TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Alexandru Onose

Greedy Adaptive Algorithms for Sparse Representations

Julkaisu 1211 • Publication 1211

Tampere 2014

Tampereen teknillinen yliopisto. Julkaisu 1211
Tampere University of Technology. Publication 1211

Alexandru Onose

Greedy Adaptive Algorithms for Sparse Representations

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Tietotalo Building, Auditorium TB219, at Tampere University of Technology, on the 13th of May 2014, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2014

Supervisors:

Dr. Bogdan Dumitrescu,
Department of Signal Processing,
Faculty of Computing and Electrical Engineering,
Tampere University of Technology,
Tampere, Finland.

Prof. Ioan Tăbuş (Custos),
Department of Signal Processing,
Faculty of Computing and Electrical Engineering,
Tampere University of Technology,
Tampere, Finland.

Pre-examiner:

Dr. Radu Bîlcu,
Nokia Research Center,
Tampere, Finland.

Pre-examiner and opponent:

Dr. Stefan Werner,
Department of Signal Processing and Acoustics,
School of Electrical Engineering,
Aalto University,
Aalto, Finland.

Opponent:

Dr. Kristiaan Pelckmans,
Division of Systems and Control,
Department of Information Technology,
Uppsala University,
Uppsala, Sweden.

ISBN 978-952-15-3282-5 (printed)
ISBN 978-952-15-3308-2 (PDF)
ISSN 1459-2045

Abstract

A vector or matrix is said to be sparse if the number of non-zero elements is significantly smaller than the number of zero elements. In estimation theory the vectors of model parameters can be known in advance to have a sparse structure, and solving an estimation problem taking into account this constraint can improve substantially the accuracy of the solution. The theory of sparse models has advanced significantly in recent years providing many results that can guarantee certain properties of the sparse solutions. These performance guarantees can be very powerful in applications and they have no correspondent in the estimation theory for non-sparse models.

Model sparsity is an inherent characteristic of many applications (image compressing, wireless channel estimation, direction of arrival) in signal processing and other related areas. Due to the continuous technological advances that allow faster numerical computations, optimization problems, too complex to be solved in the past, are now able to provide better solutions by considering also sparsity constraints. However, an exhaustive search to finding sparse solutions generally requires a combinatorial search for the correct support, a very limiting factor due to the huge numerical complexity. This motivated a growing interest towards developing batch sparsity aware algorithms in the past twenty years.

More recently, the main goal for the continuous research related to sparsity is the quest for faster, less computational intensive, adaptive methods able to recursively update the solution. In this thesis we present several such algorithms. They are greedy in nature and minimize the least squares criterion under the constraint that the solution is sparse. Similarly to other greedy sparse methods, two main steps are performed once new data are available: update the sparse support by changing the positions that contribute to the solution; compute the coefficients towards the minimization of the least squares criterion restricted to the current support. Two classes of adaptive algorithms were proposed.

The first is derived from the batch matching pursuit algorithm. It uses a coordinate descent approach to update the solution, each coordinate being selected by a criterion similar to the one used by matching pursuit. We devised two algorithms that use a cyclic update strategy to improve the solution at each time instant. Since the solution support and coefficient values are assumed to vary slowly, a faster and better performing approach is later proposed by spreading the coordinate descent update in time. It was also adapted to work in a distributed

setup in which different nodes communicate with their neighbors to improve their local solution towards a global optimum.

The second direction can be linked to the batch orthogonal least squares. The algorithms maintain a partial QR decomposition with pivoting and require a permutation based support selection strategy to ensure a low complexity while allowing the tracking of slow variations in the support. Two versions of the algorithm were proposed. They allow past data to be forgotten by using an exponential or a sliding window, respectively. The former was modified to improve the solution in a structured sparsity case, when the solution is group sparse.

We also propose mechanisms for estimating online the sparsity level. They are based on information theoretic criteria, namely the predictive least squares and the Bayesian information criterion.

The main contributions are the development of the adaptive greedy algorithms and the use of the information theoretic criteria enabling the algorithms to behave robustly. The algorithms have good performance, require limited prior information and are computationally efficient. Generally, the configuration parameters, if they exist, can be easily chosen as a tradeoff between the stationary error and the convergence speed.

Preface

The research results that coalesced to produce this thesis have been developed during the period 2010-2013 at the Signal Processing Department of Tampere University of Technology. So far it has been an interesting and motivating *journey*, with the frustration and joy that technical research always seems to bring. Having just submitted new promising results to a conference and, with hopes of a journal article to follow, the *journey* is not over. I hope that the future, and wherever the *journey* might take me, will be at least as interesting as these last four years have been.

Thus, I wish to express my deepest gratitude to my supervisors and co-authors, Prof. Bogdan Dumitrescu and Prof. Ioan Tăbuş. Especially, I wish to express my recognition for the continuous guidance received from B. Dumitrescu and for his support towards the elaboration of the thesis. Special thanks also go to my co-author and colleague Petri Helin.

I would like to thank the pre-examiners, Dr. Stefan Werner and Dr. Radu Bîlcu, for their effort dedicated to reading the manuscript and for their constructive feedback and recommendations. I also would like to thank Dr. Stefan Werner and Dr. Kristiaan Pelckmans for agreeing to serve as opponents for the public defense of the thesis.

Special thanks are due to Virve Larmila, Ulla Siltaloppi and Elina Orava for their assistance and help with administrative matters.

I am also grateful to all my colleagues and friends, Ionuţ, Jenni, Stefan, Vlad, Septimia, Victor and Florin, who created an interesting and motivating work environment and to many other friends from outside the department, too many to mention here, for all the good moments we spent together.

Last but not least I want to express my warmest gratitude to my parents, Georgeta and Valentin, and to my brother Cristian, for their support on my four year endeavor that lead to writing this thesis. At the same time I want to thank Andrada, for motivating me towards finishing the manuscript.

March 2014, Tampere
Alexandru Onose

Contents

List of publications

The thesis consists of seven publications: two journal publications [P1, P5] and five conference papers [P2, P3, P4, P6, P7]. All publications present different approaches for solving overdetermined sparse systems of equations and are grouped below based on the characteristics of the algorithms they present.

Publications [P1, P2, P3, P4] develop fast methods that update the solution via coordinate descent like approaches. The algorithms from [P5, P6, P7] use orthogonal transforms to provide accurate solutions but, computationally, are more intensive.

[P1] A. Onose and B. Dumitrescu. Adaptive matching pursuit using coordinate descent and double residual minimization. *Signal Processing*, 93(11):3143–3150, November 2013.

[P2] A. Onose and B. Dumitrescu. Distributed coordinate descent using adaptive matching pursuit. In *Proceedings of the International Symposium on Intelligent Signal Processing and Communication Systems*, pages 513–518, Naha, Japan, November 2013.

[P3] A. Onose and B. Dumitrescu. Cyclic adaptive matching pursuit. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 3745–3748, Kyoto, Japan, March 2012.

[P4] A. Onose and B. Dumitrescu. Low complexity approximate cyclic adaptive matching pursuit. In *Proceedings of the European Signal Processing Conference*, pages 2629–2633, Bucharest, Romania, August 2012.

[P5] B. Dumitrescu, A. Onose, P. Helin, and I. Tăbuş. Greedy sparse RLS. *IEEE Transaction on Signal Processing*, 60(5):2194–2207, May 2012.

[P6] A. Onose, B. Dumitrescu, and I. Tăbuş. Sliding window greedy RLS for sparse filters. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3916–3919, Prague, Czech Republic, May 2011.

[P7] A. Onose and B. Dumitrescu. Group greedy RLS sparsity estimation via information theoretic criteria. In *Proceedings of the International Conference on Control Systems and Computer Science*, volume 2, pages 359–364, Bucharest, Romania, May 2013.

List of figures

List of tables

List of algorithms

List of acronyms

LMS least mean squares . . . 3

NLMS normalized LMS . . . 3

LS least squares . . . 3

RLS recursive least squares . . . 3

FIR finite impulse response . . . 6

BPDN basis pursuit denoising . . . 11

LASSO least absolute shrinkage and selection operator . . . 11

MP matching pursuit . . . 12

OMP orthogonal matching pursuit . . . 12

OLS orthogonal least squares . . . 12

SPARLS sparse RLS . . . 17

TWL time weighted LASSO . . . 17

TNWL online cyclic coordinate descent using time and norm weighted LASSO . . . 18

AMP adaptive matching pursuit . . . 18

CMP cyclic matching pursuit . . . 18

CAMP cyclic AMP . . . 18

CD coordinate descent . . . 18

GRLS greedy sparse RLS . . . 18

ITC information theoretic criteria . . . 19

PLS predictive least squares . . . 19

BIC Bayesian information criterion . . . 19

I-CAMP iterated CAMP . . . 25

I-CAMP-A approximate I-CAMP using the PLS criterion . . . 29

CD-AMP coordinate descent AMP . . . 29

DCD-AMP double residual CD-AMP . . . 32

D-DCD-AMP distributed DCD-AMP . . . 35

Mathematical notations

$\mathbb{R}$	set of real numbers.
$\mathcal{N}$	set of all nodes belonging to a network.
$\mathcal{N}_j$	set of neighbor nodes of node j belonging to network $\mathcal{N}$.
$\mathcal{S}, \mathcal{C}, \mathcal{A}, \mathcal{I}, \mathcal{P}, \mathcal{B}, \mathcal{G}$	subsets of positive integers; used for indexing data contained by matrices and vectors.
$a : b$	notation for the set of positive integers with values between a and b; MATLAB notation.
$\mathbf{A} \in \mathbb{R}^{n \times m}$	matrix notation; a matrix of real values having n rows and m columns.
$\mathbf{A}_{\mathcal{C}}$	columns of matrix $\mathbf{A}$ associated with the set $\mathcal{C}$.
$\mathbf{A}_{\mathcal{C},\mathcal{A}}$	part of matrix $\mathbf{A}$ containing rows associated with the set $\mathcal{C}$ and columns associated with set $\mathcal{A}$.
$\boldsymbol{a} \in \mathbb{R}^n$	vector notation; a vector of real values having n elements.
$\boldsymbol{a}_{\mathcal{C}}$	elements of vector $\boldsymbol{a}$ associated with the set $\mathcal{C}$.
$a, a_i, a_{i,j} \in \mathbb{R}$	element notation; a variable of real value.
$\{\cdot\}^T$	transpose of $\{\cdot\}$.
$\{\cdot\}^{-1}$	inverse of $\{\cdot\}$.
$\{\cdot\}^{[t,j]}$	time and node notation of any variable $\{\cdot\}$ at time t and node j.
$\{\cdot\}^{[t]}$, $\{\cdot\}(t)$	time notation for any variable $\{\cdot\}$ at time t.
$\lceil \{\cdot\} \rceil$	value of $\{\cdot\}$ rounded up to the nearest integer.
$\|\{\cdot\}\|_p$	norm p of $\{\cdot\}$.
$\|\{\cdot\}\|$	set cardinality if $\{\cdot\}$ is a set; absolute value of $\{\cdot\}$.
$\text{sign}(\{\cdot\})$	sign of $\{\cdot\}$.
$\mathcal{T}_i, \mathcal{T}_r, \mathcal{T}_f$	different network topologies; unconnected, ring and fully connected, respectively.

Chapter 1

Introduction

The vast progress made in many science areas during the past decades would not have been possible without the improvement of the signal models we use. Simple models have been very appealing since they are easy to comprehend and they can be used to generate fast solutions to diverse problems. However, due to their simplicity, many lack precision. This motivated a continuous quest for better modeling techniques.

Recently, technological developments generally linked to faster computing capabilities, allowed us to solve increasingly complex problems. The complexity can however grow very fast when the problem size increases even slightly, making many problems intractable. This is motivating a sustained research towards the development of better ways of finding solutions to increasingly more complex models.

The concept of sparse and redundant representations started from a very simple idea. In essence many signals contain redundancies and a good model should obtain the simplest, sparsest representation. Thus, the redundant signal $\boldsymbol{b} \in \mathbb{R}^t$ can be expressed by a simpler, sparse representation $\boldsymbol{x} \in \mathbb{R}^n$ through a linear transform,

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b}, \tag{1.1}$$

where $\mathbf{A} \in \mathbb{R}^{t \times n}$ contains the atoms used to express the signal $\boldsymbol{b}$ and $\boldsymbol{x}$ has only $l \ll n$ non-zero elements; $\boldsymbol{b}$ is expressed as a linear combination of only l atoms. This model is very simple but finding the sparse representation, given a large number n of atoms, is very difficult. A direct approach involves a combinatorial search for the adequate atoms.

In this thesis we present several greedy algorithms that provide accurate and robust solutions to sparse problems like (1.1) while having low computational complexity. One of the main applications of such algorithms is that of producing sparse adaptive filters. Thus, we begin by introducing some of the traditional adaptive filtering tools and their applications in Chapter 2. We present the recursive least squares (RLS) algorithm as an example of an important adaptive algorithm for generating least squares (LS) non-sparse solutions. It minimizes a

similar criterion as the sparsity-aware algorithms that are presented herein, but without any sparsity constraints. Chapter 3 contains a brief summary of the evolution of the so called sparsity-aware methods. We describe some of the most relevant batch approaches and present an overview of the adaptive methods that have been developed in the past two decades. The batch methods mainly belong to two classes; they are based on convex relaxation techniques and simultaneously estimate the supports and the coefficient values or are greedy algorithms and decouple the search for the support from the computation of the coefficient values. The adaptive methods either follow the transformation of non-sparse methods, like the least mean squares (LMS), towards being sparsity-aware or are derived from the two batch classes of methods mentioned above, modified for an adaptive setup. Finally, the main contributions that serve as a base for this thesis, namely two families of greedy adaptive sparse algorithms, are summarized in Chapter 4. The first family is derived from the batch matching pursuit (MP) algorithm [30] and uses a coordinate descent (CD) approach to estimate the coefficients while the second has its roots in the batch orthogonal least squares (OLS) method [22]. Their performance is later comparatively assessed in Chapter 5.

Chapter 2

Adaptive filtering

Linear estimation theory has a very long history. It began, in the 17'th century, with the first attempts of Galileo Galilei to analyze and to model the motions of planets. The stepping stone of modern estimation theory was, however, the development of the least squares (LS) method by Gauss [45] and Legendre [63]. Modern studies of the mean squares estimation began in the context of stochastic processes with pioneer works like those of Kolmogorov [61] and Wiener [96] in the first half of the 20'th century. Both Wiener and Kolmogorov assumed the stochastic processes to be stationary and considered the amount of available data infinite. In practice however this is seldom true; the processes are variable and the data available is limited.

2.1 Traditional solutions

Starting from the 1950s, algorithms that recursively compute the weights of a linear filter emerged. The classical approaches that lead to the development of the adaptive filter theory generally follow two main directions: they are based on stochastic gradient descent methods, or try to solve an LS estimation problem. Probably the most studied and used adaptive filtering algorithms are the least mean squares (LMS), developed by Widrow and Hoff [94, 95], and the recursive least squares (RLS), which can be credited to [76], although various derivations existed before.

The LMS minimizes a cost function of the filter weights via a stochastic gradient descent, and can be viewed as an approximation of the optimal Wiener filter [96]. Although it is fairly simple, it is efficient, robust, and has a very low complexity. It was thoroughly studied and numerous improvements [53, 80] over the original form have been proposed. Most notably are the normalized LMS (NLMS) [9, 69] and various frequency domain approaches [93].

Following an LS strategy, the RLS algorithm minimizes a quadratic cost function and can be viewed as a special case of Kalman filtering [58, 81]. Although there exist a plethora of advances over the standard RLS, which either improve the

stability [46] or lower the algorithm complexity [28], herein only some basic details are presented. This provides a link to the sparsity-aware algorithms that serve as a basis for this thesis, since they minimize a similar LS criterion.

2.2 Recursive least squares

In the adaptive filtering context, we need to find a recursive solution that minimizes the LS criterion

$$J(\boldsymbol{x}(t)) = \sum_{\tau=1}^{t} \lambda^{t-\tau} |e(\tau)|^2 \tag{2.1}$$

at each time instant t. The estimation error $e(\tau)$ is defined as the difference between the desired output $d(\tau)$ and the output data $y(\tau)$ produced by a linear filter, $\boldsymbol{x}(t) \in \mathbb{R}^n$, having as input data $\mathbf{u}(\tau) \in \mathbb{R}^n$,

$$e(\tau) = d(\tau) - y(\tau) = d(\tau) - \boldsymbol{x}^T(t)\boldsymbol{u}(\tau) = d(\tau) - \sum_{i=0}^{n-1} x_i(t)u_i(\tau). \tag{2.2}$$

We use a forgetting factor $0 < \lambda \leq 1$ to define an exponentially weighing function that limits the influence of old data and thus allows the tracking of non-stationary processes. Furthermore, the data may also be windowed using a rectangular sliding window of length $0 < w \leq t$, the summation from (2.1) being performed for $t - w + 1 \leq \tau \leq t$. We explicitly treat the sliding window case in Chapter 4.2.2 and for now we work only with exponentially windowed data.

