 15: Uniform linear array

interferers are uncorrelated and that $\theta_{\rm d}$ is known. Define the $N \times S$ matrix $\mathbf{B}$, where each column $\mathbf{b}_k$ corresponds to a steering vector [7] as given by

$$\mathbf{b}_k = [1 \;\; e^{-j\pi\sin(\theta_k)} \;\; \ldots \;\; e^{-j\pi(N-1)\sin(\theta_k)}]^T, \;\; 1 \leq k \leq S.$$

At snapshot n, the sensor array data are modeled as

$$\mathbf{u}(n) = \mathbf{B}\,\mathbf{s}(n) + \boldsymbol{\eta}(n), \tag{5.1}$$

where $\mathbf{s}(n)$ contains narrowband analytic signals produced by the S sources. $\boldsymbol{\eta}(n)$ is a vector of zero-mean, independent and identically distributed (i.i.d.) Gaussian noise with variance σ_η^2. The noise in each sensor is also assumed independent from the noise in other array elements. Without loss of generality, define $\theta_{\rm d} = \theta_1$ and $\mathbf{b}_{\rm d} = \mathbf{b}_1$. The coefficients of the MVDR beamformer [7] are given by

$$\boldsymbol{\varpi}_{\rm MVDR} = \boldsymbol{w}_{\rm MVDR}/\mathbf{b}_{\rm d}^H \boldsymbol{w}_{\rm MVDR}, \tag{5.2}$$

and $\boldsymbol{w}_{\rm MVDR}$ is the solution to

$$\mathbf{R}_{\rm t}\boldsymbol{w}_{\rm MVDR} = \mathbf{b}_{\rm d}. \tag{5.3}$$

$\mathbf{R}_{\rm t}$ is the theoretical $N \times N$ correlation matrix [7], which is defined as

$$\mathbf{R}_{\rm t} = E\{\mathbf{u}(n)\mathbf{u}^H(n)\} = \mathbf{R}_{\rm dInt} + \mathbf{R}_\eta, \tag{5.4}$$

where

$$\mathbf{R}_{\text{dInt}} = \mathbf{B}E\{\mathbf{s}(n)\mathbf{s}^H(n)\}\mathbf{B}^H \tag{5.5}$$

and

$$\mathbf{R}_{\eta} = E\{\boldsymbol{\eta}(n)\boldsymbol{\eta}^H(n)\} = \sigma_{\eta}^2\mathbf{I}_N. \tag{5.6}$$

Express (5.1) explicitly in terms of the direction of interest (subscript d) and the interference (subscript Int), i.e,

$$\mathbf{u}(n) = \mathbf{b}_{\text{d}}\, s_{\text{d}}(n) + \mathbf{B}_{\text{Int}}\,\mathbf{s}_{\text{Int}}(n) + \boldsymbol{\eta}(n), \tag{5.7}$$

to write $\mathbf{R}_{\text{dInt}}$ as

$$\mathbf{R}_{\text{dInt}} = \mathbf{R}_{\text{d}} + \mathbf{R}_{\text{Int}}, \tag{5.8}$$

where $\mathbf{R}_{\text{d}} = \sigma_{\text{d}}^2\mathbf{b}_{\text{d}}\mathbf{b}_{\text{d}}^H$, $\mathbf{R}_{\text{Int}} = \mathbf{B}_{\text{Int}}E\{\mathbf{s}_{\text{Int}}(n)\mathbf{s}_{\text{Int}}^H(n)\}\mathbf{B}_{\text{Int}}^H$ and σ_{d}^2 is the variance of the signal of interest.

Note that the computation of the beamformer requires the solution of a linear system of equations. When the number of sources is smaller than the number of sensors, $\mathbf{R}_{\text{t}}$ can become ill-conditioned, requiring some form of regularization to compute $\boldsymbol{w}_{\text{MVDR}}$. In addition, if the measurements of some sensors are not available (if a sensor fails), a dimensional-reduction of the system of equations can be made before the introduction of the regularization, reducing the computations to obtain the beamformer. We consider both situations and show that by using the ℓ_1-norm regularized algorithms presented in this text, one can estimate the beamformer and improve the SINR performance with only $\boldsymbol{\mathcal{O}}(N^2)$ computations (if the number of interferers does not increase with N), while techniques such as [33] require $\boldsymbol{\mathcal{O}}(N^3)$ computations. The proposed ℓ_1 algorithms are also shown to be robust against errors in estimating faulty sensors.

5.1.1 Small number of interference sources

Consider that the number of interferers plus the source of interest is S, and that S is smaller than the number of sensors N. Assume that the interference sources are uncorrelated among each other, such that $\text{rank}(E\{\mathbf{s}\mathbf{s}^H\}) = S$, and assume that the angles θ_k are selected such that $\text{rank}(\mathbf{B}) = S$. Using properties of the rank of matrices (see [46]) one can show that[1]

$$\text{rank}(\mathbf{R}_{\text{dInt}}) = \text{rank}(\mathbf{B}E\{\mathbf{s}\mathbf{s}^H\}\mathbf{B}^H) = S. \tag{5.9}$$

[1]Note that we drop here time indices to simplify notation.

Since $\text{rank}(\mathbf{R}_{\text{dInt}}) = S < N$, the eigenvalue decomposition [46] of $\mathbf{R}_{\text{dInt}}$ is given by

$$\mathbf{R}_{\text{dInt}} = \mathbf{V} \begin{bmatrix} \mathbf{D}_0 & \mathbf{0}_{S\times(N-S)} \\ \mathbf{0}_{(N-S)\times S} & \mathbf{0}_{(N-S)\times(N-S)} \end{bmatrix} \mathbf{V}^H, \tag{5.10}$$

where the columns of $\mathbf{V}$ are the eigenvectors of $\mathbf{R}_{\text{dInt}}$, and $\mathbf{D}_0$ is a diagonal matrix containing the S non-zero eigenvalues of $\mathbf{R}_{\text{dInt}}$. Recalling that $\mathbf{V}\mathbf{V}^H = \mathbf{I}_N$, and using (5.6) and (5.10) in (5.4), we obtain

$$\mathbf{R}_{\text{t}} = \mathbf{V} \begin{bmatrix} \mathbf{D}_0 + \sigma_\eta^2 \boldsymbol{I}_S & \mathbf{0}_{S\times(N-S)} \\ \mathbf{0}_{(N-S)\times S} & \sigma_\eta^2 \boldsymbol{I}_{(N-S)} \end{bmatrix} \mathbf{V}^H. \tag{5.11}$$

Using eq. (5.11), it is easy to see that the condition number [46] of $\mathbf{R}_{\text{t}}$ is given by

$$\kappa(\mathbf{R}_{\text{t}}) = (d_{0_{\text{MAX}}} + \sigma_\eta^2)/\sigma_\eta^2, \tag{5.12}$$

where $d_{0_{\text{MAX}}}$ stands for the maximum eigenvalue of $\mathbf{R}_{\text{dInt}}$. Equation (5.12) shows that $\mathbf{R}_{\text{t}}$ becomes ill-conditioned if the noise power is much smaller than $d_{0_{\text{MAX}}}$. In this case, a regularization can be added to $\mathbf{R}_{\text{t}}$ to improve the computation of $\boldsymbol{w}_{\text{MVDR}}$ in (5.2). While this is usually done using ℓ_2-norm regularization (diagonal loading) [78], we will show that our ℓ_1-norm regularized algorithms also reduce the effects of the ill-conditioned $\mathbf{R}_{\text{t}}$, improving the computation of the beamformer and leading to low-cost algorithms. In addition, the use of homotopy allows our algorithms to choose just the right amount of regularization, reducing bias.

5.1.1.1 Reducing the system of equations when there are faulty sensors in the array

When a sensor j is not working properly, its measurements should be discarded. This information can be incorporated into the model with a modification of eq. (5.7), by introducing an $N \times N$ diagonal matrix $\mathbf{E}$, i.e.,

$$\mathbf{u}(n) = \mathbf{E}\left(\mathbf{b}_{\text{d}}\,\mathbf{s}_{\text{d}}(n) + \mathbf{B}_{\text{Int}}\,\mathbf{s}_{\text{Int}}(n) + \boldsymbol{\eta}(n)\right), \tag{5.13}$$

where the diagonal entries of $\mathbf{E}$ are equal to 0 for faulty sensors that do not contribute to beamforming, and 1 otherwise. When $e_{jj} = 0$, we zero the j-th element of all steering vectors, which eliminates the signal produced by sensor j. Using (5.13) to compute the correlation matrix, we obtain

$$\mathbf{R}_t = \mathbf{E}\left(\mathbf{R}_{\text{dInt}} + \mathbf{R}_\eta\right)\mathbf{E}. \tag{5.14}$$

Assuming that the array has $F < N$ faulty sensors (but that there are still more working sensors than sources, i.e, $N - F > S$), and that these sensors are grouped such that $\mathbf{E}$ has the last F diagonal elements equal to 0, $\mathbf{R}_t$ is given by

$$\mathbf{R}_t = \begin{bmatrix} \mathbf{R}_{\text{RR}} & \mathbf{0}_{(N-F)\times F} \\ \mathbf{0}_{F\times(N-F)} & \mathbf{0}_F \end{bmatrix}, \tag{5.15}$$

where $\mathbf{R}_{\text{RR}} = \tilde{\mathbf{R}}_{\text{dInt}} + \sigma_\eta^2 \mathbf{I}_{N-F}$ and $\tilde{\mathbf{R}}_{\text{dInt}}$ is a matrix obtained from the first $N-F$ columns and the first $N-F$ rows of $\mathbf{R}_{\text{dInt}}$. Ideally, if we know matrix $\mathbf{E}$, we can define $\mathbf{b}_{\text{RR}}$ as the entries of $\mathbf{b}_{\text{d}}$ related with the $N-F$ working sensors, so that we can solve the lower-dimension system of equations

$$\mathbf{R}_{\text{RR}}\boldsymbol{w} = \mathbf{b}_{\text{RR}}, \tag{5.16}$$

where $\boldsymbol{w}$ contains the $N-F$ non-zero entries of $\boldsymbol{w}_{\text{MVDR}}$. Matrix $\mathbf{R}_{\text{RR}}$ will still be ill-conditioned when the eigenvalues of $\tilde{\mathbf{R}}_{\text{dInt}}$ are much higher than the noise power, and regularization might be necessary to reduce the condition number and improve the computation of $\boldsymbol{w}$.

From (5.15) and (5.16), we see that matrix $\mathbf{E}$ is required to obtain $\mathbf{R}_{\text{RR}}$. In general, $\mathbf{E}$ is unknown and has to be estimated beforehand. In this text, we use the energy detection method [84] to estimate the faulty sensors. However, we show through simulations that our ℓ_1-regularized algorithms are robust against errors in detecting faulty sensors, so that one might choose not to check for sensor failure.

5.2 Proposed complex homotopy algorithms

The complex homotopy algorithm (CH) was proposed in [36] as an extension of the real-valued homotopy technique [69] to the complex field. For both cases, the algorithm solves the optimization problem[2]

$$\underset{\boldsymbol{w}}{\text{minimize}} ||\mathbf{A}\boldsymbol{w} - \mathbf{y}||_2^2/2 + h||\boldsymbol{w}||_1, \tag{5.17}$$

where $\boldsymbol{w}$ is a column vector with M entries, $\mathbf{A}$ is a $P \times M$ matrix, $\mathbf{y}$ is a $P \times 1$ vector, $||\boldsymbol{w}||_1 = \sum_{i=1}^{M} |w_i|$ and $h > 0$ is a regularization parameter. The initial value of h should be large at first, but is reduced till $h = 0$ to compute $\boldsymbol{w}$, as we explain below.

[2]Note that we introduce the algorithms for a general $P \times M$ matrix $\mathbf{A}$. For our beamforming approach, $M = P = (N-F)$ (the number of working sensors), $\mathbf{A}$ is an estimated version of $\mathbf{R}_{\text{RR}}$ and $\mathbf{y}$ corresponds to $\mathbf{b}_{\text{RR}}$.

The CH algorithm iteratively solves (5.17) using a support set Γ that is updated at every iteration. The solution to (5.17) is obtained defining the function

$$f(\boldsymbol{w}) = ||\mathbf{A}\boldsymbol{w} - \mathbf{y}||_2^2/2 + h||\boldsymbol{w}||_1, \tag{5.18}$$

and then computing the subgradient [85] of $f(\boldsymbol{w})$, that must be equal to zero, that is

$$\partial f(\boldsymbol{w}) = -\boldsymbol{A}^H(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{w}) + h\partial||\boldsymbol{w}||_1 = 0. \tag{5.19}$$

The subgradient of $\partial||w_i||_1$ of the i-th element is given by

$$\partial||w_i||_1 = \begin{cases} w_i/|w_i|, & \text{if } w_i \neq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{5.20}$$

Using (5.20), then eq. (5.19) can be written equivalently to two optimality conditions [36]

$$\begin{array}{l} \mathbf{a}_i^H(\mathbf{A}\boldsymbol{w} - \mathbf{y}) = -hz_i, \text{ for all } i \in \Gamma \\ |\mathbf{a}_i^H(\mathbf{A}\boldsymbol{w} - \mathbf{y})| < h, \text{ for all } i \in \Gamma_C \end{array}, \tag{5.21}$$

where Γ_C is the complement of Γ, and z_i is the i-th entry of the vector $\mathbf{z}$, which is obtained applying the sign function elementwise on $\boldsymbol{w}$. $\mathbf{a}_i$ is the i-th column of $\mathbf{A}$. For a large enough h, the solution of (5.17) will be $\boldsymbol{w} = \mathbf{0}$. The algorithm starts by computing $\max_i(|\mathbf{a}_i^H(\mathbf{A}\boldsymbol{w} - \mathbf{y})|)$, used to initialize h with the largest value for which Γ is non-empty. At each iteration, one element is added or removed from Γ, and h is moved to $h-\epsilon$, where ϵ is chosen so that $h - \epsilon$ is a breakpoint, i.e., the first point for which the new solution to (5.21) will need to add or remove an index to Γ. Denoting the new solution by $\boldsymbol{w} + \epsilon\partial\boldsymbol{w}$, (5.21) becomes

$$\begin{array}{l} \mathbf{A}_\Gamma^H(\mathbf{A}_\Gamma\boldsymbol{w}_\Gamma - \mathbf{y}_\Gamma) + \epsilon\mathbf{A}_\Gamma^H\mathbf{A}_\Gamma\partial\boldsymbol{w}_\Gamma = -h\mathbf{z}_\Gamma + \epsilon\mathbf{z}_\Gamma \\ |\mathbf{a}_i^H(\mathbf{A}\boldsymbol{w} - \mathbf{y}) + \epsilon\mathbf{a}_i^H\mathbf{A}\partial\boldsymbol{w}| < h - \epsilon, i \in \Gamma_C \end{array}, \tag{5.22}$$

where we use subscript Γ to define $\mathbf{A}_\Gamma$ as a matrix that contains only the columns of $\mathbf{A}$ for which the indices are in Γ. Similarly, $\boldsymbol{w}_\Gamma$, $\Delta\boldsymbol{w}_\Gamma$, $\boldsymbol{y}_\Gamma$ and $\boldsymbol{z}_\Gamma$ contain only the entries of $\boldsymbol{w}$, $\Delta\boldsymbol{w}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ with indices in Γ, respectively.

For a sufficiently small ϵ, $\partial\boldsymbol{w}$ is constant, so that we can use the first equation of (5.22), to compute $\partial\boldsymbol{w}_\Gamma$ – which corresponds to update only the entries of $\partial\boldsymbol{w}$ for which the indices are in the support set, while the other entries remain equal to zero – as the solution of the linear system of equations

$$\mathbf{A}_\Gamma^H\mathbf{A}_\Gamma\partial\boldsymbol{w}_\Gamma = \boldsymbol{z}_\Gamma. \tag{5.23}$$

To obtain the smallest value of ϵ for which the support must be updated, we substitute

$\partial \boldsymbol{w}$ in the second equation of (5.22). Then, we update h with

$$h \leftarrow h - \epsilon. \tag{5.24}$$

The algorithm continues until $h = 0$ or some stopping criterion is met. References [69] and [36] present a detailed description of the algorithm.

5.2.1 The Complex Adaptive Re-Weighting Homotopy Algorithm

In [37] the real-valued homotopy algorithm was modified to solve the ℓ_1-weighted optimization problem

$$\underset{\boldsymbol{w}}{\text{minimize}} ||\mathbf{A}\boldsymbol{w} - \mathbf{y}||_2^2/2 + \sum_{i=1}^{M} h_i |w_i|, \tag{5.25}$$

where h_i are positive weights. The motivation to modify the optimization problem and solve (5.25) instead of (5.17) was the possibility to adjust different weights to penalize the solution coefficients, which could be applied to reduce bias in the estimation of nonzero coefficients [37].

The ARH algorithm applies a re-weighting approach to quickly compute $\boldsymbol{w}$, when the column vector $\mathbf{h}$, which contains the weights of (5.25), is replaced by a re-weighting vector $\tilde{\mathbf{h}}$. The idea behind the algorithm is that the solution moves to $\boldsymbol{w} + \delta\partial\boldsymbol{w}$ when $\mathbf{h}$ moves towards $\tilde{\mathbf{h}}$ along a straight line $(1-\delta)\mathbf{h} + \delta\tilde{\mathbf{h}}$, for $\delta \in [0, 1]$, where $\partial\boldsymbol{w}$ does not depend on δ. The re-weighting is adaptively adjusted after every homotopy iteration, according to the changes in the solution and on the support set. Each $\tilde{h}_i$ can be adjusted separately from the others, such that the weights given to the elements in Γ can be set to shrink at faster rate than the other elements of $\boldsymbol{w}$, reducing the bias and also accelerating the convergence to the solution. The criterion to select the re-weighting can follow a number of heuristics [37], but should favor the quick shrinkage of the weight of the parameters in Γ for faster convergence. Simulation results in [37] have shown that ARH yields better performance and reconstruction accuracy than ℓ_1-based solvers (YALL1 [86], SpaRSA [87], SPGL1 [88]) used for recovering sparse signals from noisy measurements, while it requires lower computational complexity.