In a typical scenario, the filter length n is smaller than the total available data, $n \leq t$. Thus, by collecting for all τ the equations from (2.2), we can view the minimizer $\boldsymbol{x}(t)$ of the criterion (2.1), as the LS solution to the overdetermined system of linear equations,

$$\mathbf{A}(t)\boldsymbol{x}(t) \approx \boldsymbol{b}(t). \tag{2.3}$$

For simplicity, we include the windowing in the data matrices. The input matrix $\mathbf{A}(t) \in \mathbb{R}^{t \times n}$ is constructed using the input data collected for all time instants, each row i being equal to

$$\boldsymbol{\alpha}(i) = \lambda^{\frac{t-i}{2}} \boldsymbol{u}(i). \tag{2.4}$$

The desired output is therefore,

$$\boldsymbol{b}(t) = [\lambda^{\frac{t-1}{2}} d(1),\ \lambda^{\frac{t-2}{2}} d(2),\ \cdots,\ \lambda^{\frac{1}{2}} d(t-1),\ d(t)]^T, \tag{2.5}$$

with direct correspondence to (2.2). This allows for a recursive description and we can construct the matrices for time t based on the past ones, from time $t-1$,

$$\mathbf{A}(t) = \begin{bmatrix} \sqrt{\lambda}\mathbf{A}(t-1) \\ \boldsymbol{\alpha}^T(t) \end{bmatrix}, \qquad \boldsymbol{b}(t) = \begin{bmatrix} \sqrt{\lambda}\boldsymbol{b}(t-1) \\ \beta(t) \end{bmatrix}, \tag{2.6}$$

where $\beta(t) = \lambda^{\frac{t-t}{2}} d(t) = d(t)$. The criterion $J(\boldsymbol{x}(t))$ can be written as the squared norm of the residual of the linear system defined by (2.3), producing

$$J(\boldsymbol{x}(t)) = \|\boldsymbol{b}(t) - \mathbf{A}(t)\boldsymbol{x}(t)\|_2^2. \tag{2.7}$$

The number of rows in the matrix $\mathbf{A}(t)$ and the vector $\boldsymbol{b}(t)$ depends on the number of samples available and can grow indefinitely long. Due to memory limitations, the information can be stored in the form of the scalar products

$$\boldsymbol{\Psi}(t) = \mathbf{A}^T(t)\mathbf{A}(t) = \lambda\boldsymbol{\Psi}(t-1) + \boldsymbol{\alpha}(t)\boldsymbol{\alpha}^T(t) \tag{2.8}$$

and

$$\boldsymbol{\phi}(t) = \mathbf{A}^T(t)\boldsymbol{b}(t) = \lambda\boldsymbol{\phi}(t-1) + \boldsymbol{\alpha}(t)\beta(t). \tag{2.9}$$

To compute the solution, the RLS algorithm updates the inverse $\mathbf{P}(t) = \boldsymbol{\Psi}^{-1}(t)$ (Lemma A.1.1) and produces the recursive filter coefficient update,

$$\boldsymbol{x}(t) = \boldsymbol{x}(t-1) + \boldsymbol{k}(t)\left[\beta(t) - \boldsymbol{x}^T(t-1)\boldsymbol{\alpha}(t)\right], \tag{2.10}$$

where $\boldsymbol{k}(t)$ is a gain vector

$$\boldsymbol{k}(t) = \frac{\mathbf{P}(t-1)\boldsymbol{\alpha}(t)}{\lambda + \boldsymbol{\alpha}^T(t)\mathbf{P}(t-1)\boldsymbol{\alpha}(t)} \tag{2.11}$$

and

$$\lambda\mathbf{P}(n) = \mathbf{P}(t-1) - \boldsymbol{k}(t)\boldsymbol{\alpha}^T(t)\mathbf{P}(t-1). \tag{2.12}$$

2.3 Applications

Due to their ability to track unknown processes, the adaptive filters have been used in various applications in signal processing covering a broad range of areas, from communications to radar, sonar or to audio. There are however several main classes of applications for the adaptive linear filters, e.g. the identification and modeling of unknown processes; the linear prediction of a given input signal; the inverse modeling of a process with the goal of recovering the process input from its output and the interference cancellation of an input signal provided that some knowledge exists about the interfering signals.

2.3.1 Channel identification

The task of identifying the physical transmission channel plays a crucial role in communication since an improper model for the way the information is distorted or perturbed, can produce serious problems, limiting the bandwidth or introducing transmission errors. For wireless communication, the transmitted carriers are reflected and scattered by different surfaces or objects. Furthermore, the channel is not constant since the transmitters are often non-stationary (e.g. mobile phones). This requires periodical channel estimation and modeling. Traditionally,

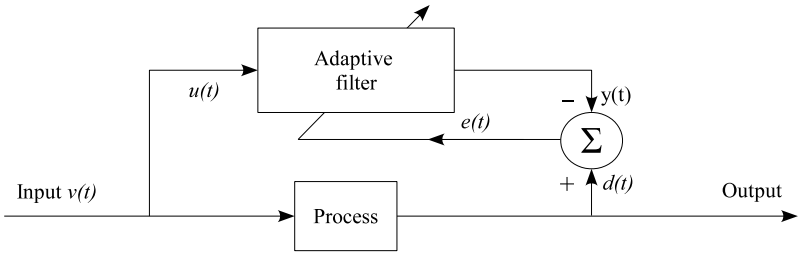


Figure 2.1: Adaptive filtering applications: Identification of an unknown process.

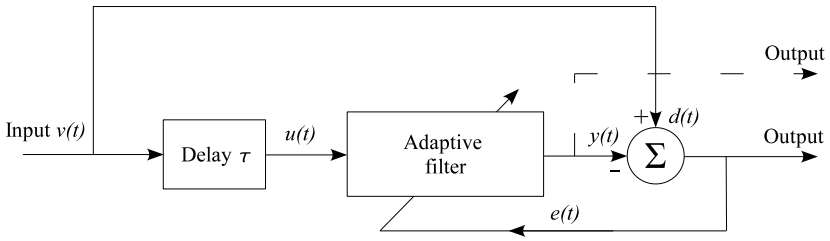


Figure 2.2: Adaptive filtering applications: Prediction.

since the computational complexity of the parametric modeling needs to be low, finite impulse response (FIR) filters are used.

For the identification task presented in Figure 2.1, we suppose that the process altering the input data is modeled by an FIR filter

$$d(t) = \boldsymbol{h}^T(t)\boldsymbol{u}(t) + \epsilon(t) = \sum_{i=0}^{n-1} h_i(t)u_i(t) + \epsilon(t), \tag{2.13}$$

where $\epsilon(t)$ represents some additive noise and

$$\boldsymbol{u}(t) = [v(t),\ v(t-1),\ \cdots,\ v(t-n+2),\ v(t-n+1)]^T. \tag{2.14}$$

The goal is to find the estimates $\boldsymbol{x}(t)$ of the true values $\boldsymbol{h}(t)$ such that the criterion (2.1) is minimized.

2.3.2 Linear prediction

In the linear prediction scenario from Figure 2.2, it is required that the adaptive filter acts as predictor of the input data, with τ steps ahead, such that the desired[1] data $d(t)$ is predicted using received past information. We have

$$d(t) = \sum_{i=0}^{n-1} x_i(t)u_i(t) + \epsilon(t), \tag{2.15}$$

where $\epsilon(t)$ is the prediction error and the input to the filter is given by the past values of the desired data

$$\boldsymbol{u}(t) = [d(t-\tau),\ d(t-\tau-1),\ \cdots,\ d(t-\tau-n+1)]^T. \tag{2.16}$$

Again, from the requirement that the estimation error should be minimized, we can employ adaptive filtering algorithms that minimize the LS criterion. In this scenario the quantities of interest can be either the prediction errors or the actual predicted values.

[1] In this case we have $d(t) = v(t)$

Chapter 3

Sparse representations: From batch to adaptive algorithms

The development of the classical linear algebra theory generated a thorough analysis of the problem of solving linear systems of equations, which lies at the core of many engineering applications [53]. During the last 20 years, the idea of sparse representations emerged from this traditional theory and is now generating increased interest due to its diverse applications in signal processing or other related fields (image compressing [15], denoising [54] and deburring [41], wireless channel estimation [30], direction of arrival [65], audio processing [47, 48]). Many media types (image, video, audio) can be sparsely represented using transform domain methods and many tasks related to them can be viewed as finding solutions to systems of linear equations [32].

A distinct feature of the derived algorithms is the way they process the available data; the batch algorithms require the whole data to be available, while the adaptive algorithms take advantage of the previously computed solution and update it recursively. Historically, batch methods have received the most attention since they allow for a more facile theoretical analysis [38]. Recently, adaptive methods [P5, 10, 12, 62] have come into focus due to applications that require low computational complexity. Many other research directions related to sparse representations are also beginning to be developed [39].

When compared to traditional adaptive approaches, the search for the sparse solutions poses the additional challenge of finding the correct sparsity. The existent sparsity-aware algorithms resulted from several distinct directions of research. The first is based on traditional tools and mostly deals with the improvement of the LMS that make it sparsity-aware [36, 70, 92]. Convex relaxation techniques, in which an LS criterion like in (2.1) is penalized with the ℓ_p-norm, $p \geq 1$, of the solution received much of the interest. This approach leads to the development of algorithms like [10, 12, 24]. Other directions lead to the development of greedy

algorithms [P5, 66, 77] or algorithms based on projections onto ℓ_1 balls [62].

The problem all algorithms try to solve is closely related to (2.3) but now has additional constraints. The solution $\boldsymbol{x}(t)$ is sparse with a sparsity level l_t. We define l_t as the number of true coefficients different from zero, $l_t = \|\boldsymbol{x}(t)\|_0$, and we suppose that the sparsity level is low such that $l_t \ll n$.

Thus, we can state the problem as the following, non convex, minimization[1]:

$$\begin{aligned} &\text{minimize } \|\boldsymbol{x}(t)\|_0 \\ &\text{subject to } \mathbf{A}(t)\boldsymbol{x}(t) \approx \boldsymbol{b}(t). \end{aligned} \tag{P_0}$$

A brute force approach to solve (P_0) involves a combinatorial search over all possible subsets of the solution support which is generally unfeasible even for relatively small dimensions of the data matrices. To overcome such difficulties, various approximate methods have been proposed together with a theoretical framework necessary for a thorough analysis of the algorithm performance.

3.1 Batch methods

The optimization problem (P_0) can be regarded as being composed of two semi-independent parts: a search for the support of the non-zero coefficients and the computation of their values. Two main directions for finding sparse solutions exist.

The first involves the relaxation of the non convex criterion containing the ℓ_0 pseudo-norm. Generally an $\ell_p = \|\boldsymbol{x}\|_p$, $p > 0$, norm or any other smooth function that can promote sparsity are used[2]. In this case there is no explicit search for the support, the zero coefficients result from the optimization problem. Furthermore, if the relaxed constraint is convex there are efficient convex optimization solvers and the convex function also guarantees the uniqueness of the solution.

The second approach introduces a separation between the coefficient values and the solution support. It tries to find the support $\mathcal{A}$ with a low cardinality $|\mathcal{A}|$ for which the norm of the residual

$$\boldsymbol{r}_{\mathcal{A}} = \boldsymbol{b} - \mathbf{A}_{\mathcal{A}}\boldsymbol{x}_{\mathcal{A}} \tag{3.1}$$

is minimized. In what follows, we name the support columns active and the remaining ones inactive since they do not contribute to the solution. We denote by $\mathcal{A}$ and by $\mathcal{I}$ the sets of indexes of the active and inactive columns associated with a given i-sparse solution, respectively.

Analyzing for all possible support combinations has an exponential complexity and is not practical, so, typically, a greedy support selection strategy is used. After the support is selected the coefficients can be easily computed by solving an LS optimization problem to the data restricted to the support,

$$\boldsymbol{x}_{\mathcal{A}} = \left(\mathbf{A}_{\mathcal{A}}^T\mathbf{A}_{\mathcal{A}}\right)^{-1}\mathbf{A}_{\mathcal{A}}^T\boldsymbol{b}. \tag{3.2}$$

[1] We discuss only approximate recovery problems since the exact recovery case is generally easier to handle.

[2] We drop the time index from the notations since it is irrelevant for the batch methods.

The requirement for the system of equations to be overdetermined, to have more than n equations, is no longer mandatory since we actually only need to compute the coefficient values $\boldsymbol{x}_{\mathcal{A}}$ associated with the support. In this sense, to compute the solution we require that $|\mathcal{A}| \leq t$ and thus the restricted solution (3.2) is still computed from an overdetermined system.

While initially the problem was studied in the context of exact representations where $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$, later the techniques were adapted for approximate solutions. We present some of the main ideas of the convex relaxation and we focus more on the greedy approaches.

3.1.1 Convex relaxation techniques

The ℓ_0 pseudo-norm from problem (P_0) has a discrete nature, it counts the number of non-zero elements,

$$\|\boldsymbol{x}\|_0 \triangleq \lim_{p \to 0} \sum_{i=0}^{n-1} |x_i|^p. \tag{3.3}$$

The definition suggests that we can relax the discontinuous ℓ_0 pseudo-norm to a continuous ℓ_p norm, $p > 0$, to make the problem more tractable. A small p ensures a good approximation but only for $p \geq 1$ the problem becomes convex and allows the use of convex programing solvers for an efficient solution computation.

The basis pursuit denoising (BPDN) optimization problem

$$\begin{array}{c} \text{minimize } \|\boldsymbol{x}\|_p^p \\ \text{subject to } \mathbf{A}\boldsymbol{x} \approx \boldsymbol{b} \end{array}, \tag{P_p}$$

introduced by [23], has attracted much interest [17, 34, 89, 90]. Choosing $p = 1$ provides the closest convex relaxation to the ℓ_0 pseudo-norm and also takes full advantage of the convex optimization solvers.

Since the constraint $\mathbf{A}\boldsymbol{x} \approx \boldsymbol{b}$ can be viewed as an LS approximation problem, it is readily translated to $\|\boldsymbol{b} - \mathbf{A}\boldsymbol{x}\|_2 \leq \epsilon$, with a given small ϵ, and thus, a natural related convex optimization task emerges,

$$\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{b} - \mathbf{A}\boldsymbol{x}\|_2^2 + \gamma\|\boldsymbol{x}\|_1. \tag{3.4}$$

This is an LS regularization problem, similar to the celebrated Tikhonov regularization [88], and was introduced under the name of least absolute shrinkage and selection operator (LASSO) [87]. It has efficient batch solver, the least angle regression [37]. Here, the parameter γ represents a tradeoff between the approximation error and the sparsity level.

While many other sparsity inducing algorithms have been proposed (e.g. iterative re-weighted least squares [51], Dantzig selector [18] or iterative re-weighted ℓ_1 minimization [19]), the LASSO and BPDN play a central role in the search for sparsity inducing optimization tasks. The main research goal surrounding them is the analysis of the performance with the establishment of strong theoretic bounds for recovering the true support.

3.1.2 Greedy algorithms

Among the existing greedy methods, the most representative are the matching pursuit (MP), the orthogonal matching pursuit (OMP) [66, 75] and the orthogonal least squares (OLS) [22], known also as optimized orthogonal matching pursuit [77]. Other methods have also been developed [71, 86] but all follow the same general principle. They involve a discrete search for the support followed by the estimation of the associated coefficients.

Matching pursuit

The greedy MP algorithm selects one by one columns from a set of candidate columns $\mathcal{C}$ such that, by choosing the coefficient x_{κ_i} associated with the column κ_i, we produce the smallest residual norm,

$$\kappa_i = \arg\min_{j \in \mathcal{C}} \|\boldsymbol{r}_{i-1} - \boldsymbol{a}_j x_j\|_2^2, \tag{3.5}$$

with $\boldsymbol{r}_{i-1}$ defined by (3.1) for an active set $\mathcal{A}$ composed of $i-1$ columns. If the active set is empty, we have $\boldsymbol{r}_0 = \boldsymbol{b}$. For a given column $\boldsymbol{a}_j$, the coefficient x_j that minimizes the residual norm is given by

$$x_j = \frac{\boldsymbol{a}_j^T \boldsymbol{r}_{i-1}}{\|\boldsymbol{a}_j\|^2} \tag{3.6}$$

and thus an equivalent selection criterion can be derived

$$\begin{aligned} \kappa_i &= \arg\min_{j \in \mathcal{C}} \|\boldsymbol{r}_{i-1} - \boldsymbol{a}_j x_j\|_2^2 = \arg\min_{j \in \mathcal{C}} \left\| \boldsymbol{r}_{i-1} - \frac{\boldsymbol{a}_j^T \boldsymbol{r}_{i-1}}{\|\boldsymbol{a}_j\|^2} \boldsymbol{a}_j \right\|_2^2 \\ &= \arg\min_{j \in \mathcal{C}} \left(\|\boldsymbol{r}_{i-1}\|_2^2 - \frac{|\boldsymbol{a}_j^T \boldsymbol{r}_{i-1}|^2}{\|\boldsymbol{a}_j\|^2} \right) = \arg\max_{j \in \mathcal{C}} \frac{|\boldsymbol{a}_j^T \boldsymbol{r}_{i-1}|^2}{\|\boldsymbol{a}_j\|^2}. \end{aligned} \tag{3.7}$$

This can be seen as the selection of the column κ_i that has the largest relative projection on the residual $\boldsymbol{r}_{i-1}$. After the column is selected, it is included in the support

$$\mathcal{A} \leftarrow \mathcal{A} \cup \{\kappa_i\} \tag{3.8}$$

and the residual is updated

$$\boldsymbol{r}_i = \boldsymbol{r}_{i-1} - x_{\kappa_i} \boldsymbol{a}_{\kappa_i}, \tag{3.9}$$

with x_{κ_i} given by (3.6). The search for the next column κ_{i+1} continues iteratively until a stopping criterion is met, generally involving the diminishing of the residual norm, $\|\boldsymbol{r}_i\|_2^2 \leq \epsilon$ or until a certain number of columns l are selected. The MP pursuit algorithm that searches for l support columns is summarized in Algorithm 3.1.