To present the C-ARH technique, and later introduce the multi-candidate C-ARH algorithm, assume that $\mathbf{A}$, $\boldsymbol{w}$ and $\mathbf{y}$ are complex entities. For convenience, also assume that $\delta \in [0, \tilde{\delta}]$. In Section 5.2.2, we use different values of $\tilde{\delta}$ to construct a diverse set of possible solutions, which is exploited by the multi-candidate algorithm to improve the

performance. Considering these assumptions, we can use an approach similar to that applied to derive the CH algorithm. For this purpose, define the function

$$f(\boldsymbol{w}) = ||\mathbf{A}\boldsymbol{w} - \mathbf{y}||_2^2/2 + \sum_{i=1}^{M} h_i|w_i|. \tag{5.26}$$

Using the subgradient of $f(\boldsymbol{w})$ and setting it equal to zero, we can rewrite the resulting expression as the optimality conditions

$$\begin{array}{l} \mathbf{A}_\Gamma^H(\mathbf{A}_\Gamma\boldsymbol{w}_\Gamma - \mathbf{y}_\Gamma) = -\mathbf{H}\mathbf{z}_\Gamma \\ |\mathbf{a}_i^H(\mathbf{A}\boldsymbol{w} - \mathbf{y})| < h_i \in \Gamma_C \end{array}, \tag{5.27}$$

where $\mathbf{H} = \text{diag}(\mathbf{h}_\Gamma)$, and $\mathbf{z}$ is a vector whose entries are the signs of the corresponding entries in $\boldsymbol{w}$. When $\mathbf{h}$ moves to $(1-\delta)\mathbf{h} + \delta\tilde{\mathbf{h}}$, (5.27) changes to

$$\begin{array}{l} \mathbf{A}_\Gamma^H(\mathbf{A}_\Gamma\boldsymbol{w}_\Gamma - \mathbf{y}_\Gamma) + \delta\mathbf{A}_\Gamma^H\mathbf{A}_\Gamma\partial\boldsymbol{w}_\Gamma = -\mathbf{H}\mathbf{z}_\Gamma + \delta(\mathbf{H} - \tilde{\mathbf{H}})\mathbf{z}_\Gamma \\ |\mathbf{a}_i^H(\mathbf{A}\boldsymbol{w} - \mathbf{y}) + \delta\mathbf{a}_i^H\mathbf{A}\partial\boldsymbol{w}| < h_i + \delta(\tilde{h}_i - h_i), i \in \Gamma_C \end{array},$$

and $\tilde{\mathbf{H}} = \text{diag}(\tilde{\mathbf{h}}_\Gamma)$. To compute $\partial\boldsymbol{w}$, we solve the system of equations

$$(\mathbf{A}_\Gamma^H\mathbf{A}_\Gamma)\partial\boldsymbol{w}_\Gamma = (\mathbf{H} - \tilde{\mathbf{H}})\mathbf{z}_\Gamma, \tag{5.28}$$

where $z_i = \mathbf{a}_i^H(\mathbf{A}\boldsymbol{w} - \mathbf{y})/h_i, i \in \Gamma$, and the elements of $\partial\boldsymbol{w}$ outside the support are set to zero.

We have to check if a breakpoint occurs to update the support. A breakpoint occurs in two situations: when an element of $\boldsymbol{w} \in \Gamma$ changes sign, or when one inequality becomes an equality in (5.21). When an element changes sign, it must be removed from Γ. Recall that $\boldsymbol{w}$ is updated as $\boldsymbol{w} \leftarrow \boldsymbol{w} + \delta\partial\boldsymbol{w}$. An element of $\boldsymbol{w}$ crosses zero when

$$\delta = -w_i/\partial w_i, \text{ for some } i \in \Gamma. \tag{5.29}$$

Define $\boldsymbol{w}_{\text{R}} = \mathcal{R}e\{\boldsymbol{w}\}$, $\boldsymbol{w}_{\text{I}} = \mathcal{I}m\{\boldsymbol{w}\}$, $\mathbf{d}_{\text{R}} = \mathcal{R}e\{\partial\boldsymbol{w}\}$ and $\mathbf{d}_{\text{I}} = \mathcal{I}m\{\partial\boldsymbol{w}\}$, and recall that δ must be a real number in the interval $[0, \tilde{\delta}]$. The parameter δ has a real value in eq. (5.29), only if

$$w_{\text{R}_i}/d_{\text{R}_i} = w_{\text{I}_i}/d_{\text{I}_i}, i \in \Gamma. \tag{5.30}$$

An element γ^- is removed from Γ if (5.30) holds for some i, and if (5.29) is in $[0, \tilde{\delta}]$, for the same breakpoint. If two or more breakpoints fulfill the restrictions, the smallest one is removed, which can be defined using

$$g = \min_+(-w_{\text{R}_i}/d_{\text{R}_i}), \text{ for all } w_{\text{R}_i}/d_{\text{R}_i} = w_{\text{I}_i}/d_{\text{I}_i}, i \in \Gamma$$

where $\min_+(\cdot)$ returns the smallest positive value in the argument. When g is empty, no term is removed, and C-ARH proceeds by choosing an element γ^+ that must be added to Γ. In this case, γ^+ is chosen by

$$\gamma^+ = \arg\max_{i \in \Gamma_C} |\mathbf{a}_i^H(\mathbf{A}\boldsymbol{w} - \mathbf{y})|. \tag{5.31}$$

The last step is the update of $h_i \in \Gamma_C$, which is given by

$$h_i \leftarrow \max_j |\mathbf{a}_j^H(\mathbf{A}\boldsymbol{w} - \mathbf{y})|, \text{ for all } i \in \Gamma_C. \tag{5.32}$$

The algorithm stops when the maximum $h_i \in \Gamma$ is smaller or equal to a pre-defined parameter τ. We summarize the algorithm in Table 17.

5.2.1.1 Re-weighting choice

We compute the re-weighting with

$$\tilde{h}_i = \min\left(\varsigma, \varsigma/\varrho|w_i|\right), \text{ for all } i \in \Gamma, \tag{5.33}$$

where $\varrho = N||\boldsymbol{w}||_2^2/||\boldsymbol{w}||_1^2$ and we use $\varsigma = 2\sigma_\eta^2$ in the algorithms proposed in this text[3]. This re-weighting is based on an approach proposed in [37]. We chose (5.33) since it allows the selection of different re-weighting values, but with a maximum pre-defined re-weighting of ς. Term ϱ introduces information of how the support size changes through the iterations. As $\boldsymbol{w}$ converges to the solution and elements are added to the support set, the value of the relation $||\boldsymbol{w}||_2^2/||\boldsymbol{w}||_1^2$ decreases. This information can be exploited to measure changes in the support set, and to update $\tilde{h}_i$. $|w_i|$ captures the influence of the solution entries in the re-weighting: for larger values of w_i, the less is the value of the re-weighting, which helps to reduce bias.

We tried different re-weightings in the simulations presented in Section 5.6, and we selected (5.33) since it provided the best SINR performance for our beamforming scenario.

5.2.1.2 Computational cost

The main contribution to the computational cost of each C-ARH iteration comes from computing

$$\mathbf{A}^H(\mathbf{y} - \mathbf{A}\boldsymbol{w}) = \mathbf{A}^H\mathbf{y} - \mathbf{A}^H\mathbf{A}\boldsymbol{w} \tag{5.34}$$

[3]Note that other values of ς and other re-weightings can also be applied.

and $\partial \boldsymbol{w}$. The term (5.34) does not explicitly appear in Table 17. However, if we consider the steps 2 and 9 (or 2 and 10, if the "else" condition of step 9 does not occur), we see that at every iteration, step 2 uses the elements of (5.34) computed with $\mathbf{a}_i$, $i \in \Gamma$, while step 9 (or 10) uses the elements which are calculated with the remaining $\mathbf{a}_i$. Recalling that steps 2 and 9 (or 2 and 10) occur at every iteration, we notice that (5.34) is computed every C-ARH step. This computation is costly in general, but it can be done with lower cost if some *a priori* information about $\mathbf{A}$ is available.

Table 17: C-ARH algorithm

Input: $\mathbf{A}$, $\mathbf{y}$, τ, $\tilde{\delta}$ **Output:** $\boldsymbol{w}$, Γ	
Initialize: $\partial \boldsymbol{w} = \mathbf{0}$, $\boldsymbol{w} = \mathbf{0}$, $h_i \leftarrow \max_i \lvert\mathbf{a}_i^H \mathbf{y}\rvert$ for all i, $\Gamma \leftarrow \arg\max_i \lvert\mathbf{a}_i^H \mathbf{y}\rvert$	
	Repeat:
1	Select $\tilde{\mathbf{h}}$
2	For all $i \in \Gamma$, compute $z_i = \mathbf{a}_i^H (\mathbf{y} - \mathbf{A}\boldsymbol{w}) / h_i$
3	Solve $(\mathbf{A}_\Gamma^H \mathbf{A}_\Gamma)\partial \boldsymbol{w}_\Gamma = \mathrm{diag}(\mathbf{h}_\Gamma - \tilde{\mathbf{h}}_\Gamma)\mathbf{z}_\Gamma$
4	Compute $\boldsymbol{w}_{\mathrm{R}} = \mathcal{R}e\{\boldsymbol{w}_\Gamma\}$, $\boldsymbol{w}_{\mathrm{I}} = \mathcal{I}m\{\boldsymbol{w}_\Gamma\}$, $\mathbf{d}_{\mathrm{R}} = \mathcal{R}e\{\partial \boldsymbol{w}_\Gamma\}$ and $\mathbf{d}_{\mathrm{I}} = \mathcal{I}m\{\partial \boldsymbol{w}_\Gamma\}$
5	$g = \min_+(-w_{\mathrm{R}_i}/d_{\mathrm{R}_i})$, for all $w_{\mathrm{R}_i}/d_{\mathrm{R}_i} = w_{\mathrm{I}_i}/d_{\mathrm{I}_i}$
6	$\delta = \min(g,\ \tilde{\delta})$
7	$\boldsymbol{w} = \boldsymbol{w} + \delta \partial \boldsymbol{w}$
8	$\mathbf{h}_\Gamma = \mathbf{h}_\Gamma + \delta(\tilde{\mathbf{h}}_\Gamma - \mathbf{h}_\Gamma)$
9	**if** $\delta < \tilde{\delta}$
	$\Gamma \leftarrow \Gamma \setminus \gamma^-$ ▷ Remove an element from Γ
	else
	$\gamma^+ = \arg\max_{i \in \Gamma_C} \lvert\mathbf{a}_i^H(\mathbf{A}\boldsymbol{w} - \mathbf{y})\rvert$
	$\Gamma \leftarrow \Gamma \cup \gamma^+$ ▷ Add a new element to Γ
	end
10	$h_i \leftarrow \max_j \lvert\mathbf{a}_j^H(\mathbf{A}\boldsymbol{w} - \mathbf{y})\rvert$, for all $i \in \Gamma_C$
	until $\max_i(h_i) \leq \tau$

In a general approach, $\mathbf{A}$ can vary during the algorithm computations, requiring $\mathbf{A}^H\mathbf{A}$ and $\mathbf{A}^H\mathbf{y}$ in (5.34) to be re-computed at every iteration. Using the fact that $\mathbf{A}^H\mathbf{A}$ is symmetric, both terms are computed with $P(2M^2+6M)$ additions and $P(2M^2+6M)-(M^2+3M)$ multiplications. On the other hand, when $\mathbf{A}$ is invariant through the iterations, pre-computation of $\mathbf{A}^H\mathbf{y}$ and $\mathbf{A}^H\mathbf{A}$ can be used to reduce the number of operations. In this case, $\mathbf{A}^H(\mathbf{y} - \mathbf{A}\boldsymbol{w})$ uses a matrix-vector product and the addition of two vectors, achieving a lower cost proportional to $|\Gamma|M$ per iteration (where $|\Gamma|$ is defined as the cardinality of Γ). The maximum cost per C-ARH iteration corresponds to $4|\Gamma|M + 5|\Gamma|$ multiplications, $4|\Gamma|M + 5|\Gamma|$ additions and $3|\Gamma|$ divisions, plus the computation of $\partial \boldsymbol{w}$ (which is the solution to a $|\Gamma| \times |\Gamma|$ system of equations), and the cost to obtain $\tilde{\mathbf{h}}$. The

re-weighting applied in this chapter uses an additional cost of $3|\Gamma| + 2$ multiplications, $3|\Gamma| - 2$ additions, $|\Gamma| + 1$ divisions and $|\Gamma|$ square-roots per iteration.

To compute the solution to (5.25), C-ARH executes a number of iterations, where it adds or removes elements to the support of $\boldsymbol{w}$. Assuming that K is the total number of non-zero entries in the solution, the minimum number of iterations required to compute $\boldsymbol{w}$ is K. In this case, the algorithm only adds elements to the support, and the solution is obtained with $(K^2 + K)(2M + 4) - 2K$ additions, $(K^2 + K)(2M + 4) + 2K$ multiplications, $2K^2 + 3K$ divisions, $(K^2 + K)/2$ square-roots, plus the solution of K systems of equations with dimension $p \times p$, where p starts at 1 and linearly increases up to K during the iterations. Note that when the algorithm removes elements from Γ, the number of iterations increases. To check how frequently elements are removed from the support, we used C-ARH to solve a large number of examples with different values for $\mathbf{A}$ and $\mathbf{y}$. We compared the number of elements removed with the total amount of iterations used to compute the sparse vector $\boldsymbol{w}$, and we noted that these events occur rarely (in less than 1% of the iterations). Therefore, we assume that the minimum complexity we just derived can be used as a reasonable approximation to the number of computations used by C-ARH.

5.2.1.3 C-ARH applied to beamforming

The algorithm presented in Table 17 can be applied to sparse systems of equations in general. For the purpose of this text, we use the C-ARH to obtain a low-cost method to regularize eq. (5.3) and compute the MVDR beamformer.

Let $\mathbf{R}(n)$ be an approximation to $\mathbf{R}_{\text{t}}$ and $\mathbf{R}_{\text{RR}}(n)$ be an approximation to $\mathbf{R}_{\text{RR}}$ at snapshot n. For beamforming, we solve eq. (5.25) using $\mathbf{A} = \mathbf{R}_{\text{RR}}(n)$ and $\mathbf{y} = \mathbf{b}_{\text{RR}}$, resulting in the following system of equations

$$\mathbf{R}_{\text{RR}}(n)\boldsymbol{w}(n) = \mathbf{b}_{\text{RR}}, \tag{5.35}$$

where the solution $\boldsymbol{w}(n)$ is used in

$$\boldsymbol{\varpi}_{\text{RR}}(n) = \boldsymbol{w}(n)/\mathbf{b}_{\text{RR}}^H\boldsymbol{w}(n) \tag{5.36}$$

to compute the beamformer.

To solve (5.35), we first need to obtain $\mathbf{R}(n)$ and $\mathbf{E}$, so that the diagonal entries of $\mathbf{E}$ can be used to access the contribution of the working sensors, allowing us to obtain

$\mathbf{R}_{\text{RR}}(n)$ and $\mathbf{b}_{\text{RR}}$. We assume that $\mathbf{R}(n)$ is iteratively updated with

$$\mathbf{R}(n) = \nu\mathbf{R}(n-1) + \mathbf{u}(n)\mathbf{u}^H(n), \tag{5.37}$$

where $0 \leq \nu < 1$ is the forgetting factor. Note that (5.37) can be written as

$$\mathbf{R}(n) = \sum_{j=0}^{n} \nu^{n-j}[\mathbf{u}(j)\mathbf{u}^H(j) + \xi\mathbf{I}], \tag{5.38}$$

where $\xi\mathbf{I}$ is the initial regularization. Taking the expectation of (5.38) and recalling that $E\{\mathbf{u}(j)\mathbf{u}^H(j)\} = \mathbf{R}_t$, one can show that [45]

$$E\{\mathbf{R}(n)\} = \mathbf{R}_t/(1-\nu), \text{ when } n \to \infty. \tag{5.39}$$

Due to normalization in (5.36), the constant $(1-\nu)$ does not affect the computation of the beamforming solution, allowing the use of $\mathbf{R}(n)$ in (5.35).

The estimate of $\mathbf{E}$ can be obtained from the diagonal elements of $\mathbf{R}(n)$, using a technique based on the energy detection method [84]. To explain this approach, recall equations (5.15) and (5.39). It is easy to see that there are only two possibilities for the diagonal elements of $\mathbf{R}(n)$, when $n \to \infty$, i.e.,

$$r_{jj}(n \to \infty) \approx \begin{cases} \sigma_\eta^2/(1-\nu), & \text{if the } j\text{-th sensor is faulty} \\ (\tilde{r}^2_{\text{dInt}_{jj}} + \sigma_\eta^2)/(1-\nu), & \text{otherwise.} \end{cases}$$

Assuming that the sources are uncorrelated, $\tilde{r}^2_{\text{dInt}_{jj}}$ is given by

$$\tilde{r}^2_{\text{dInt}_{jj}} = \sum_{i=1}^{S} \sigma^2_{s_i}, \tag{5.40}$$

where each $\sigma^2_{s_i}$ corresponds to the variance of the i-th source, and $\sigma^2_{s_1} = \sigma^2_{\text{d}}$. In this case, one expects that the diagonal entries related to faulty sensors must have smaller variance, since they only measure noise [40, 89, 90]. We exploit this fact to estimate $\mathbf{E}$.

To define if a sensor is faulty, consider that after a few snapshots it is possible to perceive that some diagonal entries of $\mathbf{R}(n)$ have higher values than others. The faulty sensors are identified using a threshold, based on the maximum $r_{jj}(n)$. We assume that if an element $r_{jj}(n)$ is at least 6dB smaller than the maximum entry, then the j-th sensor is faulty. The threshold is computed with

$$\text{Thr} = 10^{-0.6}\text{max}_j(r_{jj}(n)), \tag{5.41}$$

and all the diagonal entries of $\mathbf{R}(n)$ are compared to (5.41). If $r_{jj}(n)$ is smaller than Thr

for some j, then the j-th sensor is faulty, and e_{jj} is set to 0. Otherwise, we set $e_{jj} = 1$. This technique is easy to implement (it only requires one multiplication and N comparisons) and is efficient for finding the faulty sensors, as can be seen in our simulation results. To further reduce the complexity, we can apply (5.41) only for the first t snapshots, turn the estimation of $\mathbf{E}$ off, and then use the last estimated matrix $\mathbf{E}$ for the remaining snapshots. Using this approach, we assume that t snapshots are sufficient to obtain a good estimate of the faulty sensors.

The C-ARH algorithm applied to beamforming is summarized in Table 18. Recalling that $\mathbf{R}_{\text{RR}}(n)$ and $\mathbf{R}(n)$ are $M \times M$ and $N \times N$, respectively, the computational complexity corresponds to the cost of the algorithm in Table 17, plus the cost to update $\mathbf{R}(n)$ and to compute $\mathbf{E}$ and $\boldsymbol{\varpi}(n)$ – steps 1, 2 and 4 in Table 18. These steps require the additional cost of $3M^2+5M+3N+5$ multiplications, $2M^2+4M+2N-1$ additions and 1 division.

Table 18: C-ARH algorithm applied to beamforming

Input: $\mathbf{b}_{\text{d}}$, ξ, $\mathbf{u}(n)$, ν, $\tilde{\delta}$, τ, t **Output:**$\mathbf{h}(n)$	
Initialize: $\mathbf{R}(0) = \xi\mathbf{I}$, $\boldsymbol{w}(0) = \mathbf{0}$	
	for $n = 1, 2, \cdots$
1	$\mathbf{R}(n) = \nu\mathbf{R}(n-1) + \mathbf{u}(n)\mathbf{u}^H(n)$
2	**if** $n < t$ **then** compute $\text{Thr} = 10^{-0.6}\max_j(r_{jj}(n))$ and estimate $\mathbf{E}$ to obtain $\mathbf{R}_{\text{RR}}(n)$ and $\mathbf{b}_{\text{RR}}$
3	Use $\tilde{\delta}$ and τ in C-ARH to solve $\mathbf{R}_{\text{RR}}(n)\boldsymbol{w}(n) = \mathbf{b}_{\text{RR}} \Rightarrow \boldsymbol{w}(n)$ (see Table 17)
4	$\boldsymbol{\varpi}(n) = \boldsymbol{w}(n)/\mathbf{b}_{\text{RR}}^H\boldsymbol{w}(n)$
	end for

5.2.2 Multi-Candidate Complex Adaptive Re-Weighting Homotopy

In the C-ARH algorithm, when we choose $\tilde{\delta} = 1$, $\mathbf{h}$ moves towards $\tilde{\mathbf{h}}$. However, we can choose a different $\tilde{\delta}$ and define a re-weighting scheme that is a linear combination of $\mathbf{h}$ and $\tilde{\mathbf{h}}$. Since in general there is no information about the weighting vector that generates the most accurate $\boldsymbol{w}$, the combination of the two weighting vectors can be a better option than only $\tilde{\mathbf{h}}$. In this context, MC-C-ARH is proposed to exploit multiple weight choices.

We start with the definition of a set $\boldsymbol{\Delta}$ of N_C candidates for $\tilde{\delta}$. For each $\tilde{\delta} \in \boldsymbol{\Delta} = \{\delta_1, \delta_2 \ldots \delta_{N_C}\}$ the algorithm computes the corresponding solution $\boldsymbol{w}_i(n)$. A comparison criterion (e.g., the MSE or the SINR) is used to define the best solution computed for

each snapshot. The candidate with the best figure of merit is selected. In Table 19 we summarize the algorithm, applied to beamforming.

In general, $\mathbf{R}_{\mathrm{d}}$, $\mathbf{R}_{\mathrm{Int}}$ and $\mathbf{R}_{\eta}$ are not available, and an indirect method is required to select the candidate which provides the highest SINR. Define $\mathbf{R}_{\mathrm{Int}\eta} = \mathbf{R}_{\mathrm{Int}} + \mathbf{R}_{\eta}$ and recall that $\mathbf{R}_{\mathrm{d}} = \sigma_{\mathrm{d}}^2 \mathbf{b}_{\mathrm{d}} \mathbf{b}_{\mathrm{d}}^H$ and $\mathbf{b}_{\mathrm{d}}^H \boldsymbol{\varpi}_i(n) = 1$. The SINR for the i-th candidate is given by

$$\mathrm{SINR}_i(n) = \frac{\boldsymbol{\varpi}_i^H(n)\mathbf{R}_{\mathrm{d}}\boldsymbol{\varpi}_i(n)}{\boldsymbol{\varpi}_i^H(n)\mathbf{R}_{\mathrm{Int}\eta}\boldsymbol{\varpi}_i(n)} = \frac{\sigma_{\mathrm{d}}^2}{\boldsymbol{\varpi}_i^H(n)\mathbf{R}_{\mathrm{Int}\eta}\boldsymbol{\varpi}_i(n)}, \tag{5.42}$$

and it is maximized when $\boldsymbol{\varpi}_i^H(n)\mathbf{R}_{\mathrm{Int}\eta}\boldsymbol{\varpi}_i(n)$ is minimum. Since $\mathbf{R}_{\mathrm{Int}\eta}$ is unknown, (5.42) cannot be directly minimized. As an alternative, we note that the minimization of

$$\boldsymbol{\varpi}_i^H(n)\mathbf{R}_{\mathrm{t}}\boldsymbol{\varpi}_i(n) = \sigma_{\mathrm{d}}^2 + \boldsymbol{\varpi}_i^H(n)\mathbf{R}_{\mathrm{Int}\eta}\boldsymbol{\varpi}_i(n), \tag{5.43}$$

also maximizes the SINR, and an estimate $\mathbf{R}(n)$ of $\mathbf{R}_{\mathrm{t}}$ can be used to compute (5.43). However, the computation of (5.43) is costly, proportional to $\mathcal{O}(N^2)$. To reduce the number of computations, we propose a simpler method, with cost $\mathcal{O}(N)$.

Defining $\mathbf{R}_{\mathrm{Int}\eta} = \mathbf{D}_{\mathrm{Int}\eta} + \mathbf{G}$, where $\mathbf{D}_{\mathrm{Int}\eta} = \sigma_{\mathrm{Int}\eta}^2 \mathbf{I}$ has the diagonal entries of $\mathbf{R}_{\mathrm{Int}\eta}$, and $\mathbf{G}$ contains the other elements, we can write

$$\mathrm{SINR}_i(n) = \frac{1}{(\sigma_{\mathrm{Int}\eta}^2/\sigma_{\mathrm{d}}^2)||\boldsymbol{\varpi}_i(n)||_2^2 + (1/\sigma_{\mathrm{d}}^2)\boldsymbol{\varpi}_i^H(n)\mathbf{G}\boldsymbol{\varpi}_i(n)}. \tag{5.44}$$

If we consider only two candidates, and that $\mathrm{SINR}_1(n) > \mathrm{SINR}_2(n)$, then we obtain

$$||\boldsymbol{\varpi}_1(n)||_2^2 < ||\boldsymbol{\varpi}_2(n)||_2^2 + [\boldsymbol{\varpi}_2^H(n)\mathbf{G}\boldsymbol{\varpi}_2(n) - \boldsymbol{\varpi}_1^H(n)\mathbf{G}\boldsymbol{\varpi}_1(n)]/\sigma_{\mathrm{Int}\eta}^2. \tag{5.45}$$

Assuming that $\mathbf{G}$ is small compared to $\sigma_{\mathrm{Int}\eta}^2\mathbf{I}$, then the second term on the right-hand side of (5.45) can be neglected. Extending the idea to N_c candidates, we obtain the proposed selection algorithm

$$\boldsymbol{\varpi}_{\mathrm{MAX}}(n) = \boldsymbol{\varpi}_k(n) \;\text{ when }\; k = \arg\min_i(||\boldsymbol{\varpi}_i(n)||_2^2),\; i = 1, 2, \ldots, N_c. \tag{5.46}$$

Our simulations show that MC-C-ARH improves the SINR performance, when compared to C-ARH.

5.2.2.1 Computational cost

MC-C-ARH starts with the update of $\mathbf{R}(n)$ (step 1, Table 19) and the computation of $\mathbf{E}$ (step 2, Table 19). Then, the algorithm computes the C-ARH solution $\boldsymbol{w}_i(n)$ (step 3, Table 19), $\boldsymbol{\varpi}_i(n)$ (step 4, Table 19) and $||\boldsymbol{\varpi}_i(n)||_2^2$ (step 5, Table 19) for each candidate.

Recalling that $N \times N$ is the dimension of $\mathbf{R}(n)$, while $M \times M$ is the dimension of $\mathbf{R}_{\text{RR}}(n)$, $\mathbf{R}(n)$ is updated with $3M^2 - 3M + 3N$ multiplications and $2M^2 - 2M + 2N$ additions. The cost to compute $\boldsymbol{\varpi}_i(n)$ is $8M + 4$ multiplications, $6M - 1$ additions and 1 division, while the computation of $||\boldsymbol{\varpi}_i(n)||_2^2$ uses $2M$ multiplications and $2M - 1$ additions for each candidate. The total computational cost is $3M^2 + M(10N_c - 3) + 3N + 4N_c + 1$ multiplications, $2M^2 + M(8N_c - 2) + 2N - 2N_c$ additions, N_c divisions, plus the cost to solve the C-ARH algorithm N_c times.

Table 19: MC-C-ARH algorithm applied to beamforming

Input: $\mathbf{b}_{\text{d}}$, ξ, $\mathbf{u}(n)$, ν, $\boldsymbol{\Delta}$, τ, t	**Output:** $\boldsymbol{\varpi}_{\text{MAX}}(n)$
Initialize: $\mathbf{R}(0) = \xi\mathbf{I}, \boldsymbol{w}_i(0) = \mathbf{0}$, $\forall \delta_i \in \boldsymbol{\Delta}$	
	for $n = 1, 2, \cdots$
1	$\mathbf{R}(n) = \nu\mathbf{R}(n-1) + \mathbf{u}(n)\mathbf{u}^H(n)$
2	**if** $n < t$ **then** compute $\text{Thr} = 10^{-0.6}\max_j(r_{jj}(n))$ and estimate $\mathbf{E}$ to obtain $\mathbf{R}_{\text{RR}}(n)$ and $\mathbf{b}_{\text{RR}}$
	for all $\delta_i \in \boldsymbol{\Delta}$:
3	Use C-ARH with $\tilde{\delta} = \delta_i$ and τ to solve $\mathbf{R}_{\text{RR}}(n)\boldsymbol{w}_i(n) = \mathbf{b}_{\text{RR}} \Rightarrow \boldsymbol{w}_i(n)$ (see Table 17)
4	Compute $\boldsymbol{\varpi}_i(n) = \boldsymbol{w}_i(n)/\mathbf{b}_{\text{RR}}^H\boldsymbol{w}_i(n)$
5	Compute $||\boldsymbol{\varpi}_i(n)||_2^2$ for all i
	end for
6	$\text{m} = \arg\min_i\{||\boldsymbol{\varpi}_i(n)||_2^2\}$ ▷ Find the best candidate
7	$\boldsymbol{\varpi}_{\text{MAX}}(n) = \boldsymbol{\varpi}_{\text{m}}(n)$
	end for

5.3 Iterative algorithms using the C-ARH method

In this section, we propose iterative algorithms based on the C-ARH technique. The idea behind these approaches is that we can compute the solution at snapshot n by adding an update term to the solution obtained at snapshot $n-1$, reducing the computations to obtain $\boldsymbol{w}(n)$. For this purpose, consider that the solution to (5.35) at snapshot n is given by

$$\boldsymbol{w}(n) = \boldsymbol{w}(n-1) + \Delta\boldsymbol{w}(n), \tag{5.47}$$

where $\Delta\boldsymbol{w}(n)$ is the updating term. Since $\boldsymbol{w}(n-1)$ is known at snapshot n, we use (5.47) in (5.35), to obtain

$$\mathbf{R}_{\text{RR}}(n)\Delta\boldsymbol{w}(n) = \boldsymbol{\beta}(n), \tag{5.48}$$

where we define

$$\boldsymbol{\beta}(n) = \mathbf{b}_{\text{RR}} - \mathbf{R}_{\text{RR}}(n)\boldsymbol{w}(n-1). \tag{5.49}$$

In this case, we can write the minimization problem

$$\underset{\Delta \boldsymbol{w}}{\text{minimize}} ||\mathbf{R}_{\text{RR}}(n)\Delta\boldsymbol{w}(n) - \boldsymbol{\beta}(n)||_2^2/2 + \sum_{i=1}^{M} h_i|\Delta w_i(n)|, \tag{5.50}$$

which is similar to the minimization problem solved by C-ARH in (5.25). Therefore, we can define $\mathbf{A} = \mathbf{R}_{\text{RR}}(n)$, $\mathbf{y} = \boldsymbol{\beta}(n)$ and $\boldsymbol{w} = \Delta\boldsymbol{w}(n)$, and use C-ARH to compute $\Delta\boldsymbol{w}(n)$. The result is then applied in (5.47), to compute $\boldsymbol{w}(n)$. We call this approach the iterative C-ARH (It-C-ARH) algorithm.

The It-C-ARH algorithm computes $\boldsymbol{\beta}(n)$ at every snapshot, which requires the computation of a complex matrix-vector product and the addition of two complex vectors. Using this approach, the total number of operations corresponds to $4M^2$ multiplications and $4M^2$ additions. However, the computation of $\boldsymbol{\beta}(n)$ at snapshot n can be implemented less costly, using quantities computed in the previous snapshot. For this purpose, assume that $\mathbf{R}(n)$ is updated as presented in eq. (5.37), and use it in (5.49) to write

$$\begin{aligned}
\boldsymbol{\beta}(n) &= \mathbf{b}_{\text{RR}} - \left[\nu\mathbf{R}_{\text{RR}}(n-1) + \mathbf{u}_{\text{RR}}(n)\mathbf{u}_{\text{RR}}^H(n)\right]\boldsymbol{w}(n-1) \\
&= (1-\nu)\mathbf{b}_{\text{RR}} + \nu\mathbf{b}_{\text{RR}} - \nu\mathbf{R}_{\text{RR}}(n-1)\boldsymbol{w}(n-1) - \mathbf{u}_{\text{RR}}(n)\mathbf{u}_{\text{RR}}^H(n)\boldsymbol{w}(n-1) \\
&= (1-\nu)\mathbf{b}_{\text{RR}} + \nu[\mathbf{b}_{\text{RR}} - \mathbf{R}_{\text{RR}}(n-1)\boldsymbol{w}(n-1)] - \mathbf{u}_{\text{RR}}(n)\mathbf{u}_{\text{RR}}^H(n)\boldsymbol{w}(n-1)
\end{aligned} \tag{5.51}$$

where $\mathbf{u}_{\text{RR}}(n)$ contains only signals obtained from sensors working properly. We define the residue

$$\boldsymbol{\zeta}(n-1) = \mathbf{b}_{\text{RR}} - \mathbf{R}_{\text{RR}}(n-1)\boldsymbol{w}(n-1) \tag{5.52}$$

and

$$z(n) = \boldsymbol{w}^H(n-1)\mathbf{u}_{\text{RR}}(n) \tag{5.53}$$

to write

$$\boldsymbol{\beta}(n) = (1-\nu)\mathbf{b}_{\text{RR}} + \nu\boldsymbol{\zeta}(n-1) - \mathbf{u}_{\text{RR}}(n)z^*(n) \tag{5.54}$$

Using (5.51) and (5.47), $\boldsymbol{\zeta}(n)$ can be written in terms of the $\Delta\boldsymbol{w}(n)$, i.e.,

$$\boldsymbol{\zeta}(n) = \mathbf{b}_{\text{RR}} - \mathbf{R}_{\text{RR}}(n)(\boldsymbol{w}(n-1) + \Delta\boldsymbol{w}(n)) = \boldsymbol{\beta}(n) - \mathbf{R}_{\text{RR}}(n)\Delta\boldsymbol{w}(n), \tag{5.55}$$

which can be efficiently computed to reduce the computational cost. When C-ARH computes $\Delta\boldsymbol{w}(n)$, it computes only the K entries in the support of $\Delta\boldsymbol{w}(n)$. In this case, C-ARH gives us perfect knowledge of the K non-zero entries of $\Delta\boldsymbol{w}(n)$. With this information, we can exclude the columns of $\mathbf{R}_{\text{RR}}(n)$ that are multiplied by the zero entries of

$\Delta\boldsymbol{w}(n)$ in (5.55), such that the residue can be computed with $4KM$ multiplications and $4KM$ additions. Using this result to compute (5.51), the computational cost to calculate $\boldsymbol{\beta}(n)$ corresponds to $4KM + 6M$ multiplications and $4KM + 6M$ additions.

In Table 20 the It-C-ARH algorithm is presented, and in Table 21 we describe the iterative MC-C-ARH algorithm, introducing multiple candidates for $\tilde{\delta}$. Notice that we use the same criterion applied by MC-C-ARH to select the best candidate in the iterative multi-candidate technique.

Table 20: It-C-ARH algorithm applied to beamforming

Input: $\mathbf{b}_{\mathrm{d}}$, ξ, $\mathbf{u}$, ν, $\tilde{\delta}$, τ, t	**Output:** $\boldsymbol{\varpi}(n)$
Initialize: $\mathbf{R}(0) = \xi\mathbf{I}, \boldsymbol{w}(0) = \mathbf{0}$, $\boldsymbol{\zeta}(0) = \mathbf{0}$	
	for $n = 1, 2, \cdots$
1	$\mathbf{R}(n) = \nu\mathbf{R}(n-1) + \mathbf{u}(n)\mathbf{u}^H(n)$
2	**if** $n < t$ **then** compute $\mathrm{Thr} = 10^{-0.6}\max_j(r_{jj}(n))$ and estimate $\mathbf{E}$ to obtain $\mathbf{R}_{\mathrm{RR}}(n)$ and $\mathbf{b}_{\mathrm{RR}}$
3	$z(n) = \boldsymbol{w}^H(n-1)\mathbf{u}_{\mathrm{RR}}(n)$
4	$\boldsymbol{\beta}(n) = \nu\boldsymbol{\zeta}(n-1) - \mathbf{u}_{\mathrm{RR}}(n)z^*(n) + (1-\nu)\mathbf{b}_{\mathrm{RR}}$
5	Use C-ARH with $\tilde{\delta}$ and τ to solve $\mathbf{R}_{\mathrm{RR}}(n)\Delta\boldsymbol{w}(n) = \boldsymbol{\beta}(n) \Rightarrow \Delta\boldsymbol{w}(n)$, Γ (see Table 17)
6	Compute $\boldsymbol{w}(n) = \boldsymbol{w}(n-1) + \Delta\boldsymbol{w}(n)$
7	Compute $\boldsymbol{\zeta}_i(n) = \boldsymbol{\beta}(n) - \mathbf{R}_{\mathrm{RR}}(n)\Delta\boldsymbol{w}(n)$
8	Compute $\boldsymbol{\varpi}(n) = \boldsymbol{w}(n)/\mathbf{b}_{\mathrm{RR}}^H\boldsymbol{w}(n)$
	end for

Computational cost of the It-C-ARH algorithm: Compared to C-ARH (see Table 18), the iterative technique requires the additional computation of $z(n)$, $\boldsymbol{\beta}(n)$, $\boldsymbol{w}(n)$ and $\boldsymbol{\zeta}(n)$ (respectively steps 3, 4, 6 and 7 in Table 20). The term $(1-\nu)\mathbf{b}_{\mathrm{RR}}$ does not change through the snapshots and can be pre-computed to reduce the computations. The total implementation cost per snapshot is $3M^2+(15+4K)M+3N+5$ multiplications, $2M^2 + (14 + 4K)M + 2N + 2K - 3$ additions, 1 division, plus the cost to compute the C-ARH algorithm in Table 17.

Computational cost of the It-MC-C-ARH algorithm: The computation of the iterative multi-candidate algorithm differs from the MC-C-ARH algorithm in the addition of steps 3, 4, 6 and 7 in Table 21. With the additional steps, the complexity cost increases and is given by $3M^2 + (4KN_c + 20N_c - 3)M + 3N + 4N_c + 1$ multiplications, $2M^2+(4KN_c+18N_c-2)M+2N-2KN_c-4N_c$ additions, N_c divisions and the computation of C-ARH (Table 17) N_c times.