Any last coefficient x_{κ_i} computed by (3.6) is optimal only locally, in the context of the previously selected coefficients. Due to this and the fact that once the

Algorithm 3.1: **Matching pursuit (MP)**

input: $\mathbf{A}$; $\boldsymbol{b}$; l;
output: $\mathcal{A}$; $\boldsymbol{x}_{\mathcal{A}}$;

1 For $i = 1 : l$

1.1 Choose column κ_i according to (3.7) with $\mathcal{C} = \mathcal{I}$, i.e. all inactive columns are candidates

1.2 Compute the coefficient value x_{κ_i} like in (3.6)

1.3 Update the active set $\mathcal{A}$ and the inactive set $\mathcal{I}$

1.4 Update the residual to account for the new coefficient like in (3.9)

coefficient is computed its value does not change, the global solution is suboptimal and the residual $\boldsymbol{r}_{\mathcal{A}}$ is not orthogonal to the active columns $\mathbf{A}_{\mathcal{A}}$. To improve the solution, after the selection of the support, an orthogonalization step can be performed to recompute the whole solution by replacing the MP coefficients with the LS solution restricted to the support $\mathcal{A}$ as in (3.2). In this case all coefficient values change and the residual update is

$$\boldsymbol{r}_i = \boldsymbol{b} - \sum_{j=1}^{i} x_{\kappa_j} \boldsymbol{a}_{\kappa_j}. \tag{3.10}$$

This produces the OMP algorithm [75].

Orthogonal least squares

The decision, from (3.5), to choose a new position κ_i is a function of each column $\boldsymbol{a}_{\kappa_i}$ optimally scaled by its associated coefficient. The already selected support is indirectly considered through the residual $\boldsymbol{r}_i$. This selection does not guarantee the minimization of the LS criterion, for an i-sparse solution, because the new column is not orthogonal to the subspace defined by the previous selected $i-1$ active columns.

Thus, an improved algorithm, the OLS method, arises from the idea of the optimal selection and coefficient computation via LS. The selection of a new best column κ_i to be added to the active set $\mathcal{A}$ changes to

$$\kappa_i = \arg\min_{j \in \mathcal{C}} \|\boldsymbol{b} - \mathbf{A}_{\mathcal{A}} \boldsymbol{x}_{\mathcal{A}} - \boldsymbol{a}_j x_j\|_2^2, \tag{3.11}$$

such that now we select κ_i based on the best solution from the whole restricted support $\mathcal{A} \cup \kappa_i$, opposed to just looking for the best x_{κ_i} and keeping the same old coefficients $\boldsymbol{x}_{\kappa_{1:i-1}}$, as it is in (3.5). This changes all coefficient values on the support when we add a new position to it. The naive approach of solving for each candidate column $j \in \mathcal{C}$ an LS problem is computationally very intensive. To overcome this, we can find the solution using a partial QR decomposition with

Algorithm 3.2: **Orthogonal least squares (OLS)**

input: $\mathbf{R} = \mathbf{A}$; $\boldsymbol{b}$; l;
output: $\mathcal{A}$; $\boldsymbol{x}_{\mathcal{A}}$;

1 For $i = 1 : l$

1.1 Choose column κ_i according to (3.15) with all inactive columns as candidates, $\mathcal{C} = \mathcal{I}$

1.2 Swap columns κ_i and i of matrix $\mathbf{R}$ and update the active set $\mathcal{A}$

1.3 Find the Householder reflector $\mathbf{Q}_i$ that zeros column i from $\mathbf{R}$ below the diagonal

1.4 Apply $\mathbf{Q}_i$ on $\mathbf{R}$; $\mathbf{R} \leftarrow \mathbf{Q}_i\mathbf{R}$

1.5 Apply $\mathbf{Q}_i$ on $\boldsymbol{b}$; $\boldsymbol{b} \leftarrow \mathbf{Q}_i\boldsymbol{b}$

2 Compute the l-sparse solution restricted to the support like in (3.2)

pivoting,

$$\mathbf{AP} = \mathbf{QR}, \tag{3.12}$$

where $\mathbf{Q} \in \mathbb{R}^{t\times t}$ is an orthogonal matrix and $\mathbf{R} \in \mathbb{R}^{t\times n}$ is upper triangular for the solution support. The matrix $\mathbf{P}$ permutes the columns in the support to the first positions to allow for a simpler description of the method but it is otherwise not essential. It can be implemented using a permutation vector that records the order of the columns in $\mathbf{A}$ and hereafter we consider the data implicitly permuted to ease the presentation. The orthogonal matrix $\mathbf{Q}$ is not needed explicitly, it only has to be applied on $\mathbf{A}$ and $\boldsymbol{b}$. After the partial triangularization, any i-sparse solution is easily found by solving an upper triangular system defined by the matrix $\mathbf{R}$ and vector $\boldsymbol{b}$ restricted to the support,

$$\boldsymbol{x} = \mathbf{R}_{\mathcal{A},\mathcal{A}}^{-1}\boldsymbol{b}_{\mathcal{A}}. \tag{3.13}$$

We define $\mathbf{R} = \mathbf{Q}^T\mathbf{A}$ and, by an abuse of the notation, we keep $\boldsymbol{b} \leftarrow \mathbf{Q}^T\boldsymbol{b}$.

The solution support search is similar to MP and OMP, a greedy search over the whole set of candidate columns $\mathcal{C}$. However, now the solution needs to minimize the LS criterion (2.1). Each new selected column κ_i is permuted to the position i in the matrix $\mathbf{R}$ and a partial triangularization of the matrix is produced, with the use of a Householder reflector (Definition A.3.1), by introducing zeroes below the diagonal (Theorem A.3.1). After $i-1$ columns are selected, the matrices $\mathbf{R}$ and $\mathbf{b}$ can be partitioned as

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_{\mathcal{A},\mathcal{A}} & \mathbf{R}_{\mathcal{A},\mathcal{I}} \\ \mathbf{0} & \mathbf{F} \end{bmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{bmatrix} \boldsymbol{b}_{\mathcal{A}} \\ \boldsymbol{g} \end{bmatrix}, \tag{3.14}$$

where $\mathbf{R}_{\mathcal{A},\mathcal{A}}$ is upper triangular. Any new column κ_i is chosen such that its orthogonal component, in relation to the already selected active columns, has the

largest relative projection on the residual,

$$\kappa_i = \arg\max_{j \in \mathcal{C}} \frac{|\boldsymbol{f}_j^T \boldsymbol{g}|}{\|\boldsymbol{f}_j\|_2}. \tag{3.15}$$

A selection according to the (3.15) is guaranteed to minimize the LS criterion [P5, Appendix I]. Afterwards, the upper triangular form is restored and the search for a new column continues until a stop criterion similar to MP is met. The algorithm is summarized in Algorithm 3.2. The stopping criterion is met once l columns are selected.

3.1.3 Performance analysis

It is generally hard to guarantee the optimality of the solutions given by the greedy or ℓ_1 relaxation methods because this would require the verification of the solution given by every possible support and thus the return to the combinatorial search that we try to avoid. Due to this, the research focus was directed towards finding performance (uniqueness, optimality and stability) bounds based on the structure of the matrix $\mathbf{A}$. The studies started initially from the exact recovery problem and a matrix $\mathbf{A}$ composed of a union of orthogonal basis and developed gradually towards the more difficult approximate solutions and general matrices [38].

The main tools, used to compute performance bounds, are the spark and the mutual coherence [33, 34, 51], and the restricted isometry property [17, 16, 32]. However, sometimes in practice, the bounds are too large for an efficient characterization, empirical simulation usually providing better results than the bounds would suggest. Ongoing research is aiming at providing better bounds and understanding of the sparsity recovery conditions.

The Spark and Mutual Coherence

The spark and mutual coherence measure the dependence of the columns of $\mathbf{A}$ and serve as tools for the characterization of the exact recovery conditions.

Definition 3.1.1. *Given a matrix* $\mathbf{A}$ *the quantity* $\text{spark}(\mathbf{A})$ *is defined as the smallest number of columns from* $\mathbf{A}$ *that are linearly dependent.*

The spark describes the null space of the matrix and it is difficult to compute. The computation requires a combinatorial search through all the possible subsets of the matrix $\mathbf{A}$ and has about the same complexity as a naive approach to solving (P_0), thus making its use impractical.

The spark is however naturally bounded, $1 \leq \text{spark}(\mathbf{A}) \leq n+1$. The upper bound is too large for any practical use thus, a better one is computed with the use of the mutual coherence.

Definition 3.1.2. *Given a matrix* $\mathbf{A} \in \mathbb{R}^{t \times n}$ *the mutual coherence,* $\mu(\mathbf{A})$*, is defined as the largest absolute normalized inner product of the columns from* $\mathbf{A}$

$$\mu(\mathbf{A}) \triangleq \max_{i \neq j} \frac{|\boldsymbol{a}_i^T \boldsymbol{a}_j|}{||\boldsymbol{a}_i||_2 ||\boldsymbol{a}_j||_2}. \tag{3.16}$$

It provides a measure of the largest correlations of the columns from $\mathbf{A}$. Small values characterize near orthogonality of the columns which makes the recovery process more facile. It can provide a simple criterion for estimating the uniqueness of the sparse solutions (Theorem A.2.2).

The notion of spark can be extended to incorporate the approximate solutions. The generalization $\text{spark}_\eta(\mathbf{A})$, characterizes the proximity to the null space of the matrix $\mathbf{A}$ instead of the null space itself. Thus, the $\text{spark}_\eta(\mathbf{A})$ represents the smallest number of columns that have together the smallest singular value at most equal to η. In this case the solution is no more unique and the theoretical results aim to guarantee the solution stability and try to characterize how far it is from the true solution (Theorem A.2.4).

The mutual coherence can be used to compute bounds for the spark and spark_η (Theorems A.2.1, A.2.3) and in turn they can be used to refine the recovery conditions of either greedy methods [89] or relaxation methods like BPDN [90].

Restricted Isometry Properties

A different approach to the search for a good way of characterizing the stability of the solution was developed in [17]. It involves the use of an alternate tool, the restricted isometry property, instead of the spark and mutual coherence.

Definition 3.1.3. *For a matrix $\mathbf{A} \in \mathbb{R}^{t\times n}$ with normalized columns, for every integer $s \leq n$, the s-restricted isometry constant δ_s is defined as the smallest quantity such that the matrix $\mathbf{A}_\mathcal{S}$ obeys*

$$(1-\delta_s)\|\boldsymbol{c}\|_2^2 \leq \|\mathbf{A}_\mathcal{S}\boldsymbol{c}\|_2^2 \leq (1+\delta_s)\|\mathbf{c}\|_2^2 \tag{3.17}$$

for any $\mathbf{A}_\mathcal{S}$, containing a subset $\mathcal{S}$ of at most s columns, and any vector $\boldsymbol{c}$.

The restricted isometry property provides information about the minimal distance $1-\delta_s$ away from singularity, in terms of smallest eigenvalue, of any subset of s columns. It was developed as a main tool for the stability analysis in [16, 17].

3.2 Adaptive algorithms

In recent years the focus on sparse methods has shifted towards adaptive algorithms. They provide fast implementations and are able to track non-stationary processes.

3.2.1 Traditional methods and the sparse problem

Sparsity-aware adaptive algorithms generally provide better performance, stationary error and convergence speed, over classical methods since the number of coefficients needed to be estimated is smaller. For example, for a typical RLS algorithm, a problem where there are less samples t than the number of coefficients n, is ill-posed and the algorithm can not produce a solution. Sparse methods however, searching for only few coefficients $l \ll n$, can compute a precise solution.

The research that extends existing classical adaptive methods to deal with sparse problems consists mainly of enhancements of the LMS algorithm. The main goal is to provide faster convergence speed by incorporating the sparsity information into the solution while maintaining the light computation burden of the LMS like methods. These methods have been widely used in applications (e.g. echo canceling, system identification) that require very low computational complexity. While many algorithms exist, the majority start from the NLMS and aim at penalizing the coefficients to induce sparsity [36, 70, 92].

In [92] the authors use the framework from [91] together with the ideas from [13] to introduce an improved proportionate NLMS. They propose a family of robust adaptive filters by constraining the amount of perturbation in the solution that can be introduced by a new sample

$$\|\boldsymbol{x}(t) - \boldsymbol{x}(t-1)\|_2^2 \leq \delta_{t-1} \tag{3.18}$$

and minimizing a criterion function

$$J(\boldsymbol{x}(t)) = (\boldsymbol{x}(t) - \boldsymbol{x}(t-1))^T \mathbf{X}(t)(\boldsymbol{x}(t) - \boldsymbol{x}(t-1)) + \frac{e(t)}{\boldsymbol{\alpha}^T(t)\mathbf{Y}(t)\boldsymbol{\alpha}(t)}. \tag{3.19}$$

The matrices $\mathbf{X}(t)$ and $\mathbf{Y}(t)$ are positive definite and different choices will lead to different algorithm performance.

Building on the NLMS, [36] introduces an uneven adaptation of the solution coefficients exploiting the sparseness to achieve significantly faster adaptation than the conventional NLMS with only a modest increase in computational complexity. Other algorithms usually propose different penalization schemes [70] or require more prior information about the sparsity [44].

3.2.2 Convex relaxation techniques

The algorithms able to produce recursive solutions have the starting point in the batch convex relaxation techniques. They are able to reuse past computed solution and update them with new available data. Generally, the approach of minimizing an error criterion penalized by the ℓ_1 norm of the solution has attracted most of the interest because it translates to an optimization problem without additional constraints [10, 11, 12]. The error criterion used is the LS, producing at each time instant t a criterion similar to (3.4),

$$\min_{\boldsymbol{x}(t)} \frac{c}{2}\|\boldsymbol{b}(t) - \mathbf{A}(t)\boldsymbol{x}(t)\|_2^2 + \gamma\|\boldsymbol{x}(t)\|_1. \tag{3.20}$$

Two of the most recent representative methods are the sparse RLS (SPARLS) algorithm [12] and the time weighted LASSO (TWL) [10].

The SPARLS algorithm borrows from [41] the idea of using a low complexity expectation maximization to generate an iterative shrinkage algorithm [14] that minimizes the cost function from (3.20, $c = \sigma^{-2}$); the constant σ^2 is the variance of the noise that affects the outputs. Other expectation maximization algorithms have been developed, for instance in [59].

The TWL family of algorithms solves the same optimization problem (3.20, $c = 1$) but uses an online coordinate descent (CD) strategy to efficiently compute the solution. It can be modified to include an additional norm weighting of the solution (resulting in the online cyclic coordinate descent using time and norm weighted LASSO (TNWL) algorithm) based on the coefficients computed using the ordinary RLS, that penalize the sparse solution differently and accelerates the convergence speed. It introduces a linearly decreasing penalty on the coefficients for which the RLS estimates x_i obey $\mu_n \leq x_i \leq a\mu_n$ and removes the TWL penalty completely for coefficients greater than $a\mu_n$ (a and μ_n are configurable parameters).

3.2.3 Greedy algorithms

Research into this direction started with the pioneering work in the batch sparse methods that resulted in the development of the MP and OLS. The research performed in this direction serves as the basis for this thesis.

The MP was easily transformed allowing for the recursive implementation in the adaptive matching pursuit (AMP) [29]. Research into the batch MP produced the cyclic matching pursuit (CMP) [27], which serves as a base for the cyclic AMP (CAMP) [P3]; this can be seen as a form of CD on the direction given by the MP estimates [P1].

Spawning form the batch OLS, adaptive methods were devised with the use of tools like the Householder reflectors and Givens rotations (more details in Appendix A.3). They include the greedy sparse RLS (GRLS) family of algorithms [P5, P6, P7] which was first introduced in [35].

Both families of greedy sparse adaptive algorithms are treated in detail in Chapter 4.

3.2.4 Support cardinality estimation

The task of estimating the cardinality of the support is fundamental for the performance of the algorithms. To this regards, there are two main directions for finding the sparsity level.

The convex relaxation techniques jointly find the support and the coefficient values. They usually employ different configuration parameters that represent a compromise between the sparsity level and the coefficient estimation error. On the other hand, for greedy methods the true support cardinality is generally harder to find because of the discrete nature of the search for the support. Generally, the estimation of the true number of non-zero coefficients is performed by analyzing the residual. We describe the main ideas surrounding the sparsity estimation for greedy methods, which is closely related to the problem of order selection (e.g. for auto regressive models [49]).

Sparsity estimation for greedy methods

The decision when to stop adding columns to the solution is generally quite difficult for the greedy methods. A stopping criterion $\|\boldsymbol{r}_i\|_2^2 \leq \epsilon$ is based on a simple

analysis of the residual but it does not provide a reliable estimate without a priori information (how large or small is ϵ?).

Besides using only the residual alone, we can take advantage of the inherent order introduced by the selection strategy for the coefficients that are on the support. This can be exploited to allow the online estimation of the sparsity level with the use of the information theoretic criteria (ITC). The ITC were developed mainly for order/model selection tasks but they can be readily adapted to serve as a tool that decides how many coefficients can be included in the solution. Various methods for model selection have been suggested [8, 73, 78, 79, 83] for stationary processes. These can be extended for the non-stationary case [49, 52]; a recent overview on ITC for forgetting factor least-squares algorithms can be found in [49].

While other ITC may be used, herein we only present two, namely the Bayesian information criterion (BIC) [83] and the predictive least squares (PLS) [79].