Table 21: It-MC-C-ARH algorithm applied to beamforming

Input: $\mathbf{b}_\mathrm{d}$, ξ, $\mathbf{u}$, ν, $\mathbf{\Delta}$, τ, t	**Output:** $\boldsymbol{\varpi}_\mathrm{MAX}(n)$
Initialize: $\mathbf{R}(0) = \xi\mathbf{I}, \boldsymbol{w}_i(0) = \mathbf{0}$, $\boldsymbol{\zeta}_i(0) = \mathbf{0}$, $\forall \delta_i \in \mathbf{\Delta}$	
	for $n = 1, 2, \cdots$
1	$\mathbf{R}(n) = \nu\mathbf{R}(n-1) + \mathbf{u}(n)\mathbf{u}^H(n)$
2	**if** $n < t$ **then** compute $\mathrm{Thr} = 10^{-0.6}\max_j(r_{jj}(n))$ and estimate $\mathbf{E}$ to obtain $\mathbf{R}_\mathrm{RR}(n)$ and $\mathbf{b}_\mathrm{RR}$
	for all $\delta_i \in \mathbf{\Delta}$:
3	$z_i(n) = \boldsymbol{w}_i^H(n-1)\mathbf{u}_\mathrm{RR}(n)$
4	$\boldsymbol{\beta}_i(n) = \nu\boldsymbol{\zeta}_i(n-1) - \mathbf{u}_\mathrm{RR}(n)z_i^*(n) + (1-\nu)\mathbf{b}_\mathrm{RR}$
5	Use C-ARH with δ_i and τ to solve $\mathbf{R}_\mathrm{RR}(n)\Delta\boldsymbol{w}_i(n) = \boldsymbol{\beta}_i(n) \Rightarrow \Delta\boldsymbol{w}_i(n)$, Γ (see Table 17)
6	Compute $\boldsymbol{w}_i(n) = \boldsymbol{w}_i(n-1) + \Delta\boldsymbol{w}_i(n)$
7	Compute $\boldsymbol{\zeta}_i(n) = \boldsymbol{\beta}(n) - \mathbf{R}_\mathrm{RR}(n)\Delta\boldsymbol{w}_i(n)$
8	Compute $\boldsymbol{\varpi}_i(n) = \boldsymbol{w}_i/\mathbf{b}_\mathrm{RR}^H\boldsymbol{w}_i$
9	Compute $\|\|\boldsymbol{\varpi}_i(n)\|\|_2^2$ for all i
	end for
10	$\mathrm{m} = \arg\min_i\{\|\|\boldsymbol{\varpi}_i(n)\|\|_2^2\}$ ▷ Find the best candidate
11	$\boldsymbol{\varpi}_\mathrm{MAX}(n) = \boldsymbol{\varpi}_\mathrm{m}(n)$
	end for

5.4 The homotopy algorithms using DCD iterations

The C-ARH algorithm solves a linear system of equations (step 3, Table 17), which is costly to compute. In this case, an efficient method to compute the solution is very important to keep the complexity low.

For this purpose, the use of DCD (see Section 4.1) is particularly interesting for the iterative algorithms, since the previous solution can be employed as an initial condition to DCD, allowing the use of a small N_u. The non-iterative algorithms cannot use this initial condition, and would require a large number of DCD iterations N_u to compute the solution. For this reason, we focus now on the iterative techniques. For It-C-ARH and It-MC-C-ARH, DCD is used to solve (5.28), where $\mathbf{A} = \mathbf{A}_\Gamma^H\mathbf{A}_\Gamma$ and $\mathbf{y} = (\mathbf{H} - \tilde{\mathbf{H}})\mathbf{z}_\Gamma$. To further save operations – just like we proposed for the DCD-WL-RLS in Section 4.2 – we can also select the forgetting factor as $\nu = 1 - 2^{-l}$, where l is a positive integer. This choice of ν substitutes multiplications by additions and bit-shifts in eqs. (5.37) and (5.51), reducing the complexity to update $\mathbf{R}_\mathrm{RR}(n)$ and $\boldsymbol{\beta}(n)$. As it is shown in Section 5.6, the algorithms using DCD outperform our previous approaches, with further reduction in the number of computations. Both the complex-valued cyclic-DCD and the DCD with a leading element can be applied in the proposed iterative algorithms. However, we use the

DCD with a leading element to compute the complexity presented in Table 24 and in the simulations of Section 5.6.

5.5 Analysis of some properties of the C-ARH-based algorithms

In this section, we summarize some analysis results that explain and give insight into the algorithms' behavior. We show why the C-ARH algorithm provides SINR performance gains even when the system of equations is not sparse, and we also define the range of values for $\tilde{\delta}$ for the multi-candidate algorithms. In addition, we compare the iterative and non-iterative algorithms to give some insights into the performance differences observed in the simulations. We then dedicate a section to summarize the computational complexity, and to provide some implementation tools to achieve low-cost algorithms.

5.5.1 C-ARH applied to regularize sparse and non-sparse problems

At the k-th homotopy iteration[4], the C-ARH algorithm (see Table 17) uses

$$\mathbf{A}_\Gamma^H(\mathbf{A}_\Gamma \boldsymbol{w}_\Gamma(k-1) - \mathbf{y}_\Gamma) + \delta \mathbf{A}_\Gamma^H \mathbf{A}_\Gamma \partial \boldsymbol{w}_\Gamma(k) = -\mathbf{H}\mathbf{z}_\Gamma(k) + \delta(\mathbf{H} - \tilde{\mathbf{H}})\mathbf{z}_\Gamma(k) \tag{5.56}$$

to compute the solution $\boldsymbol{w}(k) = \boldsymbol{w}(k-1) + \partial \boldsymbol{w}(k)$. Assume $\delta = 1$, and recall that $z_{\Gamma_i}(k) = w_{\Gamma_i}(k)/|w_{\Gamma_i}(k)|$. Equation (5.56) can be written as

$$\mathbf{A}_\Gamma^H \mathbf{A}_\Gamma \boldsymbol{w}_\Gamma(k) - \mathbf{A}_\Gamma^H \mathbf{y}_\Gamma = -\mathbf{D}(k)\boldsymbol{w}_\Gamma(k), \tag{5.57}$$

where we use $\mathbf{D}(k)\boldsymbol{w}_\Gamma(k) = \tilde{\mathbf{H}}\mathbf{z}_\Gamma(k)$, and $\mathbf{D}(k)$ is a diagonal matrix with diagonal entries

$$d_{ii}(k) = \tilde{h}_i/|w_{\Gamma_i}(k)|, \;\; i \in \Gamma. \tag{5.58}$$

Using $\mathbf{D}(k)$ in eq. (5.57), we obtain

$$\left(\mathbf{A}_\Gamma^H \mathbf{A}_\Gamma + \mathbf{D}(k)\right) \boldsymbol{w}_\Gamma(k) = \mathbf{A}_\Gamma^H \mathbf{y}, \tag{5.59}$$

showing that a diagonal regularization is introduced. As shown in Section 5.1.1, when the number of interferers is small, (5.57) is ill-conditioned, and the regularization provided by C-ARH helps to improve the estimates.

[4]Note that we use k to denote different homotopy iterations and avoid confusion with index n, which denotes snapshots.

5.5.2 Selection of the regularization parameter $\tilde{\delta}$

The values of the $\tilde{\delta}$ must all be in the interval $[0, 1]$ to avoid negative weights. In fact, the weight update is

$$\mathbf{h} \leftarrow (1 - \delta)\mathbf{h} + \delta\tilde{\mathbf{h}} \in \Gamma, \tag{5.60}$$

where we just reorganized step 8 in Table 17 to make clear the contribution of $\mathbf{h}$ and $\tilde{\mathbf{h}}$ to the right-hand side of the equation. It is easy to note in eq. (5.60) that δ must be in $[0, 1]$ to guarantee that $\mathbf{h}$ is always positive.

5.5.3 Differences between the iterative and non-iterative algorithms

The C-ARH algorithm computes an iterative solution to the minimization problem presented in eq. (5.25) (where $\mathbf{A} = \mathbf{R}_{\text{RR}}(n)$ and $\mathbf{y} = \mathbf{b}_{\text{RR}}$ for beamforming). It uses an ℓ_1-norm regularization to $\boldsymbol{w}(n)$, helping ill-conditioned systems of equations and also favoring sparse solutions. The iterative approaches, on the other hand, solve a modified minimization problem (see (5.50)), where the ℓ_1-norm regularization is applied to the solution update, $\Delta\boldsymbol{w}(n)$. Using $\Delta\boldsymbol{w}(n)$, $\boldsymbol{w}(n)$ is obtained with $\boldsymbol{w}(n) = \boldsymbol{w}(n-1)+\Delta\boldsymbol{w}(n)$. Since the iterative approach employs $\boldsymbol{w}(n-1)$ as an initial condition to compute $\boldsymbol{w}(n)$, it is expected to start each iteration closer to the real solution than C-ARH, since the later uses $\boldsymbol{w}(0) = \mathbf{0}$. In this case, It-C-ARH is expected to require less updates to compute $\boldsymbol{w}(n)$, which implies in a lower computational complexity.

An additional difference between the approaches concerns on how they compute $\boldsymbol{w}(n)$ when there is no sparsity, which is the case when they are applied to regularize ill-conditioned systems of equations. In this case, C-ARH tries to compute a sparse $\boldsymbol{w}(n)$, which is expected to be costly (since many iterations may be spent to change the support, trying to obtain a sparse solution) and also have poor performance. It-C-ARH, however, only searches for a sparse $\Delta\boldsymbol{w}(n)$. The update is designed to require a low number of iterations, such that it concentrates only on a few features at each iteration. Since $\boldsymbol{w}(n)$ is iteratively updated, all the entries of $\boldsymbol{w}(n)$ can be updated through the snapshots, such that a low-cost and regularized solution is expected to be obtained. Using this approach, It-C-ARH should outperform C-ARH, as verified in the simulations of Section 5.6.

5.5.4 Computational complexity

In this section, we summarize the computational complexity of the proposed algorithms. Tables 22 - 25 are arranged in a nested structure, each table adding a layer needed for the solution of the beamforming problem.

In Table 22, we show the computational cost per iteration of the C-ARH algorithm (as described in Table 17). The number of operations is presented as a function of the support size $|\Gamma|$ at each homotopy iteration. At every iteration, the algorithm computes the solution of a linear system of equations, with the dimension $|\Gamma| \times |\Gamma|$. Different approaches can be used to solve the system of equations, such as an LU factorization, which is an $\boldsymbol{\mathcal{O}}(|\Gamma|^3)$ technique, and the DCD algorithm, which is the low-cost alternative used in this dissertation. The total cost to compute $\boldsymbol{w}(n)$ (assuming that the maximum support size is given by K) is presented in Table 23. The cost to implement C-ARH using the DCD with a leading element is also detailed.

Table 24 shows the complexity of the algorithms proposed for beamforming. In our simulations comparing the iterative and non-iterative techniques (see Section 5.6), we notice that It-C-ARH and It-MC-ARH use a very low number of homotopy iterations per snapshot, which makes their computational cost much lower than the cost of C-ARH and MC-C-ARH. The simulations also indicate that the iterative algorithms have better SINR performance. When we use the DCD in the C-ARH algorithm, further reduction of the computational cost is obtained, since multiplications are replaced by additions and bit-shifts, and the parameter N_u can be designed to be a small number. Table 25 summarizes the number of computations used by the recursive algorithms using DCD iterations.

The dominant terms in the computational complexity of all the homotopy-based techniques are K^2M (see Table 23) and M^2 (see Tables 24 and 25). Since the relation between M and K is unknown and K could be, in principle, any function of M, the computational complexity could be higher than $\boldsymbol{\mathcal{O}}(M^2)$. However, as it is shown in our simulations in Section 5.6, the computational complexity of the DCD-based techniques depend on the relation of the number of interferers and M: if the number of interferers does not increase linearly with M, the computational complexity is $\boldsymbol{\mathcal{O}}(M^2)$. If the number of interferers grows linearly with M, the techniques are more costly to compute, but we show that there is still a range of values of M in which our methods are more advantageous than other algorithms from the literature.

Note that we use (5.33) for all techniques presented in this text. Selecting a different re-weighting, the required number of computations may change, modifying the values

presented in Tables 22 and 23.

Table 22: Computational complexity of C-ARH per homotopy iteration

Alg.	$+$	$\times$	$\div$	$\sqrt{\cdot}$
C-ARH	$4\|\Gamma\|(M+2)-2$	$4\|\Gamma\|(M+2)+2$	$4\|\Gamma\|+1$	$\|\Gamma\|$
	Plus the solution of a $\|\Gamma\| \times \|\Gamma\|$ linear system of equations			
C-ARH (DCD)	$4\|\Gamma\|(M+N_u+2)+4N_u+6M_b-2$	$4\|\Gamma\|(M+2)+2$	$4\|\Gamma\|+1$	$\|\Gamma\|$

Table 23: Minimum computational complexity of C-ARH per snapshot (when the total support size is K)

Alg.	$+$	$\times$	$\div$	$\sqrt{\cdot}$
C-ARH	$K^2(2M+4)+$ $K(2M+2)$	$K^2(2M+4)+$ $K(2M+6)$	$2K^2+3K$	$(K^2+K)/2$
	Plus the solution of K systems of equations with dimension			
	$p \times p$, where $p = 1, 2, \ldots, K$			
C-ARH (DCD)	$K^2(2M+2N_u+4)+$ $K(2M+6N_u+6M_b+2)$	$K^2(2M+4)+$ $K(2M+6)$	$2K^2+3K$	$(K^2+K)/2$

Table 24: Computational complexity per snapshot of the algorithms applied to beamforming

Alg.	$+$	$\times$	$\div$	C-ARH (Tab. 23)
C-ARH (beamf.)	$2M^2+4M$ $+2N-1$	$3M^2+5M$ $+3N+5$	1	1
MC-C-ARH	$2M^2+(8N_c-2)M$ $+2N-2N_c$	$3M^2+(10N_c-3)M$ $+3N+4N_c+1$	N_c	N_c
It-C-ARH	$2M^2+(4K+14)M$ $+2N+2K-3$	$3M^2+(4K+15)M$ $+3N+5$	1	1
It-MC-C-ARH	$2M^2+(4KN_c+18N_c-2)M$ $+2N+2KN_c-4N_c$	$3M^2+(4KN_c+20N_c-3)M$ $+3N+4N_C+1$	N_c	N_c

Table 25: Total computational complexity of the algorithms using DCD iterations

Alg.	$+$	$\times$	$\div$	$\sqrt{\cdot}$
DCD-It-C-ARH	$3(M^2+5M+N-1)$ $+K^2(2M+2N_u+4)$ $+K(6M+6N_u+6M_b+4)$	$2M^2+14M+2N$ $+5+K^2(2M+6)$ $+K(6M+6)$	$2K^2$ $+3K+1$	$(K^2+K)/2$
DCD-It-MC-C-ARH	$3(M^2-M+N)$ $+N_cK^2(2M+2N_u+6)$ $+N_cK(6M+6N_u+6M_b+4)$ $+N_c(18M-4)$	$2(M^2-M+N)+1$ $+N_c(K^2(2M+4)$ $+K(6M+24)+4)$	$N_c(2K^2$ $+3K+1)$	$N_c(K^2+K)/2$

5.6 Simulations

In our simulations, we compare the SINR performance of the proposed algorithms and techniques from the literature. We also study the computational complexity of our new techniques. For all scenarios, we consider a 64-sensor ULA. The direction of interest is fixed at angle 20^o, but the number of interferers can vary. We perform four groups of simulations. In the first scenario, we assume that there are 5 faulty sensors in the array, and we use a faulty-sensor detector to identify them. The faulty sensors are randomly selected, and all the techniques considered in the simulation use $\mathbf{R}_{\text{RR}}(n)$ to compute the beamformer. In the second simulation, we use the same conditions as before, but we assume that we are unable to identify the faulty elements. For this situation, we show that the proposed techniques are robust to errors in the detection of faulty elements. In the simulations of Section 5.6.3, we study the steady-state SINR performance of the proposed techniques when the SNR varies. For all these cases, we assume that the number of interferers is fixed and positioned in the angles 30^o, 45^o, 53^o and 60^o.

In Section 5.6.4, we perform simulations using only the DCD-based techniques to observe the relation between the number of homotopy iterations (K) and the number of sensors in the array (M), if the number of interferers vary. Our purpose is to model K as a function of M, and apply it in the expressions of Table 25 to obtain an accurate result to the number of multiplications used by each technique. Using this approach, we show that if the number of sensors is kept constant, which is a scenario that approximates sonar and radar applications [82], the number of multiplications required by the techniques is $\boldsymbol{\mathcal{O}}(M^2)$. If the number of interferers grows linearly with M, the computational cost is $\boldsymbol{\mathcal{O}}(M^5)$, but we show that there is a range of values in which our techniques are advantageous, less costly than the VL approach of [80], which is $\boldsymbol{\mathcal{O}}(M^2)$. This last scenario can be associated with problems of MIMO communications [83], in which the

number of users increases with the array dimension.

In the simulations applied to compare our techniques to other algorithms from the literature, (Sections 5.6.1 and 5.6.2), we present the SINR performance of C-ARH, MC-C-ARH, It-C-ARH, It-MC-C-ARH, DCD-It-C-ARH and DCD-It-MC-C-ARH, and we compare these methods to the $\mathcal{O}(M^3)$ RCB of [33], the RLS algorithm [7] without the addition of regularization, and to a method using a diagonal loading (DL) added to $\mathbf{R}_{\text{RR}}(n)$. We use the approach of [78] to regularize $\mathbf{R}_{\text{RR}}(n)$ by adding to the matrix the diagonal loading $10\sigma_\eta^2\mathbf{I}$, assuming exact knowledge of σ_η. The solution to $(\mathbf{R}_{\text{RR}}(n) + 10\sigma_\eta^2\mathbf{I})\boldsymbol{w}(n) = \mathbf{b}_{\text{RR}}$ is then used to compute the beamformer, which cannot be implemented using RLS. Here, we use the Matlab solution of systems of equations (the \ operator, LU factorization in most cases), at cost $\mathcal{O}(M^3)$. We also compare the new techniques with the GLC technique of [81] ($\mathcal{O}(M^3)$) and to the $\mathcal{O}(M^2)$ VL algorithm of [80]. The RCB, the GLC and the VL algorithms are briefly presented in Appendix B (see page 140). We present the SINR performance and the number of homotopy steps used by the proposed techniques, which expresses the maximum dimension of the system of equations solved by the homotopy-based algorithms. To determine the faulty sensors in the first simulation, we use the energy detection method, as presented in Section 5.2.1.3. Matrix $\mathbf{E}$ is estimated only during the first 100 snapshots. Afterwards, we freeze the last estimated value of $\mathbf{E}$ for the remaining snapshots.

For all simulations, the power of each interferer is 10 times the power of the signal of interest, $\sigma_{\text{d}}^2 = 1$. The SNR is 8dB, and the noise is a zero-mean Gaussian i.i.d. sequence. The signals produced by the sources are zero-mean binary sequences of -1 and 1, and we use eq. (5.37) to estimate $\mathbf{R}_{\text{t}}$ iteratively. The forgetting factor is given by $\nu = 1 - 2^{-6} \simeq 0.9844$ and $\mathbf{R}(0) = 10^{-3}\mathbf{I}$. The algorithms are adjusted in the first simulation to achieve the maximum SINR in the steady-state. We keep the same values for the parameters in the second simulation to study the effect of the incorrect identification of faulty sensors to the beamformer computation. In the third simulation, we vary the SNR and study the SINR gain of the techniques, assuming that the parameters of the algorithms are still adjusted as proposed in the first simulation. We use 500 snapshots, and assume that the algorithms achieve the steady-state after 400 snapshots. We then use the last 100 snapshots to compute the steady-state SINR. In the last simulation, we set the parameters as applied in the first simulation, but there are no faulty sensors and the the number of interferers is varied, allowing us to plot curves to study the relation between M and the number of homotopy iterations. We also use 500 snapshots in this case, and apply the last 100 snapshots to compute the mean number of homotopy iterations used

by the techniques.

The simulations presented in this section are obtained with the mean of 200 realizations.

5.6.1 Regularization when the faulty-sensor detector is applied

For the simulation presented in this section, we assume that there are 5 faulty sensors, and that the faulty-sensor detector is applied to identify and exclude them from the computation of the beamformer. To adjust C-ARH and MC-C-ARH, we use $\tau = 0.32$ and define three candidates for the MC-C-ARH algorithm, given by $\mathbf{\Delta} = [0.6\ 0.8\ 1]$. The iterative algorithms It-C-ARH and It-MC-C-ARH use a stopping parameter $\tau_{\text{It}} = 2$, while DCD-It-C-ARH and DCD-It-MC-C-ARH use $\tau_{\text{DCD}} = 5$, $N_u = 8$, $M_b = 16$ bits and $H = 2$. The multi-candidate iterative algorithms use the same set of candidates as selected for MC-C-ARH, and the constraint of the RCB is given by $\epsilon = 0.1$. Figures 16 and 17 present the SINR performance and the number of homotopy iterations required by the proposed techniques to compute the MVDR beamformer.

From Figure 16, one can see that the C-ARH algorithm and the DL approach have almost the same SINR performance after 400 snapshots, and that both algorithms are outperformed by the MC-C-ARH algorithm after 200 snapshots. The proposed iterative algorithms outperform the other homotopy-based techniques, and the highest SINR is achieved by the DCD-based algorithms. The iterative algorithms outperform their non-iterative counterparts with a lower number of homotopy iterations, which reduces the number of computations. For instance, consider the average number of iterations used by C-ARH (which is 58), It-C-ARH (3) and DCD-It-C-ARH (1.5), presented in Figure 17, after they achieve the steady-state. Using these values of K in the equations presented in Tables 23, 24 and 25, we compare the number of multiplications used by these methods. The C-ARH algorithm requires 400471 multiplications plus the cost to solve 58 systems of equations of sizes from 1×1 to 58×58. The It-C-ARH algorithm uses 26211 multiplications and solves only 3 systems of equations of sizes from 1×1 to 3×3. The DCD-It-C-ARH algorithm, the least costly of them, uses only 8386 multiplications. No additional multiplications are required to solve the systems of equations, since the DCD algorithm does not require multiplications.

The GLC and the DCD-It-C-ARH algorithms present the same SINR at the steady-state, but they are both outperformed by DCD-It-MC-C-ARH. The DCD-based iterative algorithms are only outperformed by the RCB and VL, which achieve an SINR perfor-

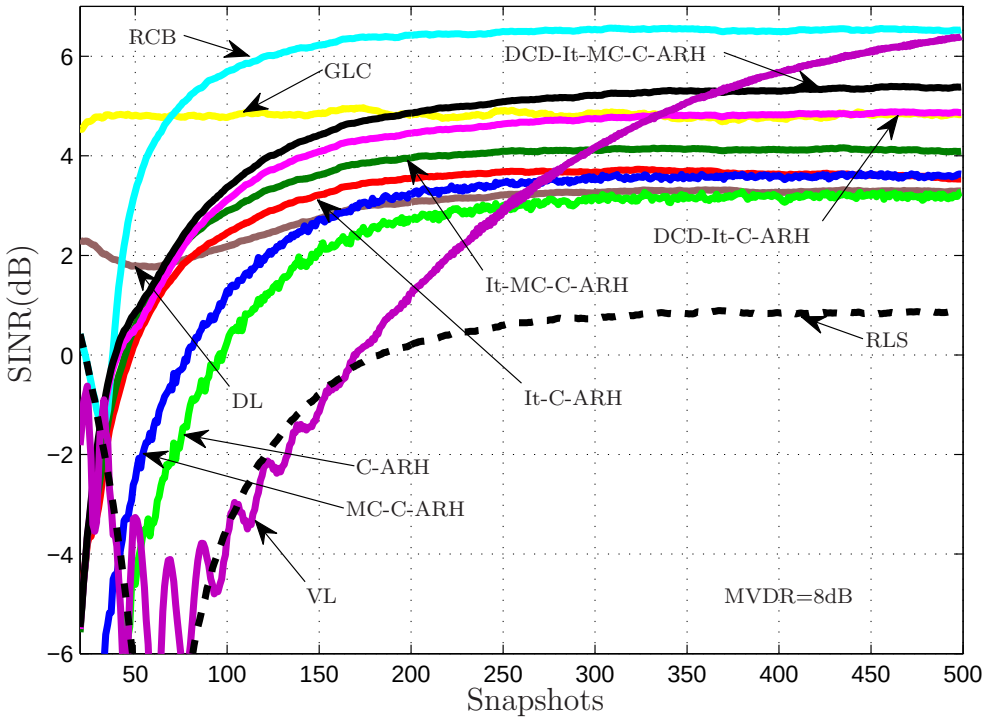

Figure 16: SINR performance when there are 5 faulty sensors. Mean of 200 realizations.

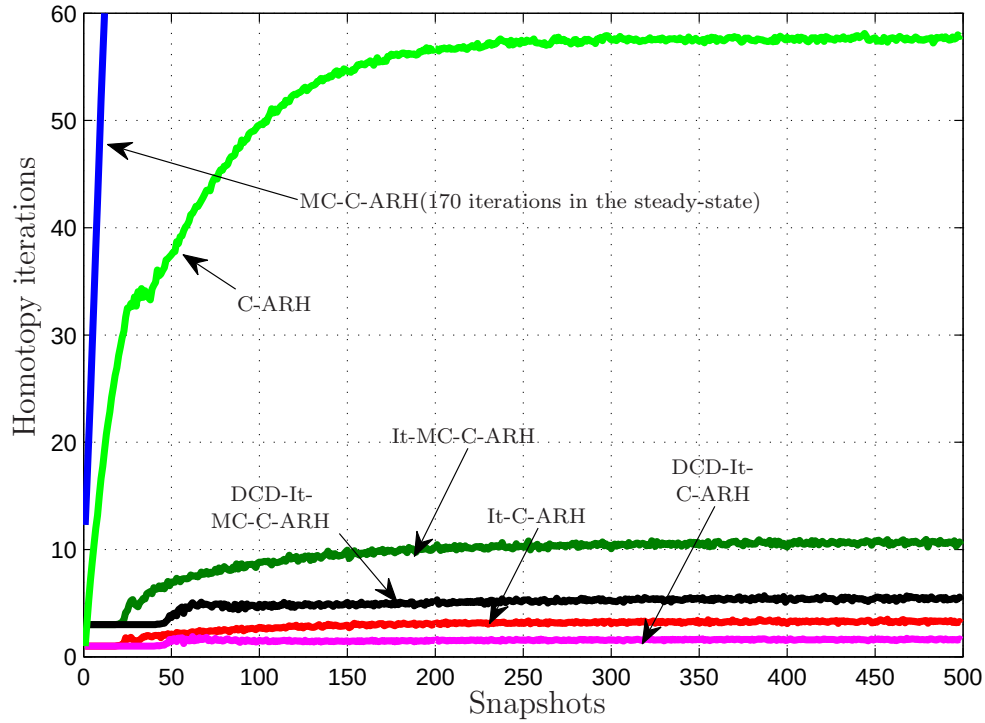

Figure 17: Average number of homotopy iterations per snapshot when there are 5 faulty sensors. Mean of 200 realizations.

mance 1.5dB higher than the DCD-It-MC-C-ARH. But the convergence rate of the VL is slower than that achieved by the DCD algorithms.

Notice that the RCB and GLC methods have computational cost of $\mathcal{O}(M^3)$, while VL algorithm is $\mathcal{O}(M^2)$, since its complexity is $20M^2+32M+5$ multiplications, $18M^2+24M-6$ additions and 3 divisions. In Section 5.6.4, we use simulations to show that the computational complexity of the DCD-based techniques is $\mathcal{O}(M^2)$, where M^2 is multiplied by 2, if the number of interferers does not increase with M. In this case, the DCD-based techniques are the less costly algorithms in this simulation.

5.6.2 Robustness to errors in the identification of faulty sensors in the array

In this simulation, we show that the iterative techniques are robust when the faulty sensors cannot be identified and excluded from the array. We assume that the array has 5 faulty sensors, but the detector is unable to identify them. For this situation, we cannot reduce the dimension of the correlation matrix, and the SINR performance of the algorithms might be affected. We maintain the value of the parameters used in Section 5.6.1, and we show that the iterative DCD-based algorithms present almost the same performance. Figure 18 and 19 show the results.

In this scenario, the performance degrades for all the algorithms, but the techniques using the DCD are less affected by the faulty sensors. The better performance of the DCD algorithms is explained by the very low number of homotopy iterations used by these techniques. To understand this, recall the algorithm presented in Table 17. To add an element to the support set, the algorithm selects the columns of the matrix which are most correlated to the residue. When the algorithm starts, it first adds to the support the highly correlated features of the matrix, for which the selection is not affected by the noise introduced by the faulty sensors. However, after a few steps of the C-ARH, the columns less correlated to the residue and the columns introduced by the faulty sensors can be mistaken, and the algorithm can add to the support wrong elements, leading to poor performance. Since the DCD iterative algorithms use a very low number of iterations, they only add to the support the most correlated elements, easier to identify, reducing the error and improving the SINR performance.

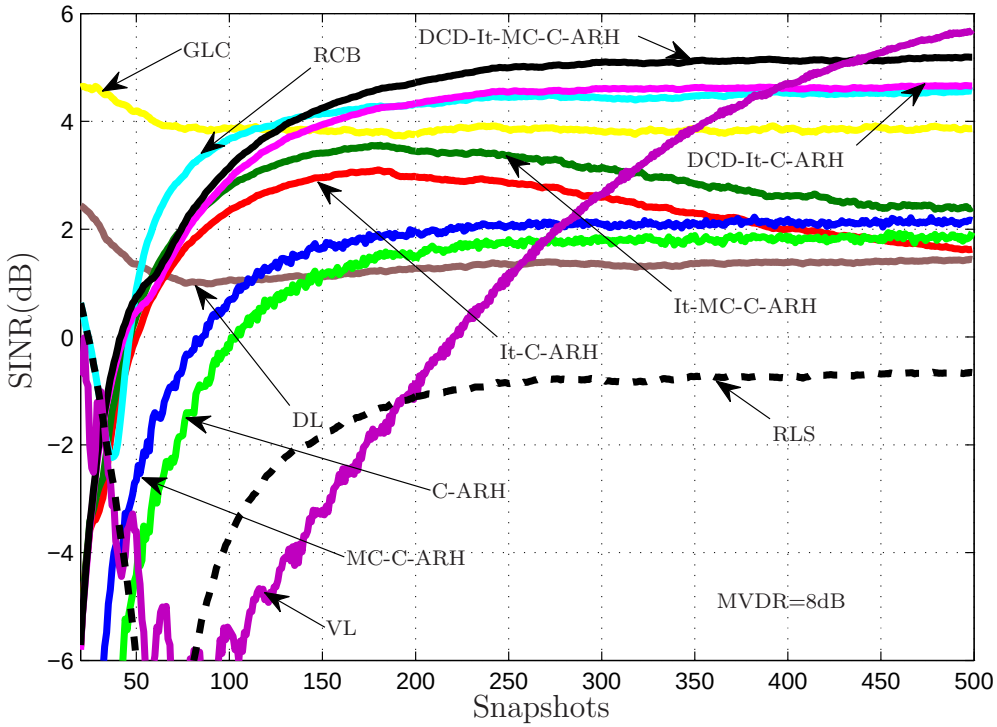

Figure 18: SINR performance when there are 5 faulty sensors, but they are not identified by the detector. Mean of 200 realizations.

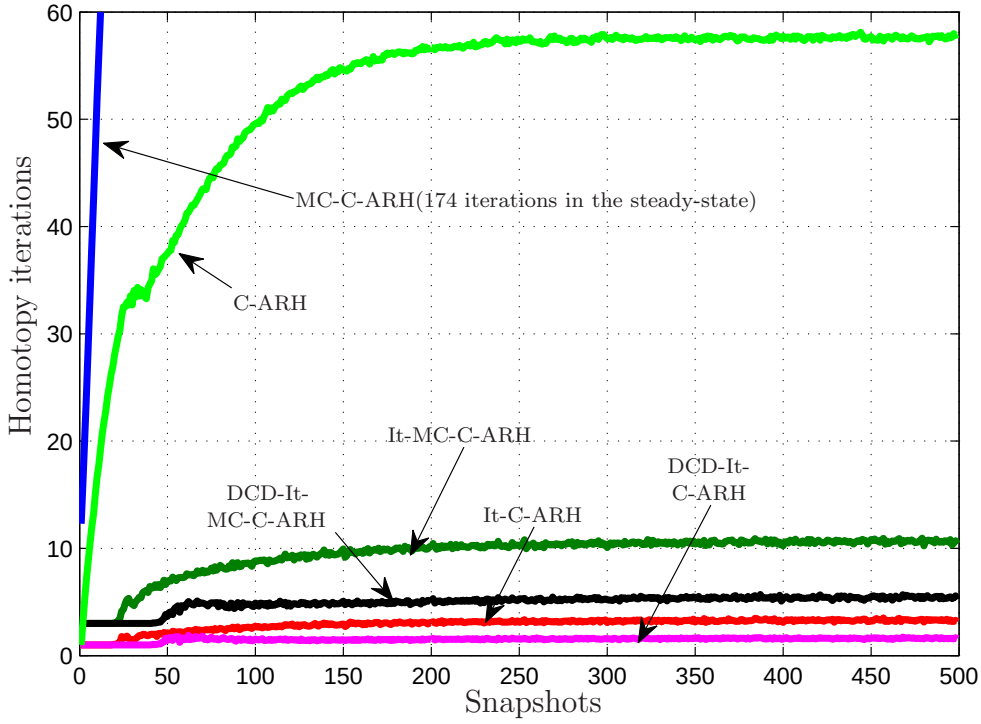

Figure 19: Average number of homotopy iterations per snapshot when there are 5 faulty sensors, but they are not identified by the detector. Mean of 200 realizations.

5.6.3 SINR × SNR curve

In this simulation, we study how the steady-state SINR varies when the SNR changes. We consider the scenario of the first simulation, but now we vary the SNR in the range of −10dB to 60dB. For this purpose, we compare the DCD-based iterative techniques with the RCB, VL, GLC and DL algorithms. We consider only DCD-based techniques in this simulation, since they presented the best SINR performance in the previous comparisons. Figure 20 presents the SINR × SNR curve, and Figure 21 shows the number of homotopy iterations used by the proposed algorithms.

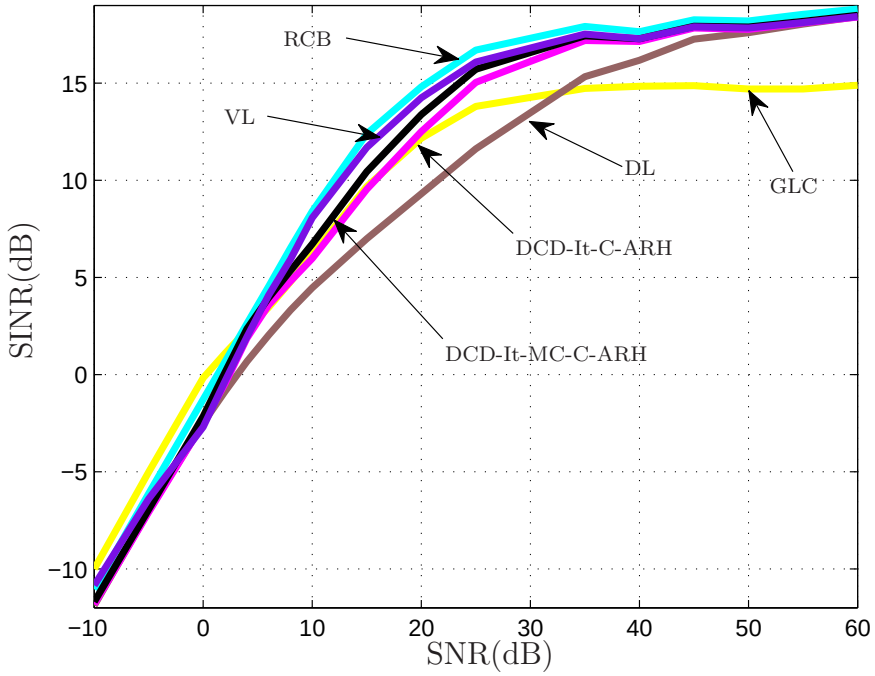

Figure 20: SINR performance of the proposed techniques when the SNR varies.