Bayesian information criterion

The BIC uses the squared norm of the residual and penalizes it with the model complexity (the number of non-zero coefficients) resulting in

$$\mathrm{BIC}_i = t_{\mathrm{ef}} \ln \|\boldsymbol{r}_i\|_2^2 + (i+1) \ln t_{\mathrm{ef}}. \tag{3.21}$$

The residual $\boldsymbol{r}_i$ is computed using the i-sparse solution. The quantity t_{ef} represents the effective number of samples that are used to compute the solution. In a non-stationary environment where a forgetting factor λ is used, it is defined by $t_{\mathrm{ef}} = \sum_{i=1}^{t} \lambda^{t-i}$. For a large t this can be approximated by $t_{\mathrm{ef}} \approx \frac{1}{1-\lambda}$. The BIC criterion attains its minimum for the solution that has the best compromise between model complexity and estimation error. This gives an estimate $\tilde{l}$ of the true sparsity l_t,

$$\tilde{l} = \arg\min_i \mathrm{BIC}_i. \tag{3.22}$$

Predictive least squares criterion

The PLS criterion [79] has the following form

$$\mathrm{PLS}_i = \sum_{j=1}^{t} \lambda^{t-j} e_i(j)^2, \tag{3.23}$$

where $e_i(j)$ is the a priori estimation error at time j produced by an i-sparse solution $\boldsymbol{x}(j-1)$ from time $j-1$,

$$e_i(j) = d(j) - \boldsymbol{\alpha}(j)^T \boldsymbol{x}(j-1). \tag{3.24}$$

The criterion (3.23) can be updated easily by

$$\mathrm{PLS}_i(t) = \lambda \mathrm{PLS}_i(t-1) + e_i(t)^2. \tag{3.25}$$

Similarly to the BIC, the PLS is minimum when the model has a sparsity close to the best one and produces the estimate

$$\tilde{l} = \arg\min_i \mathrm{PLS}_i. \tag{3.26}$$

Chapter 4

Greedy sparse adaptive algorithms

Research towards adaptive greedy methods has been somewhat limited. Most was directed towards the development of batch algorithms. This is mainly due to the difficulties of finding an accurate stopping criterion and of providing a strong theoretical analysis of the average performance. Different approaches have been proposed. For instance MP can be converted to an adaptive form [29] but its performance is usually poor. An adaptive version of the OMP algorithm for time varying environments is suggested in [60] but no implementation details are provided.

This chapter follows the developments towards robust, well performing greedy methods that emerged from two directions of research. The first can be linked with the batch cyclic matching pursuit (CMP) [27] and AMP [29]. The CMP methods was transformed to work in a time varying context producing the cyclic AMP (CAMP) algorithm [P3] and a low complexity approximate variant [P4]. By spreading the cyclic updates in time, it can be viewed as an CD on the directions given by the matching pursuit column selection criterion [P1].

A second direction, based on the principles of the OLS, produced the greedy sparse RLS (GRLS) algorithm [P5] with online sparsity estimation via ITC. It was adapted for a sliding window input data [P6] and for a group sparsity problem [P7].

4.1 Greedy sparse coordinate descent methods

Coordinate descent methods are among the simplest approaches that can be devised to solve various convex optimization problems. They have been efficiently used to produce adaptive algorithms for finding either full or sparse solutions to the LS problem (2.1). One of the greatest benefits of the CD methods, when applied to sparse problems, is their computational efficiency [10, 11, 42, 43]. CD remains a versatile tool that can be incorporated into many algorithms.

The CD implementation of the RLS algorithm proposed in [99] can be adapted for the computation of sparse solutions in [98] by adding different penalty functions. The algorithms from [10, 11] propose the use of the CD to allow for a low complexity, adaptive minimization of the LASSO penalty function.

Batch methods can also be devised. In [84, 85] the MP was combined with a CD like approach to provide improved performance for computing batch sparse solutions. This section follows the improvements that sprung from the use of the CD coupled with a greedy MP like, sparsity inducing strategy, in an adaptive context. Two families of CD adaptive algorithms have been proposed; the first uses a cyclic update of the adaptive matching pursuit coefficients [P3, P4] and can be viewed as a CD towards the current time t optimal solution; the second spreads the CD in time [P1] with the aim of approaching the optimal solution over a number of iterations. For both methods, the use of the CD improves the solution compared to the AMP and produces a relatively low complexity algorithm.

4.1.1 Recursive implementation

An adaptive implementation can not be based on the full length matrices $\mathbf{A}^{[t]} \in \mathbb{R}^{t\times n}$ and $\boldsymbol{b}^{[t]} \in \mathbb{R}^t$ since they grow indefinitely with the time[1]. For this purpose all algorithms are implemented using the scalar products

$$\boldsymbol{\Psi}^{[t]} = \boldsymbol{A}^{[t]^T}\boldsymbol{A}^{[t]} \quad \text{and} \quad \boldsymbol{\phi}^{[t]} = \boldsymbol{A}^{[t]^T}\boldsymbol{b}^{[t]}. \tag{4.1}$$

At each time instance t, when new input vector $\boldsymbol{\alpha}^{[t]}$ and output data $\beta^{[t]}$ are available, we have

$$\boldsymbol{b}^{[t]} = \begin{bmatrix} \sqrt{\lambda}\boldsymbol{b}^{[t-1]} \\ \beta^{[t]} \end{bmatrix}, \quad \mathbf{A}^{[t]} = \begin{bmatrix} \sqrt{\lambda}\mathbf{A}^{[t-1]} \\ \boldsymbol{\alpha}^{[t]^T} \end{bmatrix}. \tag{4.2}$$

This can be translated to a recursive update of the scalar products,

$$\begin{aligned} \boldsymbol{\Psi}^{[t]} &= \lambda\boldsymbol{\Psi}^{[t-1]} + \boldsymbol{\alpha}^{[t]}\boldsymbol{\alpha}^{[t]^T} \\ \boldsymbol{\phi}^{[t]} &= \lambda\boldsymbol{\phi}^{[t-1]} + \beta^{[t]}\boldsymbol{\alpha}^{[t]}. \end{aligned} \tag{4.3}$$

Using $\boldsymbol{\Psi}^{[t]}$ and $\boldsymbol{\phi}^{[t]}$ eliminates the need to store the long matrix $\mathbf{A}^{[t]}$ or vectors $\boldsymbol{b}^{[t]}$ and $\boldsymbol{r}^{[t]}$.

This is a standard approach for adaptive filter algorithms. For example, it can be also used for the RLS algorithm as presented in Chapter 2. However, in what follows we still present the operations using the vector form for simplicity. Most equations involving the coefficient computation or update are already expressed using the scalar products. For the others, multiplying by $\mathbf{A}^{[t]^T}$ on the left produces the necessary scalar product form.

[1] We switch henceforward to a different time notation by using the upper script $^{[t]}$ to allow for a more compact presentation.

4.1.2 Cyclic adaptive matching pursuit

The CAMP algorithm [P3] has its roots in the MP [66] and its adaptive counterpart [29]. The solution is improved since the method approximates an orthogonalization process via a cyclic coefficient update similarly to the batch methods introduced by [27, 85].

We aim to minimize the squared norm of the residual (2.1) under the constraints that the solution is sparse. The algorithm follows the MP selection strategy by picking one by one columns from the matrix $\mathbf{A}^{[t]}$ based on their alignment with the current residual like in (3.7). To ease the presentation and since it does not affect the algorithm, we permute the selected columns to the first positions of the matrix $\mathbf{A}^{[t]}$ similarly to the orthogonal least squares. This also permutes the solution moving the non-zero coefficients on the first locations of the vector $\boldsymbol{x}^{[t]}$.

Solution computation and cyclic updates

During the column selection at time t, we perform the search for a best column on position i like in (3.7) but on a limited subset of columns $\mathcal{C}$. This is possible due to the assumed slow time variability of the solution and consequently of the solution support. Thus, we use the existing support of the m-sparse solution from time $t-1$ and update it, after receiving new data at time t, to account for a possible small variation. This is done as follows:

- for each position $i = 1 : m-1$ of the active set we let compete only columns i and $i+1$; this translates to allowing only permutations between neighbors in the active set;
- for the last active position m, all columns m to n are allowed to compete; thus, at most a single inactive column may become active at time t and only enters the active set on the last position.

Such a selection scheme allows only gradual changes in the support but otherwise does not impose other restrictions.

To exemplify, we consider the past neighbors i and $i+1$ from time $t-1$ as candidates for the position current position $i < m$,

$$\kappa = \arg\max_{j \in \mathcal{C}} \frac{|\boldsymbol{a}_j^{[t]T} \boldsymbol{r}_{i-1}^{[t]}|^2}{\|\boldsymbol{a}_j^{[t]}\|_2^2}, \quad \text{with} \;\; \mathcal{C} = \{i, i+1\}. \tag{4.4}$$

If the column $i+1$ is found better by (4.4) then we permute[1] columns i and $i+1$ and we update the coefficient $x_i^{[t]}$ based on (3.6) and the residual $\boldsymbol{r}_i^{[t]}$ like in (3.9),

$$x_i^{[t]} = \frac{\boldsymbol{a}_i^{[t]^T} \boldsymbol{r}_{i-1}^{[t]}}{\|\boldsymbol{a}_i^{[t]}\|_2^2}, \tag{4.5}$$

[1] When we apply the permutations we implicitly update the active set $\mathcal{A}^{[t]}$. Also note that the permutation step is not critical for the algorithms; they can be easily adapted to work without it.

$$\boldsymbol{r}_i^{[t]} = \boldsymbol{r}_{i-1}^{[t]} - x_i^{[t]} \boldsymbol{a}_i^{[t]}. \tag{4.6}$$

For the last position in the active set we allow all other inactive columns to compete. This is necessary if the algorithm needs to track changes in the support since the neighbor permutation strategy does not allow inactive columns to become active. Thus we perform the search for the last active column like in (4.4) but having all other columns as candidates,

$$\kappa = \arg\max_{j \in \mathcal{C}} \frac{|\boldsymbol{a}_j^{[t]^T} \boldsymbol{r}_{i-1}^{[t]}|^2}{\|\boldsymbol{a}_j^{[t]}\|_2^2}, \quad \text{with} \quad \mathcal{C} = \{m : n\}. \tag{4.7}$$

The same permutation strategy is used to move any possible better column to the active set and it is followed by the coefficient computation (4.5) and residual update (4.6).

After m non-zero positions are found and the coefficients $\boldsymbol{x}_{1:m}^{[t]}$ are computed, we continue with a cyclic optimization of each coefficient $i = 1 : m$. This is done by excluding one by one a coefficient i from the solution,

$$\tilde{\boldsymbol{r}}_m^{[t]} = \boldsymbol{r}_m^{[t]} + x_i \boldsymbol{a}_i^{[t]}, \tag{4.8}$$

and recomputing its value

$$\tilde{x}_i^{[t]} = \frac{\boldsymbol{a}_i^{[t]T} \tilde{\boldsymbol{r}}_m^{[t]}}{\|\boldsymbol{a}_i^{[t]}\|_2^2}. \tag{4.9}$$

Equation (4.8) is the reverse of (4.6). It removes the contribution of the coefficient x_i from diminishing the residual $\boldsymbol{r}_m^{[t]}$ and produces the partial residual $\tilde{\boldsymbol{r}}_m^{[t]}$. The coefficient update (4.9) can be viewed as a CD on the i'th coordinate since we can write

$$\tilde{x}_i^{[t]} = \frac{\boldsymbol{a}_i^{[t]^T} (\boldsymbol{r}_m^{[t]} + x_i \boldsymbol{a}_i^{[t]})}{\|\boldsymbol{a}_i^{[t]}\|_2^2} = x_i^{[t]} + \mu, \quad \text{with } \mu = \frac{\boldsymbol{a}_i^{[t]^T} \boldsymbol{r}_m^{[t]}}{\|\boldsymbol{a}_i^{[t]}\|_2^2}. \tag{4.10}$$

In (4.10) the coefficient $\tilde{x}_i^{[t]}$ is updated towards minimizing the residual in the context produced by all other computed coefficients while in (4.5) this is done having only the first $i-1$ coefficients included in the solution. Since we modify the coefficient value the residual also changes,

$$\boldsymbol{r}_m^{[t]} \leftarrow \tilde{\boldsymbol{r}}_m^{[t]} - \tilde{x}_i^{[t]} \boldsymbol{a}_i^{[t]} = \boldsymbol{r}_m^{[t]} - \mu \boldsymbol{a}_i^{[t]}. \tag{4.11}$$

The new coefficient value can thus be formally included in the solution

$$x_i^{[t]} \leftarrow \tilde{x}_i^{[t]}. \tag{4.12}$$

By cyclically repeating the CD for all $i = 1 : m$, the coefficients are updated towards minimizing the residual norm $\|\boldsymbol{r}_m^{[t]}\|_2$. It can be seen in (4.10) that the CD is governed by the scalar product $\boldsymbol{a}_i^{[t]^T} \boldsymbol{r}_m^{[t]}$ and the descent stops once $\boldsymbol{r}_m^{[t]}$ becomes orthogonal with $\boldsymbol{a}_i^{[t]}$. Thus the cyclic update is approximating an orthogonalization procedure.

An adaptive implementation needs to use the scalar products (4.1). For the selection criterion and coefficient computation we already have the required form. It is trivial to replace the vector products with the values stored by $\boldsymbol{\Psi}^{[t]}$ and $\boldsymbol{\phi}^{[t]}$. In case of the residual update, to produce the necessary scalar product form, we need to multiply the equation by $\mathbf{A}^{[t]^T}$ on the left. For example, instead of (4.6) we have, for $j = 1 : n$,

$$\boldsymbol{a}_j^{[t]^T} \boldsymbol{r}_i^{[t]} = \boldsymbol{a}_j^{[t]^T} \boldsymbol{r}_{i-1}^{[t]} - x_i^{[t]} \boldsymbol{a}_j^{[t]^T} \boldsymbol{a}_i^{[t]}. \tag{4.13}$$

This update must not overwrite the original scalar product values $\boldsymbol{\phi}^{[t]} = \mathbf{A}^{[t]^T} \boldsymbol{b}^{[t]}$ since they are needed[1] at time $t + 1$.

Using the cyclic update strategy two algorithms can be developed: the first named CAMP performs first a sweep in the spirit of AMP to find the support and then improves the solution by running a number of cyclic sweeps to update all coefficients. It follows the steps presented in Algorithm 4.1. The second, iterated CAMP (I-CAMP) improves the coefficients after the introduction of each new column in the active set to allow for a better support selection and is summarized in Algorithm 4.2. We present them using the vector notation for simplicity.

Online sparsity estimation

The CAMP algorithms presented so far make no assumptions of the sparsity level. They require knowledge of the number of columns m which may or may not be equal to the current true sparsity level $l_t^{[t]}$. To account for unknown sparsity levels or sparsity changes, we propose the use of ITC, namely PLS and BIC briefly presented in Chapter 3. They were first introduced for the GRLS algorithm [P5] but can be easily adapted for the sparsity estimation associated with any greedy method.

The idea behind the sparsity estimation is to run the algorithms for a number of columns m larger than the true sparsity $l_t^{[t]}$, followed by the search for a sparsity level estimate $l^{[t]}$. Given the fact that there may be no prior information about the true sparsity level, m may be chosen fixed but large enough to accommodate any possible true sparsity or it can be changed online such that it varies according to the current sparsity estimate. We propose a slow change

$$m^{[t]} = \begin{cases} m^{[t-1]} + 1 & \text{if} \quad m^{[t-1]} < l^{[t]} + \Delta \\ m^{[t-1]} & \text{if} \quad m^{[t-1]} = l^{[t]} \\ m^{[t-1]} - 1 & \text{if} \quad m^{[t-1]} > l^{[t]} + \Delta \end{cases}, \tag{4.14}$$

to filter out any large fluctuations that can result from infrequent but otherwise large error in the estimates, where $l^{[t]}$ is the current sparsity estimate. The constant Δ determines how far away we search from the current estimate. We continue by consider m constant in time, to ease the presentation. This does not interfere with the prior discussion; a variation in the value of m can easily be implemented by adding more columns to, or removing the last columns from the solution.

[1] We remind that $\boldsymbol{r}_0^{[t]} = \boldsymbol{b}^{[t]}$.

Algorithm 4.1: **Cyclic adaptive matching pursuit (CAMP)**

input: $\mathcal{A}^{[t-1]}$; $\mathbf{A}^{[t-1]}$; $\boldsymbol{b}^{[t-1]}$; $\beta^{[t]}$; $\boldsymbol{\alpha}^{[t]}$; m; n;
output: $\mathcal{A}^{[t]}$; $\mathbf{A}^{[t]}$; $\boldsymbol{b}^{[t]}$; $\boldsymbol{x}^{[t]}$;

0 Update $\mathbf{A}^{[t-1]}$ and $\boldsymbol{b}^{[t-1]}$ to include the new available data

1 For all positions $i = 1 : m - 1$

- 1.1 Find and permute a new column candidate to position i in the active set like in (4.4)
- 1.2 Estimate the value of the new coefficient x_i using (4.5)
- 1.3 Update the residual $\boldsymbol{r}_i^{[t]}$ like in (4.6)

2 For last position m

- 2.1 Find and permute a new column candidate for the last position in the active set like in (4.7)
- 2.2 Estimate the value of the new coefficient $x_i^{[t]}$ using (4.5)
- 2.3 Update the residual $\boldsymbol{r}_i^{[t]}$ like in (4.6)

3 Estimate the solution sparsity $l^{[t]} = \tilde{l}^{[t]}$ where $\tilde{l}^{[t]}$ is found using ITC, either PLS of BIC.

4 Remove the extra columns, $i = l^{[t]} + 1 : m$, from the solution similarly to (4.8)

5 Cyclically update the $l^{[t]}$-sparse solution to improve the value of the coefficients

Both ITC can be efficiently implemented and allow a recursive description. The PLS is naturally recursive in time for each possible sparsity level. The BIC explicitly requires the computation of the residual norms $\|\boldsymbol{r}_i^{[t]}\|_2$ for all allowed sparsity levels $1 < i \leq m$. Once they are available, they are directly plugged into (3.21).