From Figure 20, we note that the GLC outperforms the other techniques when the SNR is low, but its performance degrades when the SNR increases. The RCB and VL outperform the other techniques when the SNR is higher than 0dB, but the maximum difference between these algorithms and the DCD-based techniques is approximately 2 dB, when the SNR=16dB. However, in this case the number of homotopy iterations is very low. In Figure 21, we see that when the SNR is low (less than 0dB), the DCD-based techniques require a higher number of homotopy iterations, and their performance is poor. When the SNR reaches 0dB, the number of iterations used by the techniques decreases. The low SNR probably hinders the selection of the terms that must be added to the support of C-ARH, leading to poor performance and a higher number of homotopy

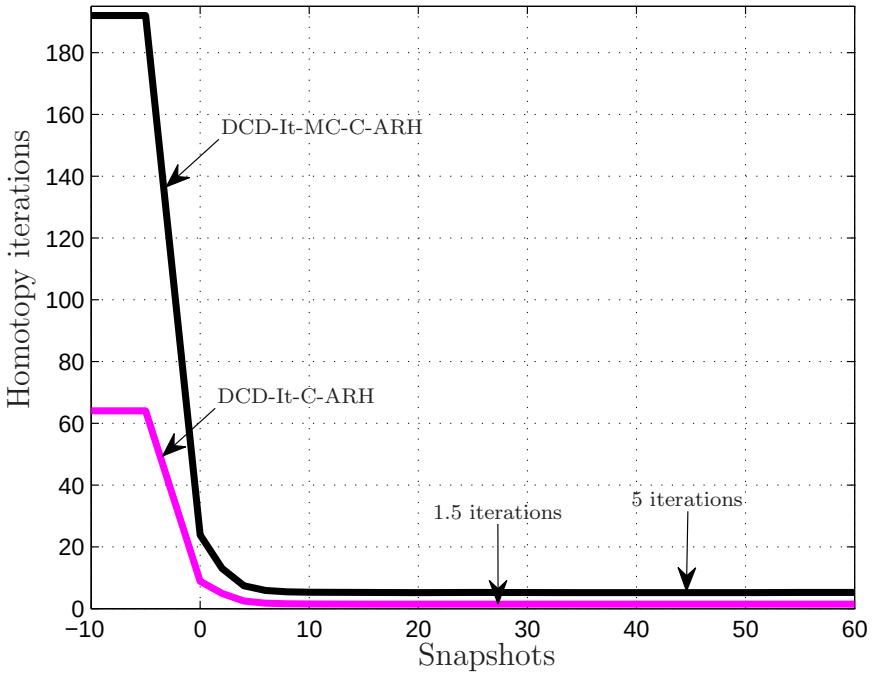

Figure 21: Average number of homotopy iterations used by the proposed techniques when the SNR varies.

iterations, which tends to the number of sensors. For higher SNR, the performance is improved and the number of iterations decreases. As we will show in Section 5.6.4, in a scenario where the number of interferers is constant if M increases, the computational cost of the DCD-based techniques is proportional to $2M^2$, while RCB and GLC have complexity proportional to M^3, and the VL algorithm has complexity quadratic on M (but multiplied by a factor 20). Since the number of homotopy iterations is high for low SNR, the expression of the number of multiplications probably is not able to capture the real number of operations. However, for higher SNR, (SNR$>$ 0dB), when the SINR rate improves, we expect the expression of the complexity to hold, and the trade-off between complexity and performance should favor the iterative methods. The relation between the SNR and the number of homotopy iterations used by the DCD-techniques will be addressed in future work.

5.6.4 Simulation study of the computational complexity of the DCD-It-C-ARH and DCD-It-MC-C-ARH algorithms

In this section, we present a simulation study on the computational complexity of the DCD-based techniques. We perform three simulations to model the relation between the number of homotopy iterations and the number of sensors in the array (M), so that we

can obtain a better approximation for the computational complexity of the techniques, presented in Table 25. The situations considered are the following:

1. In Figure 22, we study how the number of homotopy iterations varies if the number of sensors increases, but the number of interferers is kept constant. The number of sensors is varied in the set [8, 16, 20, 32, 64, 80, 128].

2. In the scenario used to obtain Figure 23, we assumed that the number of interferers depends linearly on the number of sensors, and increases at rate $M/8$. For this case, we use $M = [16, 32, 64, 128, 256]$.

3. In Figure 24, the number of interferers grows with the square root of M. The values of M applied to this simulation are [16, 25, 36, 49, 64, 81, 100, 121, 144].

To obtain the figures, we adjusted the parameters of the algorithms with the same values used in the first simulation, but there are no faulty sensors.

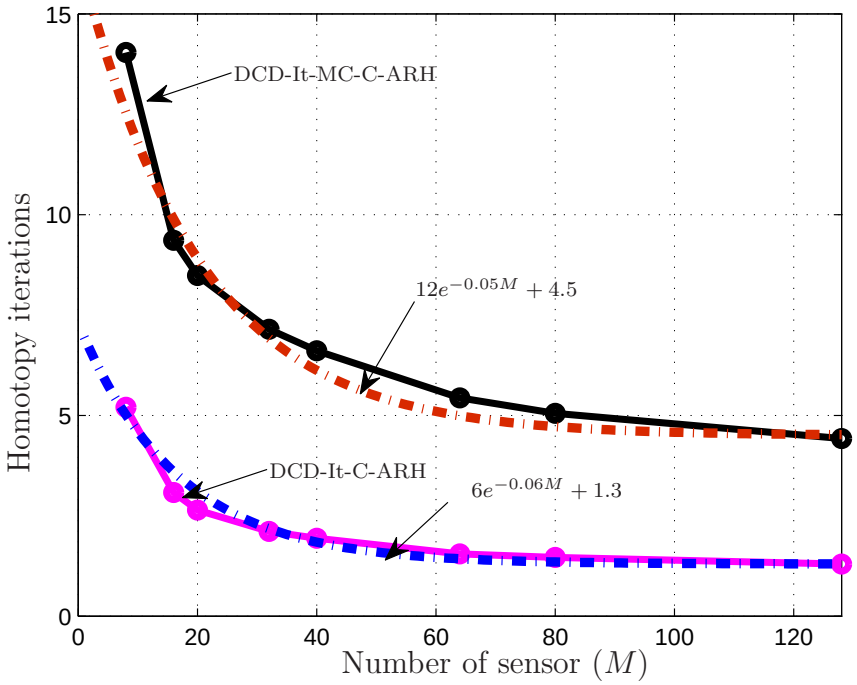

Figure 22: Average number of homotopy iterations versus the number of sensors in the array, when the number of interferers is constant.

In Figure 22, the number of iterations required by each technique to compute the beamformer starts at a low number of homotopy iterations and quickly decreases to a constant level (1.3 for DCD-It-C-ARH and 4.5 for the multi-candidate technique), which

can be modeled by an exponential decay. The number of homotopy iterations required by DCD-It-C-ARH can be modeled as $6 \cdot \exp(-0.06M) + 1.3$, while the relation is modeled by $12 \cdot \exp(-0.05M) + 4.5$ for DCD-It-MC-C-ARH. Applying these equations to the expressions obtained in Table 25 to compute the number of multiplications of each algorithm, the term $2M(6 \cdot \exp(-0.06M) + 1.3)^2$ appears in the expression of DCD-It-C-ARH. For DCD-It-MC-C-ARH, the term is given by $2M(12 \cdot \exp(-0.06M) + 4.5)^2$. To determine which is the dominant term in the computational cost, recall that $6 \cdot \exp(-0.06M) + 1.3 \to 1.3$ and $12 \cdot \exp(-0.06M) + 4.5 \to 4.5$ when M increases, so that $2M(6 \cdot \exp(-0.06M) + 1.3)^2 \to 3.4M$ and $2M(12 \cdot \exp(-0.06M) + 4.5)^2 \to 40.5M$. Therefore, the dominant element in the computational cost is $2M^2$, and the techniques are $\boldsymbol{\mathcal{O}}(M^2)$.

In Figure 23, we verify that if the number of interferers increases linearly with the number of sensors, at a rate of 1/8, the number of homotopy iteration increases quadratically. In the simulation of Figure 24, we observe a linear relation between M and the number of homotopy iterations, when the number of interferers increases with $\sqrt{M}$. Using the data from Figure 23, the equations that better model the relation are given by $5.7 \cdot 10^{-5}M^2 + 3.5 \cdot 10^{-2}M - 8.6 \cdot 10^{-2}$ for DCD-It-C-ARH, and $8.4 \cdot 10^{-5}M^2 + 1.2 \cdot 10^{-1}M + 6.9 \cdot 10^{-1}$ for the multi-candidate technique. Replacing these equations in the expressions of Table 25, we notice that our algorithms are $\boldsymbol{\mathcal{O}}(M^5)$, but the factor that multiplies M^5 is low. To show that our techniques can be advantageous even in this situation, we compare the number of multiplications of the DCD-based algorithms and VL, when M varies. Figure 25 shows the results, and we note that the DCD-It-C-ARH is less costly than VL for arrays in which the number of elements is up to 255, while the multi-candidate approach requires less multiplications if M is less than 140 elements.

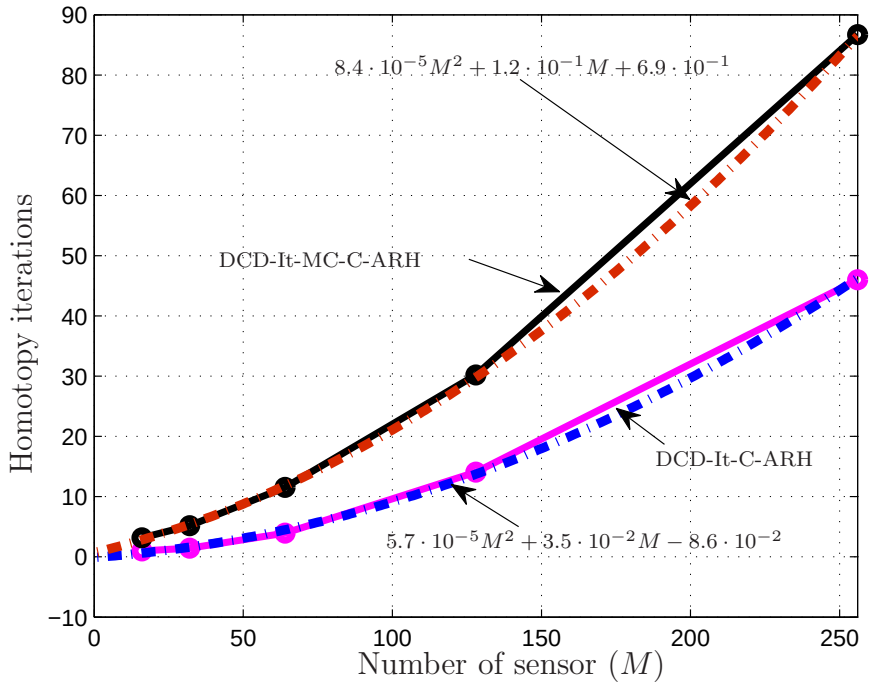

Figure 23: Average number of homotopy iterations versus the number of sensors in the array, when the number of interferers grows linearly at the rate of $M/8$.

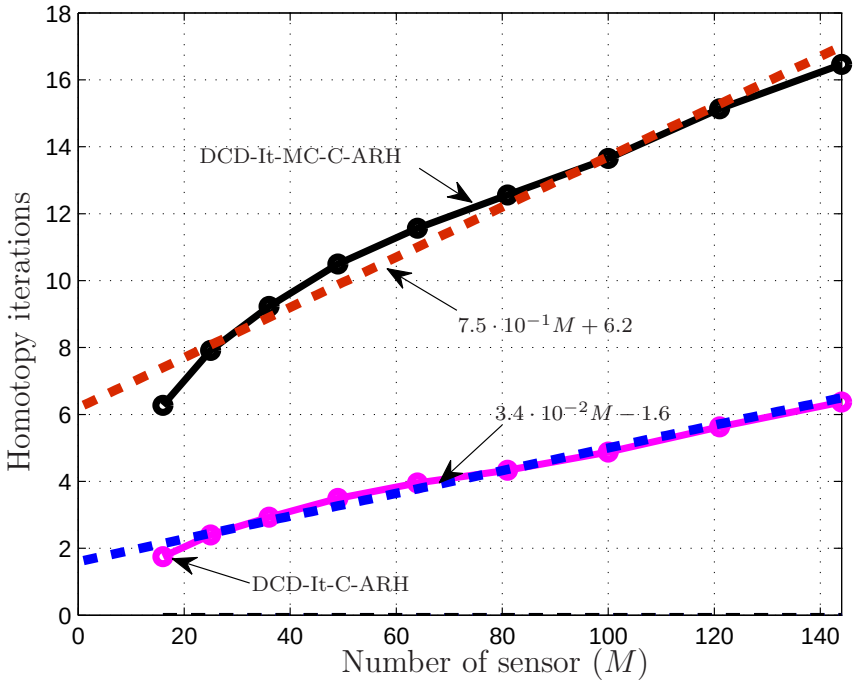

Figure 24: Average number of homotopy iterations versus the number of sensors in the array, when the number of interferers grows with $\sqrt{M}$.

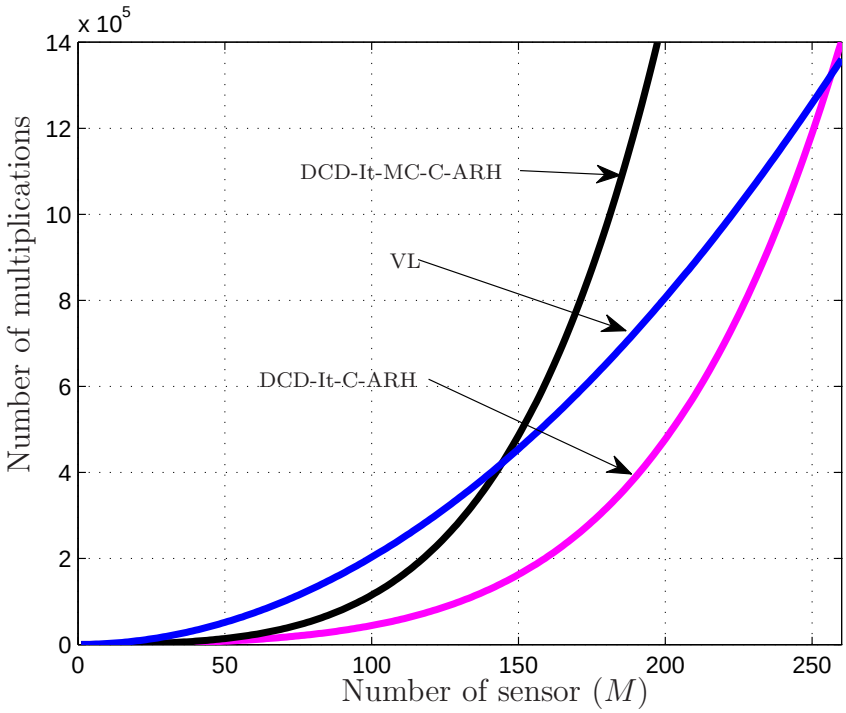

Figure 25: Number of multiplications versus the number of sensors in the array for VL, DCD-It-C-ARH and DCD-It-MC-C-ARH – the number of interferers varies linearly with M.

5.7 Conclusions

In this chapter, we have presented new beamforming algorithms based on the ℓ_1-norm regularized C-ARH technique. Our iterative algorithms have low computational complexity and are robust against ill-condition in the input autocorrelation matrix, which arises when the number of interferers is small compared to the number of sensors.

We compared our methods with the diagonal loading approach of [78], the RCB of [33], the GLC of [81] and the VL of [80], robust beamformers, in two situations: one in which faulty sensors are removed from the equations, and another in which faulty sensors are not correctly detected. In the first situation, our iterative algorithms perform better than C-ARH, MC-C-ARH, the diagonal loading approach of [78] and GLC, but they are outperformed by the RCB of [33] and the VL algorithm. In the second scenario, when the contribution of the faulty sensors is not removed from the sample correlation matrix and the algorithms keep the same parameters as in the first simulation, the DCD-based iterative methods are robust and outperform the other techniques, except for the VL algorithm. But VL converges more slowly than the DCD-based techniques to the steady-state.

The computational complexity of our new techniques was presented as a function of the number of sensors and the support size of C-ARH. Since the iterative algorithms require fewer homotopy iterations per snapshot than the C-ARH and MC-C-ARH algorithms, they are less costly to implement. For the DCD-based techniques, the number of computations is much lower, since many multiplications are replaced by bit-shifts and additions. To obtain more accurate expressions for the computational complexity of the DCD-based techniques, we performed simulations in which we studied the relation between the number of sensors and the average number of homotopy iterations used by the techniques, if the number of interferers varies. We considered two situations: in the first, the number of interferers is constant, and in the second, it increases linearly with the array dimension. For the first case, the complexity is quadratic in M, while the complexity is $\mathcal{O}(M^5)$ in the other approach. Based on these results, our DCD-iterative techniques are less costly than the RCB, the GLC and the DL algorithms, which are cubic in M, and also than VL (for which the cost is proportional to $20M^2$), if the number of interferers does not depends on M. If the number of interferers increases linearly on M, our techniques are less costly than VL only for a range of values of M.

The simulation comparing the SINR performance of the DCD-based algorithms to other techniques, when the SNR varies, has shown that the proposed algorithms are low-cost alternatives for SNR values higher than 0dB. For these values of SNR, if the number of interferers is constant, the complexity is $\mathcal{O}(M^2)$, so that the trade-off between SINR performance and computational complexity favors our techniques. For lower values of SNR, the performance is poor, the average number of homotopy iterations increases and they become costly.

Appendix B

In this appendix, we briefly present the RCB, the GLC and the VL algorithms.

The Robust Capon beamformer (RCB)

The RCB [33] is a technique which uses an eigenvalue decomposition of the correlation matrix to compute the beamformer. For this reason, it is costly to compute and is cubic on M. The algorithm is summarized in Table 26 (we use the same notation and variables used for our algorithms, to make the comparison among the techniques easier). Note that ϵ is a parameter used to adjust the regularization, and τ is a small constant to stop the update of λ in Table 26 (see step 6).