Due to the scalar product implementation we do not explicitly store the long residual vectors. Thus, we need to compute the norm based on the available scalar products. To estimate the squared norm $\|\boldsymbol{r}_j^{[t]}\|_2^2 = \boldsymbol{r}_j^{[t]^T} \boldsymbol{r}_j^{[t]}$ of the residual for a solution $\boldsymbol{x}^{[t]}$ with a support cardinality j, we use the definition of the residual

$$\boldsymbol{r}_j^{[t]} = \boldsymbol{b}^{[t]} - \sum_{i=1}^{j} x_i^{[t]} \boldsymbol{a}_i^{[t]}, \tag{4.15}$$

and rewrite the squared norm,

$$\begin{aligned} \boldsymbol{r}_j^{[t]^T} \boldsymbol{r}_j^{[t]} &= \left(\boldsymbol{b}^{[t]} - \textstyle\sum_{i=1}^{j} x_i^{[t]} \boldsymbol{a}_i^{[t]}\right)^T \left(\boldsymbol{b}^{[t]} - \textstyle\sum_{i=1}^{j} x_i^{[t]} \boldsymbol{a}_i^{[t]}\right) \\ &= \boldsymbol{b}^{[t]^T} \boldsymbol{b}^{[t]} - 2 \textstyle\sum_{i=1}^{j} x_i^{[t]} \boldsymbol{b}^{[t]^T} \boldsymbol{a}_i^{[t]} \\ &\quad + \textstyle\sum_{i=1}^{j} \sum_{k=1}^{j} x_i^{[t]} x_k^{[t]} \boldsymbol{a}_i^{[t]^T} \boldsymbol{a}_k^{[t]}. \end{aligned} \tag{4.16}$$

For the iterated cyclic adaptive matching pursuit algorithm, since it computes

Algorithm 4.2: **Iterated cyclic adaptive matching pursuit (I-CAMP)**

input: $\mathcal{A}^{[t-1]}$; $\mathbf{A}^{[t-1]}$; $\boldsymbol{b}^{[t-1]}$; $\beta^{[t]}$; $\boldsymbol{\alpha}^{[t]}$; m; n;
output: $\mathcal{A}^{[t]}$; $\mathbf{A}^{[t]}$; $\boldsymbol{b}^{[t]}$; $\boldsymbol{x}^{[t]}$;

0 Update $\mathbf{A}^{[t-1]}$ and $\boldsymbol{b}^{[t-1]}$ to include the new available data

1 For all sparsity levels $i = 1 : m-1$

1.1 Find and permute a new column candidate to position i in the active set like in (4.4)

1.2 Estimate the value of the new coefficient $x_i^{[t]}$ using (4.5)

1.3 Update the residual $\boldsymbol{r}_i^{[t]}$ like in (4.6)

1.4 Cyclically update all coefficients $j = 1 : i$

1.5 Save the i sparse solution

2 For last position m

2.1 Find and permute a new column candidate for the last position in the active set like in (4.7)

2.2 Estimate the value of the new coefficient $x_m^{[t]}$ using (4.5)

2.3 Update the residual $\boldsymbol{r}_m^{[t]}$ like in (4.6)

1.4 Cyclically update all coefficients $j = 1 : m$

1.5 Save the m sparse solution

2 Select the $l^{[t]} = \tilde{l}^{[t]}$-sparse solution where the sparsity level estimate $\tilde{l}^{[t]}$ is given by ITC

different solutions for all sparsity levels smaller than m, we need to use (4.16) to compute the residual norms.

In the case of the simpler CAMP algorithm, (4.16) can be expressed[1] recursively as follows

$$\begin{aligned}
\boldsymbol{r}_j^{[t]^T}\boldsymbol{r}_j^{[t]} &= \left(\boldsymbol{b}^{[t]} - \boldsymbol{x}_{1:j}^{[t]^T}\boldsymbol{a}_{1:j}^{[t]}\right)^T\left(\boldsymbol{b}^{[t]} - \boldsymbol{x}_{1:j}^{[t]^T}\boldsymbol{a}_{1:j}^{[t]}\right) \\
&= \left(\boldsymbol{b}^{[t]} - \boldsymbol{x}_{1:j-1}^{[t]^T}\boldsymbol{a}_{1:j-1}^{[t]}\right)^T\left(\boldsymbol{b}^{[t]} - \boldsymbol{x}_{1:j}^{[t]^T}\boldsymbol{a}_{1:j}^{[t]}\right) \\
&\quad -x_j^{[t]}\boldsymbol{b}^{[t]^T}\boldsymbol{a}_j^{[t]} + x_j^{[t]}\boldsymbol{a}_j^{[t]^T}\boldsymbol{a}_{1:j}^{[t]}\boldsymbol{x}_{1:j}^{[t]^T} \\
&= \boldsymbol{r}_{j-1}^{[t]^T}\boldsymbol{r}_{j-1}^{[t]} - 2x_j^{[t]}\boldsymbol{b}^{[t]^T}\boldsymbol{a}_j^{[t]} + x_j^{[t]}\boldsymbol{a}_j^{[t]^T}\boldsymbol{a}_{1:j}^{[t]}\boldsymbol{x}_{1:j}^{[t]^T} \\
&\quad +x_j^{[t]}\boldsymbol{a}_j^{[t]^T}\boldsymbol{a}_{1:j-1}^{[t]}\boldsymbol{x}_{1:j-1}^{[t]^T},
\end{aligned} \tag{4.17}$$

since we have available only one solution, the m-sparse one. For both criteria we use directly the raw estimate given by (3.23, 3.21), namely $l^{[t]} = \tilde{l}^{[t]}$.

[1] We use a vector notation to be more concise.

Approximations and performance improvements

The main computational burden of the cyclic update algorithms is due to the update of all the scalar products $\mathbf{\Psi}^{[t]}$ from (4.3). They are, however, not all necessary when computing the solution, if the support is the same for all time instants t.

In a stationary regime, when the active set composition does not change (the column order may change inside the active set) as new samples are received, to compute the coefficients $\boldsymbol{x}_{1:m}$ like in (4.5) we only need the data associated with the current support:

- the active column norms $\|\boldsymbol{a}_i^{[t]}\|_2$ for $i = 1 : m$;
- the scalar products $\boldsymbol{a}_{1:m}^{[t]^T}\boldsymbol{r}_{i-1}^{[t]}$ for $i = 1 : m$.

To perform the search for the last position m of the support like in (4.7) we additionally need the norms $\|\boldsymbol{a}_j^{[t]}\|_2$, of all inactive columns $j > m$ together with the scalar products $\boldsymbol{a}_{m+1:n}^{[t]^T}\boldsymbol{r}_{m-1}^{[t]}$. This also allows for the exact computation of the last coefficient m like in (4.5).

The computation of the residuals $\boldsymbol{r}_{i-1}^{[t]}$ and of the associated scalar products $\mathbf{A}^{[t]^T}\boldsymbol{r}_{i-1}^{[t]}$ is more difficult. If we multiply (4.6) on the left with $\mathbf{A}^{[t]^T}$ to produce

$$\mathbf{A}^{[t]^T}\boldsymbol{r}_i^{[t]} = \mathbf{A}^{[t]^T}\boldsymbol{r}_{i-1}^{[t]} - x_i^{[t]}\mathbf{A}^{[t]^T}\boldsymbol{a}_i^{[t]}, \tag{4.18}$$

it can be seen that we need all scalar products $\mathbf{A}^{[t]^T}\boldsymbol{a}_i^{[t]}$ between the current column and the whole matrix $\mathbf{A}^{[t]}$. In this case, if we allow for any column to possible become active on position m, we need the whole scalar product matrix $\mathbf{\Psi}^{[t]}$, which is undesirable since the update (4.3) is costly.

To overcome this we only store and update the partial scalar product matrix

$$\tilde{\mathbf{\Psi}} = \mathbf{A}^{[t]^T}\boldsymbol{a}_{1:m}^{[t]} \tag{4.19}$$

with the products between the matrix $\mathbf{A}^{[t]}$ and the current columns that are active. Doing so ensures the exact computation of the solution in case the support does not change. Thus, in the ideal scenario where no support changes are made, storing and updating only $\tilde{\mathbf{\Psi}}^{[t]}$ instead of $\mathbf{\Psi}^{[t]}$ ensures that all the required data are available. If the support changes via the permutation of a column from the inactive set to last position m, we can still compute the coefficient $x_m^{[t]}$ but we are unable to update the residual due to the unknown scalar products $\mathbf{A}^{[t]^T}\boldsymbol{a}_m^{[t]}$.

Under our assumption, there are usually no sudden changes, either in the support or in the coefficient values. When a coefficient becomes inactive it does so gradually until the associated column reaches the last position m in the active set and is then replaced by the new column $\boldsymbol{a}_j^{[t]}$, with $j > m$. We can assume that, if $m > l^{[t]}$, any coefficient $x_i^{[t]}$ above the sparsity level $l^{[t]}$ $(i > l^{[t]})$ is small since the true solution contains all the relevant coefficients. When the last position in the support changes, the coefficient $x_m^{[t]}$ has a negligible influence in decreasing the

residual and the scalar products associated with the residual updates (4.6) can be approximated by

$$\mathbf{A}^{[t]^T}\boldsymbol{r}_m^{[t]} \approx \mathbf{A}^{[t]^T}\boldsymbol{r}_{m-1}^{[t]}. \tag{4.20}$$

This approximation can be computed without the unknown scalar products $\mathbf{A}^{[t]^T}\boldsymbol{a}_j^{[t]}$, $j > m$. When the new coefficient gradually become significant, if the unknown scalar products $\mathbf{A}^{[t]^T}\boldsymbol{a}_j^{[t]}$ are far from their true values, the performance may be negatively influenced even leading to instability. To circumvent this we propose to set them to zero and allow a number of updates to be performed before the column may be introduced into the active set. For this purpose we use a buffer $\mathcal{B}$ of length p (containing columns $\boldsymbol{a}_{m+1:m+p}^{[t]}$ since we work with permuted data) to delay the introduction of any new column in the active set. The scalar products are updated also for all the columns associated with the buffer, hence diminishing the approximation errors.

Thus, a new column $\boldsymbol{a}_j^{[t]}$ selected to be introduced in the active set on position m is handled differently:

- if it is inactive and does not belong to the buffer set, i.e. it is from $\mathcal{I} \setminus \mathcal{B}$, it replaces the last column $\boldsymbol{a}_{m+p}^{[t]}$ in $\mathcal{B}$ instead of being introduced directly in the active set $\mathcal{A}$; the associated scalar products are set to zero and the solution is still computed with the old column $\boldsymbol{a}_m^{[t]}$.
- if it belongs to $\mathcal{B}$, it is promoted one position in the set; it becomes active replacing $\boldsymbol{a}_m^{[t]}$ in the active set only when it is on the first position in $\mathcal{B}$.

This ensures that a certain column is selected at least p times before becoming active which, coupled with the update of its associated scalar products as new samples are received, reduces the approximation errors. All this changes do not alter the cyclical update procedure since they do not involve other data other than that associated with the active set. In [P4], only the approximate I-CAMP using the PLS criterion (I-CAMP-A) version is presented but a similar approach is possible for the CAMP algorithm.

4.1.3 Coordinate descent adaptive matching pursuit

The coordinate descent AMP (CD-AMP) algorithm [P1] completely reuses the past solution at current time, as opposed to the cyclic updates that compute the solution from scratch in the algorithms presented in the previous section. We describe the basic operation of the algorithm assuming that, at time $t-1$, we have computed an m-sparse solution $\boldsymbol{x}^{[t-1]}$ and we update it to contain the new data from time t. We use a similar permutation strategy for the data to move the active non-zero elements in the first m positions. Also, the algorithms are implemented using the scalar products representation but for simplicity we describe the algorithms using the vector notations.

The residual, corresponding to the solution $\boldsymbol{x}^{[t-1]}$ at time $t-1$, is

$$\boldsymbol{r}_m^{[t-1]} = \boldsymbol{b}^{[t-1]} - \sum_{i=1}^{m} x_i^{[t-1]}\boldsymbol{a}_i^{[t-1]}. \tag{4.21}$$

After receiving new data, the same old solution $\boldsymbol{x}^{[t-1]}$ produces a residual

$$\boldsymbol{r}_m^{[t]} = \boldsymbol{b}^{[t]} - \sum_{i=1}^{m} x_i^{[t-1]} \boldsymbol{a}_i = \begin{bmatrix} \boldsymbol{r}_m^{[t-1]} \\ \beta^{[t]} - \sum_{i=1}^{m} x_i^{[t-1]} \alpha_i^{[t]} \end{bmatrix}. \tag{4.22}$$

At time t we manage the order of the active positions and we update the values of the solution $\mathbf{x}^{[t]}$ associated with the support. The ordering of the active positions is performed similarly to the previous section. We use a column ordering scheme that produces a low variation speed of the support since we consider the support and the coefficient values to be slowly variable. Namely we use the same neighbor permutation scheme as before but we also show that this is not limiting the algorithm since other schemes (a sweep of any sorting algorithm based on permutations) may be used [P1].

The coefficient update is performed via a simple CD on each of the active positions, optimizing the residual corresponding to the m-sparse solution. The search for the support and the coefficient updates are intertwined and act successively on each active coefficient.

To decide the best of two columns i and j, with $i < j \leq m$, we compute a partial residual with the two positions i and j temporarily removed from the active set,

$$\tilde{\boldsymbol{r}}_m^{[t]} = \boldsymbol{r}_m^{[t]} + x_i^{[t-1]} \boldsymbol{a}_i^{[t]} + x_j^{[t-1]} \boldsymbol{a}_j^{[t]}. \tag{4.23}$$

The column that takes position i is decided by looking at the best alignment with this partial residual similarly to (4.4),

$$\kappa = \arg\max_{k \in \mathcal{C}} \frac{|\boldsymbol{a}_k^{[t]T} \tilde{\boldsymbol{r}}_m^{[t]}|^2}{\|\boldsymbol{a}_k^{[t]}\|_2^2}, \quad \text{with} \quad \mathcal{C} \in \{i, j\}. \tag{4.24}$$

If position j is better, then positions i and j are permuted. The optimization used to produce the coefficient $x_i^{[t]}$ is a CD on the LS criterion (2.1). This is similar to the cyclic updates (4.10) and is achieved by projecting the residual on the column $\boldsymbol{a}_i^{[t]}$ after excluding position i from the active set like in (4.8),

$$x_i^{[t]} = \frac{\boldsymbol{a}_i^{[t]^T} (\boldsymbol{r}_m^{[t]} + x_i^{[t-1]} \boldsymbol{a}_i^{[t]})}{\|\boldsymbol{a}_i^{[t]}\|_2^2} = x_i^{[t-1]} + \mu, \quad \text{with} \quad \mu = \frac{\boldsymbol{a}_i^{[t]^T} \boldsymbol{r}_m^{[t]}}{\|\boldsymbol{a}_i^{[t]}\|_2^2}. \tag{4.25}$$

This operation takes place immediately after the index in position i is found via (4.24). The residual is the updated to account for the modified coefficient,

$$\boldsymbol{r}_m^{[t]} \leftarrow \boldsymbol{r}_m^{[t]} - \mu \boldsymbol{a}_i^{[t]}. \tag{4.26}$$

For the last active column m, the search for the best column is again performed considering the entire inactive set, $\mathcal{C} = m : n$ like in (4.7) but using the partial residual

$$\tilde{\boldsymbol{r}}_m^{[t]} = \boldsymbol{r}_m^{[t]} + x_m^{[t-1]} \boldsymbol{a}_m^{[t]}. \tag{4.27}$$

This removes the old coefficient $x_m^{[t-1]}$ from the solution and allows any other column $\boldsymbol{a}_j^{[t]}$, $j > m$ to compete for the last position using the search criterion

Algorithm 4.3: **Coordinate descent adaptive matching pursuit (CD-AMP)**

input: $\mathcal{A}^{[t-1]}$; $\mathbf{A}^{[t-1]}$; $\boldsymbol{r}_m^{[t-1]}$; $\boldsymbol{x}^{[t-1]}$; $\beta^{[t]}$; $\boldsymbol{\alpha}^{[t]}$; m; n;
output: $\mathcal{A}^{[t]}$; $\mathbf{A}^{[t]}$; $\boldsymbol{r}_m^{[t]}$; $\boldsymbol{x}^{[t]}$;

0 Update $\mathbf{A}^{[t-1]}$ to include the new available data
Compute the residual $\boldsymbol{r}_m^{[t]}$ using the new available data like in (4.22)

1 For $i = 1 : m-1$

1.1 Find and permute a new column candidate to position i in the active set using (4.24)

1.2 Update the associated coefficient like in (4.25)

1.3 Update the residual $\boldsymbol{r}_m^{[t]}$ like in (4.26)

2 For last position $i = m$

2.1 Find and permute a new column candidate to the last position in the active set based on (4.24) and the partial residual (4.27); search over the set $\mathcal{C} = \{m : n\}$

2.2 Update the associated coefficient like in (4.28)

2.3 Update the residual $\boldsymbol{r}_m^{[t]}$ like in (4.29).

(4.24) together with the partial residual from (4.27). After the position is decided and the column is permuted into place, the optimal coefficient is recomputed

$$x_m^{[t]} = \frac{{\boldsymbol{a}_m^{[t]}}^T \tilde{\boldsymbol{r}}_m^{[t]}}{\|\boldsymbol{a}_m^{[t]}\|_2^2} \tag{4.28}$$

and the new residual updated

$$\boldsymbol{r}_m^{[t]} = \tilde{\boldsymbol{r}}_m^{[t]} - x_m^{[t]} \boldsymbol{a}_m^{[t]}. \tag{4.29}$$

Note that, if an inactive position becomes active, the influence of the former column m must be removed from the residual, which explains the use of $\tilde{\boldsymbol{r}}_m^{[t]}$ from (4.27) in the above formulas.