Table 26: Robust Capon beamformer

Input: $\boldsymbol{b}_{\text{RR}}$, $\boldsymbol{u}_{\text{RR}}$, ν,ϵ, τ	**Output:** $\boldsymbol{\varpi}(n)$
Initialize: $\boldsymbol{R}_{\text{RR}}(0) = \xi\boldsymbol{I}, \boldsymbol{w}(0) = \boldsymbol{0}$, $\tilde{\boldsymbol{b}}_{\text{RR}} = \boldsymbol{b}_{\text{RR}}$	
	for $n = 1, 2, \ldots$
1	$\boldsymbol{R}_{\text{RR}}(n+1) = \nu\boldsymbol{R}_{\text{RR}}(n) + \boldsymbol{u}_{\text{RR}}(n+1)\boldsymbol{u}_{\text{RR}}^H(n+1)$
2	Compute the eigenvalue decomposition of $\boldsymbol{R}_{\text{RR}}(n+1)$ to obtain $\boldsymbol{V}$ (the matrix of the eigenvectors) and $\boldsymbol{d}$, a vector with the eigenvalues.
3	Compute $\boldsymbol{z} = \lvert\boldsymbol{V}^H\tilde{\boldsymbol{b}}_{\text{RR}}\rvert$
4	Define $\lambda = 0$
5	Compute $y_i = z_i/(1+\lambda d_i)$
6	While $\lvert\lvert\lvert\boldsymbol{y}\rvert\rvert_2 - \epsilon\rvert > \tau$
7	$\lambda = \lambda + (\lvert\lvert\boldsymbol{y}\rvert\rvert_2^2 - \epsilon)/\sum_{i=1}^{M} \dfrac{2z_i^2 d_i}{(1+\lambda d_i)^3}$
8	end
9	Compute matrix $\boldsymbol{D}_0$, for which the diagonal entries are given by $1/\lambda d_i$
10	$\tilde{\boldsymbol{b}}_{\text{RR}} = \boldsymbol{b}_{\text{RR}} - \boldsymbol{V}\boldsymbol{D}_0\boldsymbol{V}^H\boldsymbol{b}_{\text{RR}}$
11	$\boldsymbol{R}_{\text{RR}}(n+1)\boldsymbol{w}(n+1) = \tilde{\boldsymbol{b}}_{\text{RR}} \Rightarrow \boldsymbol{w}(n+1)$
12	$\boldsymbol{\varpi}(n+1) = \dfrac{\boldsymbol{w}(n+1)}{\tilde{\boldsymbol{b}}_{\text{RR}}^H\boldsymbol{w}(n+1)}$
	end

Adaptive beamformer with variable loading (VL)

The VL technique uses an iterative variable loading approach to add regularization to the autocorrelation matrix and improve the computation of the beamformer. The approach of [80] is summarized in Table 27. One can show that the complexity of this approach is given by $20M^2 + 32M + 5$ multiplications, $18M^2 + 24M - 6$ additions and 3 divisions.

Table 27: Adaptive beamformer with variable loading

Input: $\boldsymbol{b}_{\text{RR}}$, $\boldsymbol{u}_{\text{RR}}$, ν, $\tau = 10\sigma_\eta^2$	**Output:** $\boldsymbol{\varpi}(n)$
Initialize: $\boldsymbol{R}_{\text{RR}}(0) = \xi \boldsymbol{I}, \boldsymbol{w}_0(0) = \boldsymbol{0}$, $\boldsymbol{p} = \boldsymbol{0}$	
	for $n = 1, 2, \ldots$
1	$\boldsymbol{R}_{\text{RR}}(n+1) = \dfrac{1-\nu^n}{1-\nu^{n+1}} \boldsymbol{R}_{\text{RR}}(n) + \dfrac{1-\nu}{1-\nu^{n+1}} \boldsymbol{u}_{\text{RR}}(n+1) \boldsymbol{u}_{\text{RR}}^H(n+1)$
2	$\boldsymbol{r}_1 = \boldsymbol{b}_{\text{RR}} - \tau \boldsymbol{p}(n) - \boldsymbol{R}_{\text{RR}}(n+1) \boldsymbol{w}_0(n)$
3	$\mu_1 = \dfrac{\boldsymbol{r}_1^H \boldsymbol{r}_1}{\boldsymbol{r}_1^H \boldsymbol{R}_{\text{RR}}(n+1) \boldsymbol{r}_1}$
4	$\boldsymbol{w}_0(n+1) = \boldsymbol{w}_0(n) + \mu_1 \boldsymbol{r}_1$
5	$\boldsymbol{r}_2 = \boldsymbol{w}_0(n+1) - \boldsymbol{R}(n+1) \boldsymbol{p}(n)$
6	$\mu_2 = \dfrac{\boldsymbol{r}_2^H \boldsymbol{r}_2}{\boldsymbol{r}_2^H \boldsymbol{R}_{\text{RR}}(n+1) \boldsymbol{r}_2}$
7	$\boldsymbol{p}(n+1) = \boldsymbol{p}(n) + \mu_2 \boldsymbol{r}_2$
8	$\boldsymbol{\varpi}(n+1) = \dfrac{\boldsymbol{w}_0(n+1)}{\boldsymbol{b}_{\text{RR}}^H \boldsymbol{w}_0(n+1)}$
	end

The general-linear-combination-based robust Capon beamformer (GLC)

The GLC is presented in Table 28. Note that it updates $\boldsymbol{R}_{\text{RR}}$ using a window with N_{w} entries. The method requires the solution of a linear system of equations (see step 8), which would require $\boldsymbol{\mathcal{O}}(M^3)$ operations in a approach using LU decomposition.

Table 28: General-linear-combination-based robust Capon beamformer

Input: $\mathbf{b}_{\text{RR}}$, $\boldsymbol{u}_{\text{RR}}$, ν, M, N_{w} **Output:** $\boldsymbol{\varpi}(n)$	
Initialize: $\mathbf{R}_{\text{RR}}(0) = \xi\mathbf{I}, \boldsymbol{w}_0(0) = \mathbf{0}$, $\rho_1 = \mathbf{0}$	
	for $n = 1, 2, \cdots$
1	$\boldsymbol{R}_{\text{RR}}(n+1) = \boldsymbol{R}_{\text{RR}}(n) + \frac{1}{N_{\text{w}}}(\boldsymbol{u}_{\text{RR}}(n+1)\boldsymbol{u}_{\text{RR}}^H(n+1) - \boldsymbol{u}_{\text{RR}}(n-N_{\text{w}}+1)\boldsymbol{u}_{\text{RR}}^H(n-N_{\text{w}}+1)$
2	$\rho_1 = \rho_1 + \|\|\boldsymbol{u}_{\text{RR}}(n+1)\|\|_2^4 - \|\|\boldsymbol{u}_{\text{RR}}(n-N_{\text{w}}+1)\|\|_2^4$
3	$\rho = \frac{1}{N_{\text{w}}^2}\rho_1 - \frac{1}{N_{\text{w}}}\|\|\boldsymbol{R}_{\text{RR}}(n+1)\|\|_2^2$
4	$v = \text{trace}(\boldsymbol{R}_{\text{RR}}(n+1))/M$
5	$a = \min(v\rho/\|\|\boldsymbol{R}(n+1) - v\boldsymbol{T}\|\|_2^2, v)$
6	$b = 1 - a/v$
7	$\boldsymbol{R}_{\text{GLC}} = a\boldsymbol{I} + b\boldsymbol{R}_{\text{RR}}(n+1)$
8	$\boldsymbol{R}_{\text{GLC}}\boldsymbol{w}(n+1) = \boldsymbol{b}_{\text{RR}} \Rightarrow \boldsymbol{w}(n+1)$
9	$\boldsymbol{\varpi}(n+1) = \frac{\boldsymbol{w}(n+1)}{\boldsymbol{b}_{\text{RR}}^H\boldsymbol{w}(n+1)}$
	end

6 POSSIBILITIES TO CONTINUE THIS WORK

In this chapter, we present an overview of possible research lines to continue this work. We present ideas which can lead to new algorithms and some possibilities to better exploit the techniques proposed in this text.

6.1 Development of an analysis for the DCD algorithm

The computational complexity gain of the DCD method is based on the iterative solution of a modified normal system of equations, where the coefficients are bit-wise updated. Since the update is nonlinear, the performance analysis of the algorithm is challenging and still unsolved.

A first attempt to study the convergence of DCD-based algorithms was proposed in [91], where a second-order analysis of the DCD-RLS algorithm was developed, assuming some simplifications. In that case, the instantaneous correlation quantities were replaced by their deterministic expressions in the normal equations, and DCD was applied to solve the resulting system of equations. That approach provided some insights on the algorithm's behavior, but there is still a misfit between the model and the simulations. Additionally, the approach was not used to directly study the DCD, since the non-linearity of the DCD iterations make it difficult to apply traditional tools used to study iterative methods.

Another consideration in the study of DCD is the computational complexity of the technique. Papers such as [25, 28, 92] assume that the DCD's complexity is linear with the regressor length N, since it appears multiplied by N_u (generally chosen $N_u \ll N$). However, one can question if N_u is related to N somehow, that is, if N_u must be increased as N increases to keep some performance criterion constant. This problem also remains an open question, and the development of an accurate analysis of the DCD algorithm could help to answer this question, providing better support to apply this method.

6.2 Using DCD iterations in other algorithms

Since DCD iterations can be used to reduce the complexity of complex-valued algorithms, it is natural to imagine the extension of this approach to quaternion techniques. The WL quaternion RLS [15], for instance, presents high complexity due to the quaternion elements and also due to the WL approach. The development of a quaternion algorithm using DCD iterations could lead to a substantial cost reduction. If an algorithm using both DCD iterations and our RC approach, one could obtain a quaternion WL algorithm of very low complexity.

Another possibility is the use of DCD iterations in an approach similar to that of [93]. In that work, a series-cooperative structure was proposed to combine two adaptive filters, in order to achieve better MSE performance than a parallel combination [94] of algorithms. In that paper, two LMS filters were used in the cooperative structure. If a series structure of a DCD-RLS filter and an LMS filter was considered, one could expect a faster-converging and lower-complexity combination of filters, very attractive for hardware implementation.

Given the flexibility of the DCD, this method can be extended to other algorithms which solve the least-squares problem. Other possibilities can be proposed, leading to new low-complexity algorithms.

6.3 Distributed estimation with adaptive filters applying DCD iterations

In distributed estimation techniques [95, 96], it was shown that the communication between the nodes provides estimation gains and robustness to the global estimation. However, the information exchange between the nodes leads to an energy consumption problem, since data must be transmitted to every node. In this case, DCD iterations can be an alternative to save energy.

In [31], a DCD-based diffusion RLS was proposed to reduce the computational cost and also make the distributed technique robust against numerical instability. Simulation results have shown that the DCD-based technique can perform as well as the original algorithm but, as it happens to techniques which apply DCD iterations, there is a trade-off between complexity and performance. Since distributed techniques using DCD iterations are just starting to be studied, there is plenty of opportunities to research.

Some interesting questions that arise – and that were not addressed in [31] – can be listed:

1. How the maximum number of DCD iterations N_u must be designed to achieve a pre-defined estimation error, and what is the impact in the network if we choose a small N_u, or a different value of N_u for each adaptive filter in the network.

2. How to choose M_b to save energy and still guarantee satisfactory performance of the distributed estimation, since an increase of the number of bits to be transmitted implies in the increase of the number of transmissions and energy consumption.

3. How many bits must be used for internal storage within a node and for communication?

4. How N_u and M_b can be related to provide the best trade-off energy consumption-estimation gain.

Note that the answer to these questions can improve the knowledge of DCD and also help the design of new approaches, mixing DCD and distributed algorithms.

6.4 Advances with quaternion algorithms

In the research with quaternion algorithms, one possibility to continue the work presented in this dissertation is the development of simpler expressions to the second-order analysis obtained for QLMS-based algorithms in general. Another possibility would be the development of an analysis using quaternion algebra, which could be difficult to obtain, since this algebra still lacks some basic tools, such as ways to compute the eigenvalues.

As already proposed in Section 6.2, we can also modify DCD for the QRLS (or WL-QRLS) algorithm. Note that we can use an approach similar to that used in Chapter 2 to modify the WL-QRLS, and then extend the DCD to the quaternion domain to obtain a low-complexity WL algorithm.

6.5 Research on the beamforming techniques using the homotopy algorithm

The beamforming techniques proposed in the Chapter 5 can be easily extended to WL algorithms, for which we could also apply the extended real regressor vector used by

the RC-WL-LMS algorithm, as it was presented in Chapter 2.

Considering the techniques proposed in the text, some points still need more research to clarify the design of the parameters and also the behavior of the techniques:

1. The re-weighting used by C-ARH algorithm was selected based on simulation results. It would be interesting to study other re-weighting options and also try to obtain an analytical approach to design $\tilde{\boldsymbol{h}}$.

2. The values used for the stopping criteria were also selected based on extensive simulations. An analytical approach – or at least some bounds to select these parameters – would help the designer to access the best properties of the algorithms for a given problem.

3. A simulation study of the C-ARH-based algorithms is also required to understand their behavior for other scenarios (for instance, for non-Gaussian noise or when the noise is spatially or temporally correlated).

7 FINAL CONSIDERATIONS

In this text, we concentrated on the development and on the analysis of new low-cost techniques for adaptive filtering. In the first part of the dissertation, we proposed WL techniques for which the complexity reduction was achieved excluding the redundant second-order information from the autocorrelation matrix. In the second part, we developed new techniques for WL processing and for beamforming, which applied DCD iterations to reduce the computational cost.

In Chapter 2, we exploited WL estimation concepts to develop low-cost complex-valued WL adaptive filters. We replaced the WL data vector of the WL-LMS and WL-RLS algorithms by a real-valued vector containing the same information. Using this approach, redundant second-order statistics in the autocorrelation matrix are avoided, and the computational complexity is reduced, since many complex-complex operations are substituted by real-complex operations, less costly. Simulations comparing the proposed and the standard WL algorithms showed that they present the same MSE performance. The computational complexity evaluation showed that the RC-WL-LMS and the SL-LMS algorithms have the same computational cost, which is about one-fourth the cost to compute WL-LMS. RC-WL-RLS also has a complexity similar to that obtained with the SL-RLS algorithm, which is less than one-third of the cost of WL-RLS.

The approach used for complex-valued WL estimation in Chapter 2 was extended to the quaternion domain in Chapter 3. We showed that the quaternion autocorrelation matrix has redundant terms that can be avoided, and we exploited it to propose RC algorithms with real input vector. A general description to quaternion gradients was applied to study QLMS-based techniques, and it was shown that different gradients may lead to the same quaternion adaptive filter. This general approach helped us to show that gradients similar to the i-gradient lead to the fastest-converging WL algorithms if there is correlation only between two quaternion elements, or when the input is real and given by the concatenation of the entries of $\boldsymbol{q}$. Mean and a mean-square analyses suitable for any QLMS-based algorithm were obtained and particularized for WL algorithms using real

regressor vector. We derived simple equations to compute the steady-state EMSE and MSD, and a model to study the convergence of WL techniques. Bounds to choose the step size of real-input WL-QLMS algorithms which guarantee the convergence in the mean and in the variance were also obtained. We proposed the RC-WL-iQLMS algorithm, and showed that it is the fastest-converging WL-QLMS algorithm with real regressor vector. The technique is equivalent to WL-iQLMS, but it is less costly to compute. It also corresponds to the 4-Ch-LMS algorithm, written in the quaternion domain.

In Chapter 4, the RC-WL-RLS algorithm was revisited and modified to apply DCD iterations. We proposed the DCD-WL-RLS algorithm as a low-cost method ($\boldsymbol{\mathcal{O}}(N)$) for cases where the real and the imaginary entries of the input vector are tap delay-lines. This fact was exploited to obtain a low-cost update of the estimated WL autocorrelation matrix. DCD iterations were applied to further reduce the complexity, which was possible due to a modification on the equations of the RC-WL-RLS algorithm. We showed that DCD computes the update at very low cost, since a few number of DCD iterations (N_u) can be used for this. The DCD-WL-RLS algorithm was compared to other WL-RLS algorithms, and we showed that it can achieve an MSE performance similar to that obtained with RC-WL-RLS, but at reduced cost.

In Chapter 5, we extended the ARH algorithm to the complex field, and we proposed low-cost beamforming algorithms based on the C-ARH method. We first developed the C-ARH and multi-candidate C-ARH algorithms, which were later modified to obtain their iterative versions, less costly. The iterative algorithms exploit the solution of the previous snapshot as an initial condition for the actual iteration, helping to reduce the number of homotopy iterations and thus reducing the computational complexity. Using DCD iterations, further complexity reduction was obtained with the DCD-It-C-ARH and DCD-It-MC-C-ARH algorithms. We used simulations to show that the DCD-based techniques are $\boldsymbol{\mathcal{O}}(M^2)$ if the number of interferers does not grow with the number of sensors. When the number of interferers increases linearly with the number of sensors, the techniques are $\boldsymbol{\mathcal{O}}(M^5)$, but for a range of values of M, they are still less costly than the $\boldsymbol{\mathcal{O}}(M^2)$ VL technique. Simulations also have shown that the C-ARH-based algorithms are robust against sensor failure. We compared the SINR performance of the proposed algorithms with other techniques from the literature: the RCB, DL and GLC, $\boldsymbol{\mathcal{O}}(M^3)$ techniques, and the VL algorithm. The trade-off between SINR performance and computational complexity favors our DCD-based techniques.

REFERENCES

1 PICINBONO, B.; CHEVALIER, P. Widely linear estimation with complex data. *IEEE Transaction on Signal Processing*, v. 43, n. 8, p. 2030 –2033, Aug. 1995. ISSN 1053-587X.

2 ADALI, T.; LI, H.; ALOYSIUS, R. On properties of the widely linear MSE filter and its LMS implementation. In: *43rd Annual Conference on Information Sciences and Systems (CISS)*. [S.l.: s.n.], 2009. p. 876 –881.

3 KANTOR, I.; SOLODOVNIKOV, A.; SHENITZER, A. *Hypercomplex numbers: an elementary introduction to algebras*. [S.l.]: Springer-Verlag, 1989. ISBN 0387969802.

4 SCHREIER, P.; SCHARF, L. Second-order analysis of improper complex random vectors and processes. *IEEE Transactions on Signal Processing*, v. 51, n. 3, p. 714–725, 2003.

5 PAPOULIS, A. *Probability & statistics*. [S.l.]: Prentice-Hall, 1990.

6 SAYED, A. H. *Fundamentals of adaptive filtering*. [S.l.]: John Wiley & Sons, 2003.

7 TREES, H. V. *Optimum Array Processing: Part IV of Detection, Estimation and Modulation Theory*. [S.l.]: Wiley, 2002.

8 ADALI, T.; SCHREIER, P.; SCHARF, L. Complex-valued signal processing: The proper way to deal with impropriety. *IEEE Transactions on Signal Processing*, v. 59, n. 11, p. 5101–5125, Nov 2011. ISSN 1053-587X.

9 OLLILA, E. On the circularity of a complex random variable. *IEEE Signal Processing Letters*, v. 15, p. 841–844, 2008. ISSN 1070-9908.

10 ADALI, T.; LI, H. *Complex-valued adaptive signal processing*. [S.l.]: Hoboken, NJ: Wiley, 2010.

11 SCHREIER, P.; SCHARF, L. *Statistical signal processing of complex-valued data: the theory of improper and noncircular signals*. [S.l.]: Cambridge University Press, 2010.

12 TOOK, C.; MANDIC, D. Fusion of heterogeneous data sources: A quaternionic approach. In: *IEEE Workshop on Machine Learning for Signal Processing*. [S.l.: s.n.], 2008. p. 456 –461. ISSN 1551-2541.

13 UJANG, C. et al. A split quaternion nonlinear adaptive filter. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2009. p. 1745 –1748. ISSN 1520-6149.

14 TOOK, C.; MANDIC, D. The Quaternion LMS Algorithm for Adaptive Filtering of Hypercomplex Processes. *IEEE Transactions on Signal Processing*, IEEE, v. 57, n. 4, p. 1316–1327, 2009. ISSN 1053-587X.