The main algorithm, called CAMP is shown in Algorithm 4.3 and follows closely the equations presented above. Figure 4.1 gives a graphical image of the operations performed. The support search is based on the neighbor permutation strategy. Each search and update operation is represented by a horizontal line with one or several ticks showing the columns that compete for the current position, marked with an arrow. The arrow also means an update operation. Active columns are represented by blue bullets, inactive columns by black bullets. It can be observed that in the active set only neighbors can exchange positions when we update the support. For the last position, we have all inactive columns as candidates. The numbers on the left belong to the steps from Algorithm 4.3. On the right, we remind that the residual is updated at each step.

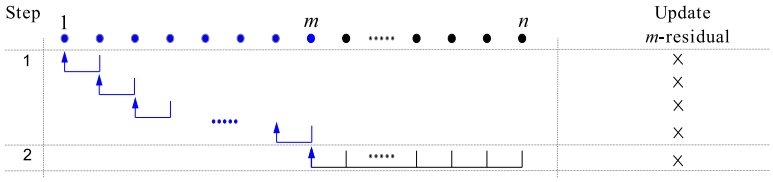


Figure 4.1: Graphical representation of the operations from the coordinate descent AMP algorithm.

Online sparsity estimation

A sparsity estimation that follows the same strategy as for the CAMP algorithms does not exploit well the recursive implementation of the CD strategy. While it is still possible to use the same method as before, we propose a new approach, the double residual CD-AMP (DCD-AMP) algorithm, that relies on the minimization of two residuals and does not require any cyclic updates.

We start by supposing that the number of columns m is large enough such that we can select all columns that contribute to the sparse solution. We require that m is larger than the true sparsity level which, when having a consistent sparsity level estimate $l^{[t]}$, can be generally achieved by setting it larger than this estimate, $m > l^{[t]}$ similarly to the case of the CAMP algorithms. This ensures that we have a large enough set of ordered columns for which to apply the ITC. Since this set is generally larger than the true sparsity, any solution computed using all the m columns is less accurate due to the columns that are not associated with the true support. This is not vital since it influences the sparsity estimation to a lesser degree.

However, to accurately compute the solution we introduce a second residual that is not affected by the extra columns and is associated with a support of size $l^{[t]}$. It is used to compute an approximation of the true solution and, since it relies on the sparsity estimate $l^{[t]}$, the approximation is good when the sparsity estimate is close to the true sparsity level. The computation of the two residuals follows the same strategy as for the CD-AMP algorithm with only a few changes. The two solutions overlap such that the first $l^{[t]}$ coefficients of the m-sparse solution are computed accurately using $\boldsymbol{r}_l^{[t]}$; that is, coefficients of the $l^{[t]}$-sparse solution are the first coefficients of the m-sparse one[1].

Although both the PLS and the BIC can be used, [P1] relies only on the use of the former. The raw estimate $\tilde{l}^{[t]}$ found using the ITC (3.21, 3.23) is not used directly. We set

$$l^{[t]} = l^{[t-1]} + \operatorname{sign}\left(\tilde{l}^{[t]} - l^{[t-1]}\right). \tag{4.30}$$

[1]Note that we drop the time $^{[t]}$ index from $l^{[t]}$ when we use the associated residual $\boldsymbol{r}_l^{[t]}$. Keeping it would make the notation cumbersome.

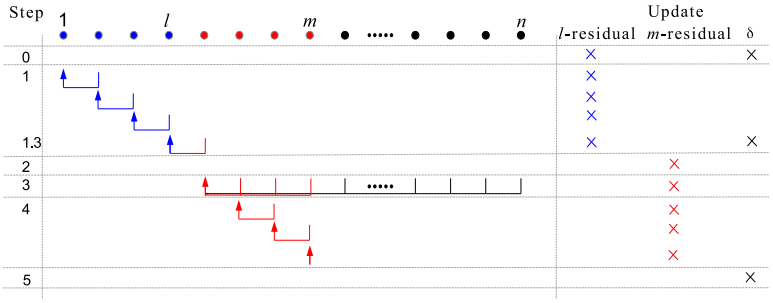


Figure 4.2: Graphical representation of the operations for the double residual coordinate adaptive matching pursuit algorithm.

It was found that allowing changes by 1 in the value of $l^{[t]}$ ensures smooth tracking.

The exact changes to the algorithm are presented below. The residual $\boldsymbol{r}_l^{[t]}$ is used for computing the coefficients on positions $1 : l^{[t]}$, while the residual $\boldsymbol{r}_m^{[t]}$ is used to estimate only the coefficients on positions $l^{[t]} + 2 : m$. The coefficient on position $l^{[t]} + 1$ is estimated using the residual $\boldsymbol{r}_l^{[t]}$, in order to provide a good value in case an increase of the sparsity level is required at the next time instant. However, not being included in the $l^{[t]}$-sparse solution, the $l^{[t]} + 1$ coefficient does not modify $\mathbf{r}_l^{[t]}$, it only affects $\boldsymbol{r}_m^{[t]}$. To ensure computational efficiency, we use the residual difference $\delta^{[t]} = \boldsymbol{r}_m^{[t]} - \boldsymbol{r}_l^{[t]}$ to avoid the update of both residuals when a coefficient is updated. Since the m-sparse solution includes the $l^{[t]}$-sparse solution, updating the $l^{[t]}$-sparse solution does not change the residual difference.

The positions $l^{[t]} + 1 : m$ are named pending (they belong to the set $\mathcal{P}$) since they do not contribute directly to the solution. The minimization of the residual $\boldsymbol{r}_l^{[t]}$ can be seen as the minimization of the LS criterion (2.1) based on the current estimate of the sparsity level. The modifications of the CD-AMP algorithm to work with $\boldsymbol{r}_l^{[t]}$ are minimal. In particular, the computations for positions $1 : l^{[t]} - 1$ are not changed while for position $l^{[t]}$ the only alternative candidate is the position $l^{[t]} + 1$. Since the column on position $l^{[t]} + 1$ belongs to the m-sparse solution, if we include it in the $l^{[t]}$-sparse solution and move the column $l^{[t]}$ to the m-sparse solution, we need to update the residual difference. Once the coefficient on position $l^{[t]}$ is chosen and computed, $\boldsymbol{r}_l^{[t]}$ is not modified anymore and we can reconstruct the residual $\boldsymbol{r}_m^{[t]}$ based on the residual difference.

For position $l^{[t]} + 1$, all pending and inactive positions are candidates. The updated residual $\boldsymbol{r}_l^{[t]}$ is used in the criterion (4.24), for the best column search, and also for computing the coefficient value. The updated coefficient modifies only $\boldsymbol{r}_m^{[t]}$ since the first $l^{[t]}$ coefficients were previously chosen and $\boldsymbol{r}_l^{[t]}$ was properly updated. If a new position $i > l^{[t]} + 1$ is found best, then the pending positions are shifted

Algorithm 4.4: **Double residual coordinate descent adaptive matching pursuit (DCD-AMP)**

input: $\mathcal{A}^{[t-1]}$; $\mathcal{P}^{[t-1]}$; $\mathbf{A}^{[t-1]}$; $\boldsymbol{r}_l^{[t-1]}$; $\delta^{[t-1]}$; $\boldsymbol{x}^{[t-1]}$; $\beta^{[t]}$; $\boldsymbol{\alpha}^{[t]}$; $l^{[t-1]}$; m; n;
output: $\mathcal{A}^{[t]}$; $\mathcal{P}^{[t]}$; $\mathbf{A}^{[t]}$; $\boldsymbol{r}_l^{[t]}$; $\delta^{[t]}$; $\boldsymbol{x}^{[t]}$; $l^{[t]}$;

0 Update $\mathbf{A}^{[t-1]}$ to include the new available data
Update the residual $\boldsymbol{r}_m^{[t]}$ and $\delta^{[t]}$ with the new data like in (4.22)

1 For $i = 1 : l^{[t-1]}$

1.1 Find new best column for position i; update the coefficient based on residual $\boldsymbol{r}_l^{[t]}$; perform neighbor permutations similarly to Algorithm 4.3, step 1

1.2 Update the residual $\boldsymbol{r}_l^{[t]}$

1.3 For position $i = l^{[t-1]}$, update $\delta^{[t]}$ if a permutation is performed

2 Compute the residual $\boldsymbol{r}_m^{[t]}$ using $\delta^{[t]}$ and the updated $\boldsymbol{r}_l^{[t]}$
$\boldsymbol{r}_m^{[t]} = \boldsymbol{r}_l^{[t]} + \delta^{[t]}$

3 For $i = l^{[t-1]} + 1$

3.1 Find new best columns for position i and the estimate coefficient based on the residual $\boldsymbol{r}_l^{[t]}$; perform a full search over the pending and inactive set similarly to Algorithm 4.3, step 2

3.2 Update the residual $\boldsymbol{r}_m^{[t]}$ instead of $\boldsymbol{r}_l^{[t]}$

4 For $i = l^{[t-1]} + 2 : m$

4.1 Find new best columns for position i; update coefficient based on residual $\boldsymbol{r}_m^{[t]}$; perform neighbor permutations similarly to Algorithm 4.3, step 1; for position $i = m$ no search is performed

4.2 Update the residual $\boldsymbol{r}_m^{[t]}$

5 Update the difference vector $\delta^{[t]} = \boldsymbol{r}_m^{[t]} - \boldsymbol{r}_l^{[t]}$

6 Select the $l^{[t]}$-sparse solution with $l^{[t]}$ given by (4.30)

- if $i \leq m$, then positions $l^{[t]} + 1 : i - 1$ move into $l^{[t]} + 2 : i$;

- if $i > m$, then positions $l^{[t]} + 1 : m - 1$ move into $l^{[t]} + 2 : m$, hence the former position m is discarded from the pending set.

The remaining coefficients on positions $l^{[t]} + 2 : m - 1$ are computed similarity to the CD-AMP algorithm but using and updating the residual $\boldsymbol{r}_m^{[t]}$. There is no open selection for position m, only the last coefficient is updated.

Finally, we estimate the current sparsity level $l^{[t]}$ using the PLS criterion. We use the same approach for choosing the parameter m. It is either fixed to a sufficiently large value, or is variable, being computed as $m = l^{[t]} + \Delta$, where Δ is a small positive integer constant. If the value $l^{[t]}$ is changed, then both residuals are changed in the variable m case (when m is also changed), but only $\boldsymbol{r}_l^{[t]}$ is

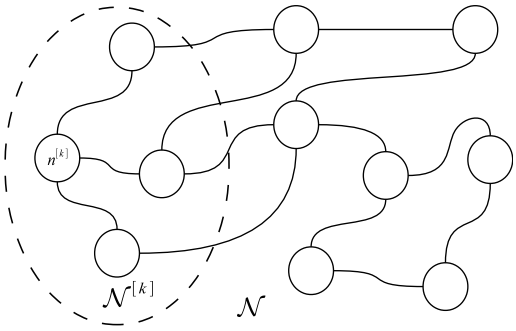


Figure 4.3: Example of a connected network $\mathcal{N}$ with a random topology $\mathcal{T}$. A node $n^{[k]}$ can communicate to neighbor nodes $\mathcal{N}^{[k]}$.

changed in the fixed m case. As a last step, the residual difference is stored after which a new sample can be processed.

A graphical representation is given in Figure 4.2 using the same conventions as in Figure 4.1; color red is reserved for the pending positions. The algorithm summary is listed as Algorithm 4.4. Again, the algorithm does not explicitly use the scalar products to keep the presentation clear.

Distributed coordinate descent adaptive matching pursuit

The CD algorithm can be adapted for a distributed scenario where, at the same time instant t, new data are available locally at a number of nodes. If we view the network as a graph with vertices between computing nodes that can communicate with each other, then a distributed algorithm generally achieves improved performance by exchanging information between these connected neighbor nodes and fusing it with the local measurements. To this regard, traditional methods proposed for distributed scenarios are generally based on LMS [82, 21] or on RLS [67, 20]. When sparsity is required, distributed algorithms have been developed to work either in batch mode [68] or to estimate the solution in an adaptive fashion (e.g. the sparsity-aware, projection based distributed algorithm from [26] or the sparse distributed LMS algorithm from [31]).

The distributed DCD-AMP (D-DCD-AMP) algorithm from [P2] extends the algorithms from the CD-AMP family to work in a distributed setup with limited inter-node communication. We use the coordinate descent strategy proposed by [P1] to compute in each node a local solution estimate (coefficient values, sparsity level and solution support) and we allow this data to be exchanged between neighbor nodes. Each node performs two tasks:

- first it combines the neighbor estimates with the local solution;

- then, performs an adaptation step to update the solution with newly avail-

able measurements.

The first task is achieved by averaging the neighbor solution coefficients with the local solution. This is a standard strategy and is employed in other distributed algorithms [26, 31]. To allow for all nodes to converge towards a common sparsity level, the neighbor sparsity level estimates influence the local sparsity level. The second task involves a CD on the direction given by the MP criterion [66] and is governed by a descent step size γ that allows for a tradeoff between the local adaptation speed and the convergence towards a common neighbor estimate. Additionally, to achieve fast convergence to a common solution, we account for the neighbor coefficient data when deciding the local support.

We study a static network $\mathcal{N}$ consisting of p nodes with a fixed topology $\mathcal{T}$. We consider the network to be connected, namely that there exist a path linking any two nodes $n^{[i]}$ and $n^{[j]}$. The true coefficient vector $\boldsymbol{x}^{[t]}$ is the same all over the network, $\boldsymbol{x}^{[t,i]} = \boldsymbol{x}^{[t,j]}$. An example of such a network is found in Figure 4.3. The distributed algorithm uses two sets of data:

- *The local data for each $n^{[k]}$:* The input data $\beta^{[t,k]}$ and output data $\boldsymbol{\alpha}^{[t,k]}$, different for each node.

- *The neighbor nodes data:* Since the bandwidth is limited, the node $n^{[k]}$ only receives from each neighbor node $n^{[j]}$, $j \in \mathcal{N}_k$ ($\mathcal{N}_k$ is the set of all nodes neighbor with $n^{[k]}$), data corresponding to the solution from time $t-1$:

  i) The sparsity level estimate $\tilde{l}^{[t-1,j]}$ computed using only the local data;

  ii) The $l^{[t-1,j]}$ coefficients of the current active solution $\boldsymbol{x}^{[t-1,j]}$;

  iii) The coefficient $l^{[t-1,j]}+1$, the best of the other coefficients not included in the current active solution;

  iv) The absolute coefficient positions.

We assume that each node computes $l^{[t,k]}$ coefficients associated with the local active positions by mixing using local and neighbor information. It can also independently compute $\tilde{l}^{[t,k]}$, the sparsity level estimate found locally using the information theoretic criteria, via ITC. Thus, the number $l^{[t,k]}$ of active positions computed using also neighbor data can differ from the raw $\tilde{l}^{[t,k]}$, $l^{[t,k]} \neq \tilde{l}^{[t,k]}$. The coefficient $l^{[t-1,j]}+1$ is transmitted to cover a possible increase of the sparsity estimate at the next time instant and also to allow better averaging and faster convergence in case it is actually in the solution support in other nodes. All other coefficients on positions $l^{[t-1,j]}+2:n$ of each neighbor solution $\boldsymbol{x}^{[t-1,j]}$ do not contribute directly to the solution and are implicitly considered 0.

Thus, after receiving the new data, our goal is to minimize a criterion similar to (2.1) but based on the local data $\mathbf{A}^{[t,k]}$ and $\boldsymbol{b}^{[t,k]}$,

$$J(\boldsymbol{x}^{[t,k]}) = \|\boldsymbol{b}^{[t,k]} - \mathbf{A}^{[t,k]}\boldsymbol{x}^{[t,k]}\|_2^2. \tag{4.31}$$

We also use the neighbors data to improve the solution and to force the estimates to converge towards a common value, as explained below.

To provide a reliable number $l^{[t,k]}$ of active positions we average the estimated sparsity level of the current node with the one received from its neighbors. To prevent large fluctuations, we allow only changes by at most one,

$$l^{[t,k]} = l^{[t-1,k]} + \operatorname{sign}\left(\left\lceil \frac{l^{[t,k]}+\sum_{j\in\mathcal{N}_k} \bar{l}^{[t-1,j]}}{|\mathcal{N}_k|+1} \right\rceil - l^{[t-1,k]}\right). \tag{4.32}$$

We compute the average coefficient estimate,

$$\bar{\boldsymbol{x}}^{[t-1,k]} = \frac{\boldsymbol{x}^{[t-1,k]} + \sum_{j\in\mathcal{N}_k} \boldsymbol{x}^{[t-1,j]}}{|\mathcal{N}_k|+1}. \tag{4.33}$$

For positions where the support differs we average with the implicit coefficient 0. Since we will use the average coefficient values, the residual update (4.22) associated with the $l^{[t,k]}$-sparse solution changes to

$$\boldsymbol{r}_l^{[t,k]} = \begin{bmatrix} \boldsymbol{r}_l^{[t-1,k]} + \sum_{i=1}^{l^{[t,k]}} \xi_i \boldsymbol{a}_i^{[t-1,k]} \\ \beta^{[t]} - \sum_{i=1}^{l^{[t,k]}} \bar{x}_i^{[t-1,k]} \alpha_i^{[t]} \end{bmatrix}, \tag{4.34}$$

where $\xi_i = x_i^{[t-1,k]} - \bar{x}_i^{[t-1,k]}$. Thus we replace the local coefficient values existing on the local support by averaging them with the neighbor data. This data, $\bar{\mathbf{x}}^{[t-1,k]}$, represents the new solution from time $t-1$ that has to be updated to include new available local measurements similarly to the update of $\boldsymbol{x}^{[t-1]}$ in the non distributed scenario. Thus, to keep the same notations we replace

$$\boldsymbol{x}^{[t-1,k]} \leftarrow \bar{\boldsymbol{x}}^{[t-1,k]} \tag{4.35}$$

and we update $\mathbf{x}^{[t-1,k]}$ as presented below.