15 JAHANCHAHI, C.; TOOK, C.; MANDIC, D. The widely linear quaternion recursive least squares filter. In: *2nd International Workshop on Cognitive Information Processing (CIP)*. [S.l.: s.n.], 2010. p. 87 –92.

16 UJANG, B. C.; TOOK, C.; MANDIC, D. Quaternion-valued nonlinear adaptive filtering. *IEEE Transactions on Neural Networks*, v. 22, n. 8, p. 1193 –1206, aug. 2011. ISSN 1045-9227.

17 TOOK, C.; MANDIC, D. Quaternion-valued stochastic gradient-based adaptive IIR filtering. *IEEE Transactions on Signal Processing*, v. 58, n. 7, p. 3895 –3901, July 2010. ISSN 1053-587X.

18 ALMEIDA NETO, F.; NASCIMENTO, V. A novel reduced-complexity widely linear QLMS algorithm. In: *IEEE Statistical Signal Processing Workshop (SSP)*. [S.l.: s.n.], 2011. p. 81 –84. ISSN pending.

19 XIA, Y. et al. The HC calculus, quaternion derivatives and Caylay-Hamilton form of quaternion adaptive filters and learning systems. In: *International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2014. p. 3395–3401.

20 SUH, Y. S. Orientation estimation using a quaternion-based indirect Kalman filter with adaptive estimation of external acceleration. *IEEE Transactions on Instrumentation and Measurement*, v. 59, n. 12, p. 3296–3305, Dec 2010. ISSN 0018-9456.

21 CHOUKROUN, D.; BAR-ITZHACK, I.; OSHMAN, Y. Novel quaternion Kalman filter. *IEEE Transactions on Aerospace and Electronic Systems*, v. 42, n. 1, p. 174–190, Jan 2006. ISSN 0018-9251.

22 JIANG, M. et al. Frequency-domain quaternion-valued adaptive filtering and its application to wind profile prediction. In: *IEEE Region 10 Conference*. [S.l.: s.n.], 2013. p. 1–5. ISSN 2159-3442.

23 ZAKHAROV, Y.; TOZER, T. Multiplication-free iterative algorithm for LS problem. *Electronics Letters*, v. 40, n. 9, p. 567 – 569, April 2004. ISSN 0013-5194.

24 NASCIMENTO, V.; SILVA, M. Adaptive filters. In: CHELLAPPA, R.; THEODORIDIS, S. (Ed.). *Academic Press Library in Signal Processing:*. [S.l.]: Chennai: Academic Press, 2014. v. 1, Signal Processing Theory and Machine Learning, p. 619—761. ISBN 978-0-12-396502-8.

25 ZAKHAROV, Y.; WHITE, G.; LIU, J. Low-complexity RLS algorithms using dichotomous coordinate descent iterations. *IEEE Transaction on Signal Processing*, v. 56, n. 7, p. 3150 –3161, July 2008. ISSN 1053-587X.

26 LIU, J.; ZAKHAROV, Y. Low complexity dynamically regularised RLS algorithm. *Electronics Letters*, v. 44, n. 14, p. 886 –885, 3 2008. ISSN 0013-5194.

27 ZAKHAROV, Y. Low-complexity implementation of the affine projection algorithm. *IEEE Signal Processing Letters*, v. 15, p. 557–560, 2008. ISSN 1070-9908.

28 ZAKHAROV, Y.; WHITE, G.; LIU, J. Fast RLS algorithm using dichotomous coordinate descent iterations. In: *Conference on Signals, Systems and Computers (Asilomar)*. [S.l.: s.n.], 2007. p. 431 –435. ISSN 1058-6393.

29 ALMEIDA NETO, F.; NASCIMENTO, V.; ZAKHAROV, Y. Low-complexity widely linear RLS filter using DCD iterations. In: *XXX Simpósio Brasileiro de Telecomunicações*. [S.l.: s.n.], 2012.

30 ZAKHAROV, Y.; NASCIMENTO, V. Orthogonal matching pursuit with DCD iterations. *Electronics Letters*, v. 49, n. 4, p. 295–297, 2013. ISSN 0013-5194.

31 ARABLOUEI, R.; DOGANCAY, K.; WERNER, S. Reduced-complexity distributed least-squares estimation over adaptive networks. In: *IEEE 14th Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. [S.l.: s.n.], 2013. p. 150–154. ISSN 1948-3244.

32 SAYED, A. *Adaptive filters*. [S.l.]: Wiley-IEEE Press, 2008.

33 LI, J.; STOICA, P.; WANG, Z. On robust capon beamforming and diagonal loading. *IEEE Trans. on Signal Processing*, v. 51, n. 7, p. 1702–1715, July 2003. ISSN 1053-587X.

34 VOROBYOV, S.; GERSHMAN, A.; LUO, Z.-Q. Robust adaptive beamforming using worst-case performance optimization: a solution to the signal mismatch problem. *IEEE Trans. on Signal Processing*, v. 51, n. 2, p. 313–324, Feb 2003. ISSN 1053-587X.

35 LI, J.; STOICA, P. *Robust adaptive beamforming*. [S.l.]: Wiley Online Library, 2006.

36 QI, C.; WU, L.; WANG, X. Underwater acoustic channel estimation via complex homotopy. In: *IEEE Int. Conf. on Commun. (ICC)*. [S.l.: s.n.], 2012. p. 3821–3825. ISSN 1550-3607.

37 ASIF, M.; ROMBERG, J. Fast and accurate algorithms for re-weighted L1-norm minimization. *IEEE Trans. Signal Processing*, v. 61, n. 23, p. 5905–5916, Dec 2013. ISSN 1053-587X.

38 ALMEIDA NETO, F.; NASCIMENTO, V.; SILVA, M. Reduced-complexity widely linear adaptive estimation. In: *7th International Symposium on Wireless Communication Systems*. [S.l.: s.n.], 2010. p. 399–403. ISSN 2154-0217.

39 ALMEIDA NETO, F.; NASCIMENTO, V. Second-order analysis of the RC-WL-QLMS algorithm. In: *Proceedings 9th International Symposium on Wireless Communication Systems*. [S.l.: s.n.], 2012.

40 ALMEIDA NETO, F. et al. Adaptive re-weighting homotopy for sparse beamforming. In: *Proceedings of the 22nd European Signal Processing Conference (EUSIPCO)*. [S.l.: s.n.], 2014. p. 1–5.

41 ALMEIDA NETO, F. et al. Adaptive re-weighting Homotopy Algorithms Applied to Beamforming. *Accepted for publication in the IEEE Trans. on Aerospace and Electronic Systems*, 2015.

42 JAHANCHAHI, C.; TOOK, C. C.; MANDIC, D. P. On gradient calculation in quaternion adaptive filtering. In: *IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2012. ISSN pending.

43 YOU, Q.; JIANYUN, Z.; XIN'AN, Z. A Widely-Linear LMS Algorithm for Adaptive Beamformer. In: *Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications, 2007 International Symposium on*. [S.l.: s.n.], 2007. p. 1060–1063.

44 DOUGLAS, S. Widely-linear recursive least-squares algorithm for adaptive beamforming. In: *Proceedings of the 2009 IEEE International Conference on Acoustics, Speech and Signal Processing-Volume 00*. [S.l.: s.n.], 2009. p. 2041–2044.

45 HAYKIN, S. *Adaptive Filter Theory, 4ed.* [S.l.]: Prentice Hall, 2002.

46 MEYER, C. D. *Matrix Analysis and Applied Linear Algebra*. Philadelphia, USA: SIAM, 2000.

47 HANSON, A. *Visualizing quaternions*. [S.l.]: Morgan Kaufmann, 2006. ISBN 978-0-12-088400-1.

48 WEN, J.-Y.; KREUTZ-DELGADO, K. The attitude control problem. *IEEE Trans. Autom. Control*, v. 36, n. 10, p. 1148–1162, Oct 1991. ISSN 0018-9286.

49 LIZARRALDE, F.; WEN, J. Attitude control without angular velocity measurement: a passivity approach. *IEEE Trans. Autom. Control*, v. 41, n. 3, p. 468–472, Mar 1996. ISSN 0018-9286.

50 BULOW, T.; SOMMER, G. Hypercomplex signals-a novel extension of the analytic signal to the multidimensional case. *IEEE Transactions on Signal Processing*, v. 49, n. 11, p. 2844 –2852, Nov 2001. ISSN 1053-587X.

51 PEI, S.-C.; CHENG, C.-M. Color image processing by using binary quaternion-moment-preserving thresholding technique. *IEEE Transactions on Image Processing*, v. 8, n. 5, p. 614 –628, may 1999. ISSN 1057-7149.

52 BIHAN, N. L.; SANGWINE, S. Quaternion principal component analysis of color images. In: *Proceedings of the 2003 Conference on Image Processing*. [S.l.: s.n.], 2003. v. 1, p. I–809–12 vol.1. ISSN 1522-4880.

53 BAS, P.; BIHAN, N. L.; CHASSERY, J.-M. Color image watermarking using quaternion fourier transform. In: *Proceedings of the 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing*. [S.l.: s.n.], 2003. v. 3, p. III–521–4 vol.3. ISSN 1520-6149.

54 ELL, T. A.; BIHAN, N. L.; SANGWINE, S. J. *Quaternion Fourier Transforms for Signal and Image Processing*. [S.l.]: John Wiley & Sons, 2014.

55 CHANG, H.-T. et al. DNA sequence representation and comparison based on quaternion number system. *International Journal of Advanced Computer Science and Applications*, v. 3, n. 11, p. 39–46, 2012.

56 TAO, J.-W.; CHANG, W.-X. A novel combined beamformer based on hypercomplex processes. *IEEE Trans. Aerosp. Electron. Syst.*, v. 49, n. 2, p. 1276–1289, April 2013. ISSN 0018-9251.

57 JAHANCHAHI, C.; TOOK, C.; MANDIC, D. On HR calculus, quaternion valued stochastic gradient, and adaptive three dimensional wind forecasting. In: *International Joint Conference on Neural Networks (IJCNN)*. [S.l.: s.n.], 2010. p. 1 –5. ISSN 1098-7576.

58 TOOK, C. C.; MANDIC, D. A quaternion widely linear adaptive filter. *IEEE Transactions on Signal Processing*, v. 58, n. 8, p. 4427–4431, 2010. ISSN 1053-587X.

59 MANDIC, D.; JAHANCHAHI, C.; TOOK, C. A quaternion gradient operator and its applications. *IEEE Signal Processing Letters*, v. 18, n. 1, p. 47 –50, jan. 2011. ISSN 1070-9908.

60 BARTHELEMY, Q.; LARUE, A.; MARS, J. About QLMS derivations. *IEEE Signal Process. Lett.*, v. 21, n. 2, p. 240–243, Feb 2014. ISSN 1070-9908.

61 VÍA, J.; RAMIREZ, D.; SANTAMARÍA, I. Properness and widely linear processing of quaternion random vectors. *IEEE Transactions on Information Theory*, v. 56, n. 7, p. 3502 –3515, July 2010.

62 TOOK, C.; JAHANCHAHI, C.; MANDIC, D. A unifying framework for the analysis of quaternion valued adaptive filters. In: *Conference on Signals, Systems and Computers (ASILOMAR)*. [S.l.: s.n.], 2011. p. 1771 –1774. ISSN 1058-6393.

63 HORN, R. A.; JOHNSON, C. R. *Topics in Matrix Analysis*. Cambridge, MA: Cambridge University Press, 1991.

64 BRAATZ, R.; MORARI, M. Minimizing the Euclidean condition number. *SIAM Journal on Control and Optimization*, v. 32, n. 6, p. 1763–1768, 1994. Disponível em: ⟨http://epubs.siam.org/doi/abs/10.1137/S0363012992238680⟩.

65 DINIZ, P. *Adaptive filtering: algorithms and practical implementation*. [S.l.]: Springer, 2004.

66 JR, J. A. A. *QRD-RLS adaptive filtering*. [S.l.]: Springer, 2009.

67 LIU, J. et al. An FPGA-based MVDR beamformer using dichotomous coordinate descent iterations. In: *IEEE International Conference on Communications*. [S.l.: s.n.], 2007. p. 2551–2556.

68 ZAKHAROV, Y.; NASCIMENTO, V. Homotopy algorithm using dichotomous coordinate descent iterations for sparse recovery. In: *Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*. [S.l.: s.n.], 2012. p. 820–824. ISSN 1058-6393.

69 DONOHO, D. Compressed sensing. *IEEE Transactions on Information Theory*, v. 52, n. 4, p. 1289 –1306, april 2006. ISSN 0018-9448.

70 LIU, J.; ZAKHAROV, Y.; WEAVER, B. Architecture and FPGA design of dichotomous coordinate descent algorithms. *IEEE Trans. on Circuits and Systems I: Regular Papers*, v. 56, n. 11, p. 2425–2438, 2009. ISSN 1549-8328.

71 YANG, Z.; LAMARE, R. de; LI, X. L1-regularized STAP algorithms with a generalized sidelobe canceler architecture for airborne radar. *IEEE Trans. Signal Processing*, v. 60, n. 2, p. 674–686, 2012. ISSN 1053-587X.

72 YANG, Z.; LAMARE, R. de; LI, X. Sparsity-aware space-time adaptive processing algorithms with L1-norm regularisation for airborne radar. *IET Signal Processing*, v. 6, n. 5, p. 413–423, 2012. ISSN 1751-9675.

73 YANG, Z.; LAMARE, R. de; LI, X. L1-regularized STAP algorithm with a generalized sidelobe canceler architecture for airborne radar. In: *IEEE Statistical Signal Process. Workshop (SSP)*. [S.l.: s.n.], 2011. p. 329–332. ISSN pending.

74 GUERCI, J.; BERGIN, J. Principal components, covariance matrix tapers, and the subspace leakage problem. *IEEE Trans. on Aerospace and Electronic Systems*, v. 38, n. 1, p. 152–162, Jan 2002. ISSN 0018-9251.

75 BUCKLEW, J.; SETHARES, W. Convergence of a class of decentralized beamforming algorithms. *IEEE Trans. on Signal Processing*, v. 56, n. 6, p. 2280–2288, June 2008. ISSN 1053-587X.

76 LIN, C.; VEERAVALLI, V.; MEYN, S. A random search framework for convergence analysis of distributed beamforming with feedback. *IEEE Trans. on Information Theory*, v. 56, n. 12, p. 6133–6141, Dec 2010.

77 SONG, S. et al. Exploiting negative feedback information for one-bit feedback beamforming algorithm. *IEEE Trans. on Wireless Communications*, v. 11, n. 2, p. 516–525, Feb. 2012.

78 COX, H.; ZESKIND, R.; OWEN, M. Robust Adaptive Beamforming. *IEEE Trans. on Acoustics, Speech and Signal Processing*, v. 35, n. 10, p. 1365–1376, Oct 1987. ISSN 0096-3518.

79 DU, L.; LI, J.; STOICA, P. Fully automatic computation of diagonal loading levels for robust adaptive beamforming. *IEEE Trans.on Aerospace and Electronic Systems*, v. 46, n. 1, p. 449–458, Jan 2010. ISSN 0018-9251.

80 GU, J.; WOLFE, P. Robust adaptive beamforming using variable loading. In: *Fourth IEEE Workshop onSenso r Array and Multichannel Processing*. [S.l.: s.n.], 2006. p. 1–5.

81 DU, L.; LI, J.; STOICA, P. Fully automatic computation of diagonal loading levels for robust adaptive beamforming. *IEEE Transactions on Aerospace and Electronic Systems*, v. 46, n. 1, p. 449–458, Jan 2010. ISSN 0018-9251.

82 MELVIN, W. A STAP overview. *IEEE Aerospace and Electronic Systems Magazine*, v. 19, n. 1, p. 19–35, Jan 2004. ISSN 0885-8985.

83 GESBERT, D. et al. Shifting the MIMO paradigm. *IEEE Signal Processing Magazine*, v. 24, n. 5, p. 36–46, Sept 2007. ISSN 1053-5888.

84 KAY, S. *Fundamentals of Statistical Signal Processing: Detection Theory*. [S.l.]: Prentice Hall, 1998.

85 DONOHO, D.; TSAIG, Y. Fast solution of L1-norm minimization problems when the solution may be sparse. *IEEE Trans. Inform. Theory*, v. 54, n. 11, p. 4789–4812, 2008. ISSN 0018-9448.

86 YANG, J.; ZHANG, Y. Alternating direction algorithms for L1-problems in compressive sensing. *SIAM Journal on Scientific Computing*, SIAM, v. 33, n. 1, p. 250–278, 2011.

87 WRIGHT, S.; NOWAK, R.; FIGUEIREDO, M. Sparse reconstruction by separable approximation. *IEEE Trans. Signal Processing*, IEEE, v. 57, n. 7, p. 2479–2493, 2009.

88 BERG, E. V. D.; FRIEDLANDER, M. Probing the Pareto frontier for basis pursuit solutions. *SIAM Journal on Scientific Computing*, SIAM, v. 31, n. 2, p. 890–912, 2008.

89 KAZANCI, O.; KROLIK, J. Single-snapshot beamformer performance using large arrays with faulty sensors. In: *IEEE Workshop on Sensor Array and Multichannel Processing*. [S.l.: s.n.], 2006. p. 6–10.

90 QIN, S.; LI, W. Detection and identification of faulty sensors with maximized sensitivity. In: *Proceedings of the American Control Conference*. [S.l.: s.n.], 1999. v. 1, p. 613–617 vol.1. ISSN 0743-1619.

91 CHEN, T.; ZAKHAROV, Y. Convergence analysis of RLS-DCD algorithm. In: *IEEE/SP 15th Workshop on Statistical Signal Processing (SSP)*. [S.l.: s.n.], 2009. p. 157 –160.

92 LIU, J.; QUART, Z.; ZAKHAROV, Y. Parallel FPGA implementation of DCD algorithm. In: *15th International Conference on Digital Signal Processing*. [S.l.: s.n.], 2007. p. 331 –334.

93 LOPES, W.; LOPES, C. Incremental-cooperative strategies in combination of adaptive filters. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. [S.l.: s.n.], 2011. p. 4132 –4135. ISSN 1520-6149.

94 ARENAS-GARCIA, J.; FIGUEIRAS-VIDAL, A.; SAYED, A. Mean-square performance of a convex combination of two adaptive filters. *IEEE Transactions on Signal Processing*, v. 54, n. 3, p. 1078 – 1090, March 2006. ISSN 1053-587X.

95 LOPES, C.; SAYED, A. Incremental adaptive strategies over distributed networks. *IEEE Transactions on Signal Processing*, v. 55, n. 8, p. 4064 –4077, aug. 2007. ISSN 1053-587X.

96 LOPES, C.; SAYED, A. Diffusion least-mean squares over adaptive networks: Formulation and performance analysis. *IEEE Transactions on Signal Processing*, v. 56, n. 7, p. 3122 –3136, july 2008. ISSN 1053-587X.