The coefficient selection rule (4.24) is changed to use the neighbor coefficients and thus to improve the accuracy of finding the true support. A coefficient, received from one of the neighbor nodes $n^{[j]}$, has associated a partial residual $\tilde{\boldsymbol{r}}^{[t,j]}$ based on the node's local data. A selection criterion based on each $\tilde{\boldsymbol{r}}^{[t,j]}$ can be written as

$$\kappa = \arg\max_{i\in\mathcal{C}} \frac{|\boldsymbol{a}_i^{[t,k]T}\tilde{\boldsymbol{r}}^{[t,k]}|^2}{\|\boldsymbol{a}_i^{[t,k]}\|_2^2} + \sum_{j\in\mathcal{N}_k} \frac{|\boldsymbol{a}_i^{[t,j]T}\tilde{\boldsymbol{r}}^{[t,j]}|^2}{\|\boldsymbol{a}_i^{[t,j]}\|_2^2}. \tag{4.36}$$

The neighbor residuals $\tilde{\boldsymbol{r}}^{[t,j]}$ are not available locally and we substitute them with the available coefficient data. Since the underlying process generating the input data matrix is the same for all nodes, with $\|\boldsymbol{a}^{[t,k]}\|_2^2 \approx \|\boldsymbol{a}^{[t,j]}\|_2^2$, we approximate (see (4.25) and (4.28))

$$\frac{|\boldsymbol{a}^{[t,j]T}\tilde{\boldsymbol{r}}^{[t,j]}|^2}{\|\boldsymbol{a}^{[t,j]}\|_2^2} \approx \left(\boldsymbol{x}_l^{[t-1,j]}\right)^2 \|\boldsymbol{a}_l^{[t,k]}\|_2^2, \tag{4.37}$$

and incorporate the neighbor residual data into the local selection criterion,

$$\kappa = \arg\max_{l\in\mathcal{C}} \frac{|\boldsymbol{a}_l^{[t,k]T}\tilde{\boldsymbol{r}}^{[t,k]}|^2}{\|\boldsymbol{a}_l^{[t,k]}\|_2^2} + \sum_{j\in\mathcal{N}_k} \left(\boldsymbol{x}_l^{[t-1,j]}\right)^2 \|\boldsymbol{a}_l^{[t,k]}\|_2^2. \tag{4.38}$$

Algorithm 4.5: **Distributed double residual coordinate adaptive matching pursuit (D-DCD-AMP)**

local input: $\mathcal{A}^{[t-1,k]}$; $\mathcal{P}^{[t-1,k]}$; $\mathbf{A}^{[t-1,k]}$; $\boldsymbol{r}_l^{[t-1,k]}$; $\delta^{[t-1,k]}$; $\boldsymbol{x}^{[t-1,k]}$; $\beta^{[t,k]}$; $\boldsymbol{\alpha}^{[t,k]}$; $l^{[t-1,k]}$; m; n;
neighbor input: $\mathcal{A}^{[t-1,j]}$; $\mathcal{P}^{[t-1,j]}$; $\boldsymbol{x}^{[t-1,j]}$; $\tilde{l}^{[t-1,j]}$; $j \in \mathcal{N}_k$
output: $\mathcal{A}^{[t,k]}$; $\mathcal{P}^{[t,k]}$; $\mathbf{A}^{[t,k]}$; $\boldsymbol{r}_l^{[t,k]}$; $\delta^{[t,k]}$; $\boldsymbol{x}^{[t,k]}$; $\tilde{l}^{[t,k]}$; $l^{[t,k]}$

0 Update $\mathbf{A}^{[t-1]}$ to include the new available data

1 Compute the local $l^{[t,k]}$ from (4.32) using the neighbor data

2 Average the coefficients like in (4.33)
Update the residual $\boldsymbol{r}_l^{[t-1,k]}$ and $\delta^{[t-1,k]}$ to include the averaged coefficients and the new available data similarly to (4.34)

3 For $i = 1 : l^{[t,k]}$

- 3.1 Find new best column candidate for position i based on residual $\boldsymbol{r}_l^{[t,k]}$; perform neighbor permutations similarly to Algorithm 4.3, step 1 using the selection criterion (4.38)
- 3.2 Update the coefficient like in (4.40) and the residual $\boldsymbol{r}_l^{[t,k]}$ like in (4.39)
- 3.3 For position $i = l^{[t,k]}$, update $\delta^{[t,k]}$ similarly to (4.40) if a permutation is performed

4 Compute the residual $\boldsymbol{r}_m^{[t,k]}$ using $\delta^{[t,k]}$ and the updated $\boldsymbol{r}_l^{[t,k]}$
$\boldsymbol{r}_m^{[t,k]} = \boldsymbol{r}_l^{[t,k]} + \delta^{[t,k]}$

5 For position $i = l^{[t,k]} + 1$

- 5.1 Find new best column candidate for position i based on residual $\boldsymbol{r}_l^{[t,k]}$; perform the full search over the pending and inactive set similarly to Algorithm 4.3, step 2 using the selection criterion (4.38)
- 5.2 Compute coefficient on position $l^{[t,k]} + 1$ using $\boldsymbol{r}^{[t,k]}$ but modify the residual $\boldsymbol{r}_m^{[t,k]}$

6 For $i = l^{[t,k]} + 2 : m$

- 6.1 Find new best columns for position i; update coefficient based on residual $\boldsymbol{r}_m^{[t,k]}$; perform neighbor permutations similarly to Algorithm 4.3, step 1 using the selection criterion (4.38); for position $i = m$ no search is performed
- 6.2 Update the residual $\boldsymbol{r}_m^{[t,k]}$

7 Update the difference vector $\delta^{[t,k]} = \boldsymbol{r}_m^{[t,k]} - \boldsymbol{r}_l^{[t,k]}$

8 Estimate the new sparsity level $\tilde{l}^{[t,k]}$ using the local data

We also change the update rule (4.25) to

$$x_i^{[t,k]} = x_i^{[t-1,k]} + \gamma\mu, \tag{4.39}$$

creating a slower CD governed by the descent step $\gamma \in (0, 1]$. A smaller value of γ means a lower influence of the local data. This allows the algorithm to remain close

to the mean coefficient values computed from neighbor data but also allows the adaptation to take advantage of new local data. The residual update is modified in a similar fashion,

$$\boldsymbol{r}^{[t,k]} \leftarrow \boldsymbol{r}^{[t,k]} - \gamma\mu\boldsymbol{a}_i^{[t,k]}. \tag{4.40}$$

For the additional coefficient $l^{[t,k]}+1$ we use the same averaging strategy (4.33) and the updated residual $\boldsymbol{r}^{[t,k]}$; the selection criterion (4.38) is used for all column searches to introduce a preference for positions existing in neighbor nodes. For the other positions $l^{[t,k]}+2:m$, since there is a low probability that the columns are in the active set of the neighbor nodes, we use the full length CD, with $\gamma=1$, to compute their associated coefficients. This achieves the fastest convergence given the local data. An overview of the full distributed algorithm presenting the main steps is available in Algorithm 4.5.

4.2 Greedy sparse orthogonal algorithms

The straightforward use of the batch OLS in an adaptive scenario is prohibitive due to its huge complexity. If we assume that the algorithm selects the same support all the time, transforming the batch method to work recursively and reuse the past solution is relatively simple. Having performed the orthogonalization for m columns at time $t-1$, the new data available at time t can be directly appended to the matrices from (3.14), namely to the block upper triangular input matrix $\mathbf{R}^{[t-1]}$ and corresponding output data vector $\boldsymbol{b}^{[t-1]}$, to produce the arrow like structure

$$\mathbf{R}^{[t]} = \begin{bmatrix} \sqrt{\lambda}\begin{bmatrix} \mathbf{R}_{\mathcal{A},\mathcal{A}}^{[t-1]} & \mathbf{R}_{\mathcal{A},\mathcal{I}}^{[t-1]} \\ \mathbf{0} & \mathbf{F}^{[t-1]} \end{bmatrix} \\ \boldsymbol{\alpha}^{[t]^T} \end{bmatrix} \quad \text{and} \quad \boldsymbol{b}^{[t]} = \begin{bmatrix} \sqrt{\lambda}\begin{bmatrix} \boldsymbol{b}_{\mathcal{A}}^{[t-1]} \\ \boldsymbol{g}^{[t-1]} \end{bmatrix} \\ \beta^{[t]} \end{bmatrix}. \tag{4.41}$$

To restore the upper triangular structure a simple procedure that uses Givens rotations is used.

If the support changes however, the restoration of the triangular form becomes very computationally intensive especially if inactive columns are selected to be included in the active set. In this case the triangular form is completely destroyed to the right of the insertion point. We describe in what follows an efficient recursive algorithm derived from the batch OLS, namely the GRLS algorithm. The development of the method started with [35], which proposed a first version, and it was later improved to work with the online estimation of the sparsity in [P5] and thus it provides a robust solver for the sparsity estimation problem. Both [35] and [P5] work for an exponentially windowed input data, however, for low sparsity levels a sliding window may provide the same performance with lower computational complexity by using an efficient implementation [P6]. The algorithm can be also adapted for a group sparsity problem where the non-zero coefficients are grouped together. With this a priori information available, a group level column selection was developed in [P7] and different group sparsity level selection criteria were proposed and analyzed.

4.2.1 Greedy sparse recursive least squares

The GRLS algorithm is an adaptive method able to produce a sparse solution which, at each time t, is in principle equivalent to the OLS output from Algorithm 3.2. We implicitly maintain the same column reshuffling strategy of the OLS to bring the support column on the first positions in the input data matrix. The only difference is that, in order to achieve a low complexity, a different support selection strategy is generally used. This is the same neighbor permutation[1] strategy that was used for the CD-AMP family of algorithms. After a sufficient amount of data are available, the permutation is usually the same as the one given by OLS, in the sense that it selects the same active columns. However, due to the selection strategy used the adaptive algorithm is less prone to errors in the support since it allows for a lower variability of the support.

In order to restore the upper triangular matrix factorization from (3.14) after the new data are appended like in (4.41), the algorithm needs to update, at each t, the first m rows of the input and output matrices, precisely the matrix

$$\mathbf{R}_{\mathcal{A}}^{[t]} = \sqrt{\lambda}\begin{bmatrix}\mathbf{R}_{\mathcal{A},\mathcal{A}}^{[t-1]} & \mathbf{R}_{\mathcal{A},\mathcal{I}}^{[t-1]}\end{bmatrix}, \quad \text{with} \quad \mathbf{R}_{\mathcal{A}}^{[t]} \in \mathbb{R}^{m\times n}, \tag{4.42}$$

and the vector $\boldsymbol{b}_{\mathcal{A}}^{[t]} \in \mathbb{R}^m$, while introducing zeros on the first m positions in the newly appended data. Since this information is associated with present active columns and it is used to compute the current LS solution,

$$\boldsymbol{x}_{\mathcal{A}} = (\mathbf{R}_{\mathcal{A},\mathcal{A}}^{[t]})^{-1}\boldsymbol{b}_{\mathcal{A}}^{[t]}, \tag{4.43}$$

it is needed explicitly. The past information of the other inactive columns, stored by $\mathbf{F}^{[t]}$ and $\boldsymbol{g}^{[t]}$, is needed only in the case the active columns change. It can not be stored explicitly however, since it grows indefinitely with t. We replace it with the fixed size scalar product matrices $\boldsymbol{\Psi}^{[t]} \in \mathbb{R}^{n-m\times n-m}$ and $\boldsymbol{\phi}^{[t]} \in \mathbb{R}^{n-m}$,

$$\boldsymbol{\Psi}^{[t]} = {\mathbf{F}^{[t]}}^T\mathbf{F}^{[t]} \quad \text{and} \quad \boldsymbol{\phi}^{[t]} = {\mathbf{F}^{[t]}}^T\boldsymbol{g}^{[t]}. \tag{4.44}$$

This is similar to the use of scalar products for the algorithms derived from the MP presented in the previous subsection. We remind that $\boldsymbol{\Psi}^{[t]}$ is a symmetric matrix, hence only its upper triangle has to be computed and stored. We update the full matrix in the description of the algorithms to allow for an easier presentation but we count only the relevant operations when we estimate the algorithm complexity. By an abuse of notation we denote henceforward with $\mathbf{R}^{[t]}$ and $\boldsymbol{b}^{[t]}$ only the present data, i.e. we drop the index $\mathcal{A}$ used in (4.42) and for $\boldsymbol{b}_{\mathcal{A}}^{[t]}$. The information presented so far, namely $\mathbf{R}^{[t]}$, $\boldsymbol{b}^{[t]}$, $\boldsymbol{\Psi}^{[t]}$, $\boldsymbol{\phi}^{[t]}$, is required for the solution computation.

Upon receiving new data $\boldsymbol{\alpha}^{[t]}$ and $\beta^{[t]}$, if we impose constrains for the support selection, the GRLS data can be maintained with relatively low complexity. Allowing a potential complete support change between two time instants $t-1$ and t, like

[1] Chronologically, the permutation strategy was first introduced for the GRLS algorithm. While for the CD-AMP algorithms other permutation based, ordering strategies can be used, for the GRLS algorithms only neighbor permutation ensures a low complexity.

in the case of applying the batch OLS algorithm from scratch two times for the two corresponding data sets, is destructive for the triangular form of $\mathbf{R}^{[t]}$ and renders the past triangular form useless since the arbitrary reshuffle of the active columns generally requires a full triangularization. Furthermore, it is unlikely that the solution changes drastically from one iteration to the other. Due to the windowing, the algorithm can react only after the past data has been sufficiently forgotten in case of sharp changes of the support.

Thus, we rely on the slow time variability of the solution support and replace the full search done by the OLS with the neighbor permutation strategy:

- for each position $k = 1 : m-1$ of the active set, only columns i and $i+1$ compete;
- for the last active position m all columns m to n are allowed to compete.

Basic update

When the active set does not change, the necessary operations are those of a typical orthogonal triangularization update. Actually, the same operations are employed every time since the new data can be added and the triangular form restored before the new column order has to be decided. This is due to the fact that we employ only orthogonal transforms and, when they multiply several vectors, they do not modify the scalar products (Property A.3.5).

The arrow like structure from (4.41) is updated to produce a triangular form in the first m columns by zeroing the elements of the last row via Givens rotations. The data matrices are not stored in full and we only work with present data $\mathbf{R}^{[t-1]}$ and $\boldsymbol{b}^{[t-1]}$, to which we append the new data available at time t,

$$\mathbf{R}^{[t]} = \begin{bmatrix} \sqrt{\lambda}\mathbf{R}^{[t-1]}_{\mathcal{A},\mathcal{A}} & \sqrt{\lambda}\mathbf{R}^{[t-1]}_{\mathcal{A},\mathcal{I}} \\ \boldsymbol{\alpha}^{[t]^T}_{\mathcal{A}} & \boldsymbol{\alpha}^{[t]^T}_{\mathcal{I}} \end{bmatrix} \quad \text{and} \quad \boldsymbol{b}^{[t]} = \begin{bmatrix} \sqrt{\lambda}\boldsymbol{b}^{[t-1]}_{\mathcal{A}} \\ \beta^{[t]} \end{bmatrix}. \tag{4.45}$$

This does not introduce any limitations since we operate only with orthogonal transforms and we can update $\boldsymbol{\Psi}^{[t]}$ and $\boldsymbol{\phi}^{[t]}$ (Property A.3.6). Notice that we implicitly account for the forgetting factor in the scalar products

$$\begin{aligned} \boldsymbol{\Psi}^{[t]} &= \lambda\boldsymbol{\Psi}^{[t-1]} = \lambda\mathbf{F}^{[t-1]^T}\mathbf{F}^{[t-1]}, \\ \boldsymbol{\phi}^{[t]} &= \lambda\boldsymbol{\phi}^{[t-1]} = \lambda\mathbf{F}^{[t-1]^T}\boldsymbol{g}^{[t-1]}. \end{aligned} \tag{4.46}$$

After zeroing the first m elements on the last column, the remaining transformed elements of the new row are moved into the past by updating the associated scalar products and are not stored explicitly. Thus the last row from $\mathbf{R}^{[t]}$ and element from $\boldsymbol{b}^{[t]}$ can be formally deleted. The operations are summarized in Algorithm 4.6.

Neighbor permutation

Appending the new data available at time t and restoring the triangular form does not change the decision associated with the support selection. Thus we describe

Algorithm 4.6: **Greedy recursive least squares (GRLS) - basic update**

input: $\mathbf{R}^{[t-1]}$; $\boldsymbol{b}^{[t-1]}$; $\boldsymbol{\Psi}^{[t-1]}$, $\boldsymbol{\phi}^{[t-1]}$; $\boldsymbol{\alpha}^{[t]}$; $\beta^{[t]}$; m; n;
output: $\mathbf{R}^{[t]}$; $\boldsymbol{b}^{[t]}$; $\boldsymbol{\Psi}^{[t]}$; $\boldsymbol{\phi}^{[t]}$;

0 Append current data like in (4.41)

1 For $i = 1 : m$ restore the upper triangular by zeroing each position i on row $m + 1$ of $\mathbf{R}^{[t]}$

  2.1 Compute the Givens rotation $\mathbf{G}_i$ that zeros $\mathbf{R}^{[t]}_{m+1,i}$ using $\mathbf{R}^{[t]}_{i,i}$ (Theorem A.3.2)

  2.2 Apply the rotation on $\mathbf{R}^{[t]}$, $\mathbf{R}^{[t]} \leftarrow \mathbf{G}_i\mathbf{R}^{[t]}$

  2.3 Apply the rotation on $\boldsymbol{b}^{[t]}$, $\boldsymbol{b}^{[t]} \leftarrow \mathbf{G}_i\boldsymbol{b}^{[t]}$

3 Update the past by exponentially weighting the scalar products and include the remaining non-zeroed data of the appended row
$\boldsymbol{\Psi}^{[t]} \leftarrow \lambda\boldsymbol{\Psi}^{[t-1]} + \mathbf{R}^{[t]}_{m+1,m+1:n}\mathbf{R}^{[t]^T}_{m+1,m+1:n}$,
$\boldsymbol{\phi}^{[t]} \leftarrow \lambda\boldsymbol{\phi}^{[t-1]} + \boldsymbol{b}_{m+1}\mathbf{R}^{[t]}_{m+1,m+1:n}$

4 Delete row $m + 1$ from $\mathbf{R}^{[t]}$ and $\boldsymbol{b}^{[t]}$

the neighbor permutation strategy based on the data updated according to the re-triangularization from Algorithm 4.6.

At iteration i, instead of choosing the best next column among all columns $i : n$, we restrain our search to columns i and $i + 1$. The column selection test is the restrained version of (3.15) and the best candidate column κ is given by

$$\kappa = \arg \max_{j \in \{i,i+1\}} \frac{|\sum_{k=i}^{i+1} \mathbf{R}_{k,j}\boldsymbol{b}_k|}{\sqrt{\sum_{k=i}^{i+1} \mathbf{R}^2_{k,j}}}. \tag{4.47}$$

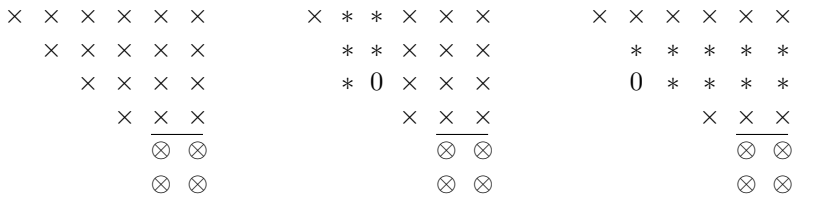
Since the matrix $\mathbf{R}^{[t]}$ contains zeros below the diagonal, columns i and $i + 1$ have only one and two, respectively, relevant non-zero elements on and below row i. This translates to selecting to permute the neighbors if

$$|\boldsymbol{b}_i| < \frac{|\mathbf{R}_{i,i+1}\boldsymbol{b}_i + \mathbf{R}_{i+1,i+1}\boldsymbol{b}_{i+1}|}{\sqrt{\mathbf{R}^2_{i,i+1} + \mathbf{R}^2_{i+1,i+1}}}. \tag{4.48}$$

A permutation introduces only a sub-diagonal element that can be eliminated with a single Givens rotation. The situation is illustrated in Figure 4.4. The whole permutation strategy is presented in Algorithm 4.7.

Last column search

After the neighbor permutations are performed, the last active column is selected like in the OLS algorithm. However, for the inactive columns, found in positions $m + 1 : n$, the matrix $\mathbf{R}^{[t]}$ does not contain all necessary information. The past



(a) (b) (c)

Figure 4.4: Matrix $\mathbf{R}^{[t]}$ illustrating the operations from Algorithm 4.7 for $m = 4$, $n = 6$; a permutation between the second and the third column is performed; the non-zero elements are represented by $\times$ and those modified, to a non-zero value by the current operation, by $*$; we mark with $\otimes$ the past data not stored explicitly to remind that the matrix $\mathbf{R}^{[t]}$ does not contain the whole input data; the figures show the initial matrix (Figure 4.4a), the matrix after the permutations (Figure 4.4b) and after the application of the Givens rotation (Figure 4.4c).

Algorithm 4.7: **Greedy recursive least squares (GRLS) - neighbor permutation**

---

input: $\mathbf{R}^{[t-1]}$; $\boldsymbol{b}^{[t-1]}$; $\boldsymbol{\alpha}^{[t]}$; $\beta^{[t]}$; m; n;
output: $\mathbf{R}^{[t]}$; $\boldsymbol{b}^{[t]}$;

1 For $i = 1 : m-1$, if column $i+1$ is found better than i by (4.47)

1.1 Swap columns i and $i+1$ in $\mathbf{R}^{[t]}$

1.2 Compute Givens rotation $\mathbf{G}_i$ that zeroes $\mathbf{R}^{[t]}_{i+1,i}$ using $\mathbf{R}^{[t]}_{i,i}$ (Theorem A.3.2)

1.3 Apply the rotation to $\mathbf{R}^{[t]}$, $\mathbf{R}^{[t]} \leftarrow \mathbf{G}_i\mathbf{R}^{[t]}$

1.4 Apply the rotation to $\boldsymbol{b}^{[t]}$, $\boldsymbol{b}^{[t]} \leftarrow \mathbf{G}_i\boldsymbol{b}^{[t]}$

---

information is needed for the search of the best column. The selection criterion is given by

$$\kappa = \arg \max_{j \in \{m:n\}} \begin{cases} |\boldsymbol{b}_m^{[t]}| & \text{if} \quad j = m \\ \frac{|\mathbf{R}^{[t]}_{m,j}\boldsymbol{b}_m^{[t]} + \boldsymbol{\phi}^{[t]}_{j-m}|}{\sqrt{\mathbf{R}^{[t]^2}_{m,j} + \boldsymbol{\Psi}^{[t]}_{j-m,j-m}}} & \text{if} \quad j > m. \end{cases} \tag{4.49}$$

This means that performing the permutation and restoring the triangular form as presented in Figure 4.5 is done only if

$$|\boldsymbol{b}_m^{[t]}| < \max_{j \in \{m+1:n\}} \frac{|\mathbf{R}^{[t]}_{m,j}\boldsymbol{b}_m^{[t]} + \boldsymbol{\phi}^{[t]}_{j-m}|}{\sqrt{\mathbf{R}^{[t]^2}_{m,j} + \boldsymbol{\Psi}^{[t]}_{j-m,j-m}}}. \tag{4.50}$$

The form of the equation (4.49) results from (3.15) by properly replacing the past data with the corresponding scalar products. Note that $\boldsymbol{\Psi}^{[t]}$ and $\boldsymbol{\phi}^{[t]}$ are defined only for the inactive columns, hence index $j-m$ corresponds to column j; also for

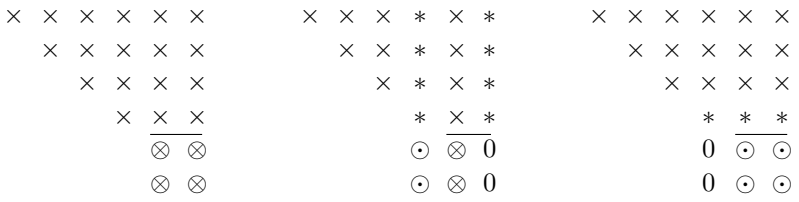


Figure 4.5: Matrix $\mathbf{R}^{[t]}$ illustrating the operations from Algorithm 4.8 for $m = 4$, $n = 6$; a permutation between the forth and the sixth column is performed; the non-zero elements are represented by $\times$ and those modified, to a non-zero value by the current operation, by $*$; we mark with $\otimes$ the past data not stored explicitly and with $\odot$ the past data modified, to a non-zero value by the current operation, to remind that the matrix $\mathbf{R}^{[t]}$ does not contain the whole input data; the figures show the initial matrix (Figure 4.5a), the matrix after the permutations (Figure 4.5b) and after the application of the Householder transform (Figure 4.5c); note that the past data are modified which justifies the update of the scalar products.

column m the past data are already zeroed and since no scalar products are used the expression is much simpler in (4.49).

A summary of the operations is presented in Algorithm 4.8 and a visual aid to the modifications that occur in $\mathbf{R}^{[t]}$ is included in Figure 4.5. If column m is still the best, there is nothing more to be done. If the inactive column $\kappa > m$ is better, it has to be permuted to position m and then zeroed below the diagonal, using a Householder reflector. However, since the past of the column is not stored explicitly, we operate with the scalar products instead. This is possible because the Householder transform used, like in theorem A.3.1, only require the scalar products of the indefinite long vectors. Coupled with the observation that any orthogonal transform does not change the scalar product values when it is applied, properties (A.3.5, A.3.6), the update presented in Algorithm 4.8 is straightforward.

Online sparsity estimation and the complete algorithm

Knowing a priori the number of non-zero elements m is generally quite restrictive. We assume that m is an upper bound for the maximum number of non-zero elements similarly to the algorithms described in the previous sections. If this bound is good, in particular equal to the true value, the behavior of greedy recursive least squares is very good; however, a too large m can degrade the performance.

It is reasonable to assume that for slowly time varying processes and after a sufficiently long transitory regime, the first elements of the active set given by the algorithm are the true non-zero positions. Thus, since a good guess for the sparsity level estimate $l^{[t]}$ is usually not available a priori and may change in time, we select m large enough to accommodate any possible sparsity level and find the estimate online using information theoretic criteria. We keep the same strategy for changing m like we did for the CAMP and CD-AMP algorithms. Namely, we

Algorithm 4.8: **Greedy recursive least squares (GRLS) - last active column selection**

input: $\mathbf{R}^{[t-1]}$; $\boldsymbol{b}^{[t-1]}$; $\boldsymbol{\Psi}^{[t]}$; $\boldsymbol{\phi}^{[t]}$; $\boldsymbol{\alpha}^{[t]}$; $\beta^{[t]}$; m; n;
output: $\mathbf{R}^{[t]}$; $\boldsymbol{b}^{[t]}$; $\boldsymbol{\Psi}^{[t]}$; $\boldsymbol{\phi}^{[t]}$;

1 Find the best column κ like in (4.49)

2 If the new column is found better i.e. (4.50) is true

2.1 Swap columns m and κ of $\mathbf{R}^{[t]}$.

2.2 Save the partial scalar product data
$\tilde{\boldsymbol{\phi}}^{[t]} = \mathbf{R}^{[t]}_{m,m+1:n}\boldsymbol{b}^{[t]}_{m}$
$\tilde{\boldsymbol{\Psi}}^{[t]} = \mathbf{R}^{[t]^T}_{m,m+1:n}\mathbf{R}^{[t]}_{m,m+1:n}$

2.3 Compute the Householder reflector $\mathbf{H}$ that zeros column m in $\mathbf{R}^{[t]}$ below the diagonal (Theorem A.3.1)

2.4 Apply the $\mathbf{H}$ to $\mathbf{R}^{[t]}$ and $\boldsymbol{b}^{[t]}$ (Property A.3.2)

2.5 Update the scalar products (Property A.3.6)
$\boldsymbol{\phi}^{[t]} \leftarrow \boldsymbol{\phi}^{[t]} + \tilde{\boldsymbol{\phi}}^{[t]} - \mathbf{R}^{[t]}_{m,m+1:n}\boldsymbol{b}_{m}$
$\boldsymbol{\Psi}^{[t]} \leftarrow \boldsymbol{\Psi}^{[t]} + \tilde{\boldsymbol{\Psi}}^{[t]} - \mathbf{R}^{[t]^T}_{m,m+1:n}\mathbf{R}_{m,m+1:n}$

propose two choices for the value of m. We can either have a fixed value m chosen such that is greater than the true sparsity level or allow it to vary such that it is linked to the estimated sparsity level like in (4.14). Let us note here that a change of $m^{[t-1]}$ by 1 to produce $m^{[t]}$ can be easily incorporated in the algorithm.

Decreasing $m^{[t-1]}$ means simply deleting row $m^{[t-1]}$ of $\mathbf{R}^{[t]}$ and $\boldsymbol{b}^{[t]}$, but not before using it for updating the past information, similarly to the last step in Algorithm 4.6. Increasing $m^{[t-1]}$ can be done by running an algorithm similar with Algorithm 4.8 for the selection of an extra column active column. We continue the discussion as we did before only for the fixed m since there are no major differences for the case when it varies.

The BIC can be easily computed in the context of the greedy recursive algorithm, taking advantage of the properties of the orthogonal triangularization. The residual of any i-sparse solution has the same norm as the vector $\boldsymbol{b}^{[t]}$ (after applying in place all the orthogonal transforms $\mathbf{Q}^{[t]}$ that produce the upper triangular form in $\mathbf{R}^{[t]}$) with the first i elements removed. With the notation from (4.42) the residual $\boldsymbol{r}_i^{[t]}$ is defined by the lower part of the vector $\boldsymbol{b}^{[t]}$ that is not stored explicitly after the orthogonalization procedure is performed for i columns. It results that the residual $\boldsymbol{r}_i^{[t]}$ is given by applying the inverse orthogonal transform $\mathbf{Q}^{[t]}$ on

$\boldsymbol{g}^{[t]}$,

$$\boldsymbol{r}_i^{[t]} = \mathbf{Q}^{[t]} \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \boldsymbol{g}^{[t]} \end{bmatrix}. \tag{4.51}$$

Since we only need the residual norm in (3.21), not having $\boldsymbol{g}^{[t]}$ stored explicitly does not cause any problems. The squared norm of any full vector $\boldsymbol{c}$ can be computed with the recursion in time

$$\|\boldsymbol{c}^{[t]}\|_2^2 = \lambda \|\boldsymbol{c}^{[t-1]}\|_2^2 + {c_t^{[t]}}^2; \tag{4.52}$$

here we consider the data to be exponentially weighted with the use of the forgetting factor λ. Since the orthogonal transforms do not change the norm when they are applied to a vector (Property A.3.5), it results that $\|{\mathbf{Q}^{[t]}}^T \boldsymbol{c}^{[t]}\|_2^2 = \|\boldsymbol{c}^{[t]}\|_2^2$.

Thus, at time t, the squared norm of the residual (4.51) produced by the i-sparse solution can be computed as

$$\|\boldsymbol{r}^{[t]}\|_2^2 = \|\boldsymbol{b}^{[t]}\|_2^2 - \sum_{k=1}^{j} {b_k^{[t]}}^2, \tag{4.53}$$

where $\boldsymbol{b}^{[t]}$ is the vector from (4.41), as updated by the algorithm at time t. We note that (4.53) can be used to compute the residual norms recursively for all $j = 1 : m$.

For the PLS, the computation of the a priori errors needed by (3.23) requires the algorithm to find all solutions of sparsity levels $j = 1 : m$ by solving upper triangular systems of equations. An efficient recursive implementation is present in [P5, Appendix II]. Note that if at time t we increase $m^{[t]} = m^{[t-1]} + 1$, there are no past a priori errors produced by the $m^{[t]}$-sparse solution and we replace them with the errors of the $m^{[t-1]} - 1$-sparse solution. The criterion tends to its true value, due to the effect of the forgetting factor. Also, an implicit approximation is made at neighbor permutations, where the past values are preserved, although the new models are actually different. Despite these approximations, the PLS criterion works well in practice.

The structure of the overall algorithm is shown in Algorithm 4.9. It combines the basic update and the column selection strategies.

4.2.2 Sliding window algorithm

Until now we considered the data to be exponentially windowed such that the older information has a decreasing influence on the results. Another possibility is to completely remove the old data by using a rectangular sliding window of length w. When using the sliding window, at each time instant t, the LS criterion is defined by

$$J(\boldsymbol{x}^{[t]}) = \sum_{i=t-w+1}^{t} \lambda^{t-i} |e^{[i]}|^2, \tag{4.54}$$

Algorithm 4.9: **Greedy recursive least squares (GRLS) - algorithm summary**

input: $\mathbf{R}^{[t-1]}$; $\boldsymbol{b}^{[t-1]}$; $\boldsymbol{\Psi}^{[t-1]}$; $\boldsymbol{\phi}^{[t-1]}$; $\boldsymbol{\alpha}^{[t]}$; $\beta^{[t]}$; m; n;
output: $\mathbf{R}^{[t]}$; $\boldsymbol{b}^{[t]}$; $\boldsymbol{\Psi}^{[t]}$; $\boldsymbol{\phi}^{[t]}$;

1 Perform the basic update from Algorithm 4.6

2 Reorder the support and restore the upper triangular form by searching between neighbors as in Algorithm 4.7

3 Search for a possible better candidate for the last position in the active set like in Algorithm 4.8

4 Estimate the current sparsity level $l^{[t]} = \tilde{l}^{[t]}$-sparse solution using either PLS or BIC.

5 Produce the $l^{[t]}$-sparse solution like in (4.43).

which translates into the same matrix expression (2.7). The main difference is that now the matrices have a fixed size, equal to the window length w. Thus we have $\mathbf{A}^{[t]} \in \mathbb{R}^{w \times n}$ and $\boldsymbol{b}^{[t]} \in \mathbb{R}^{w}$. We continue to use the same notations as for the exponential windowed data since essentially they represent the same concepts.

For a non-sparse solution $\boldsymbol{x}^{[t]}$, it is necessary to take $w \geq n$ in order to obtain a LS solution. However, for sparse filters, the required condition is $w \geq l_t^{[t]}$, which leaves open the possibility to take $w < n$. In what follows we consider a sufficiently fast time varying scenario where the past data soon begins to misrepresent the current environment and we show how the GRLS algorithm [P5, 35] is adapted to work with a sliding window in [P6].

The sliding window GRLS (SW-GRLS) algorithm is based on the same partial QR factorization with pivoting (3.12), on the first m columns,

$$\mathbf{A}^{[t]}\mathbf{P}^{[t]} = \mathbf{Q}^{[t]}\mathbf{R}^{[t]}, \tag{4.55}$$

like the OLS and GRLS algorithms but this time both the orthogonal matrix $\mathbf{Q}^{[t]} \in \mathbb{R}^{w \times w}$ and the upper triangular matrix $\mathbf{R}^{[t]} \in \mathbb{R}^{w \times n}$ need to be stored. Note that we store the whole matrix $\mathbf{R}^{[t]}$ not just the upper part. While we do not use the scalar products $\boldsymbol{\Psi}^{[t]}$ since we store the full matrix $\mathbf{A}^{[t]}$, we still use $\boldsymbol{\phi}^{[t]}$ to achieve lower computational complexity. At time t, the algorithm has two main steps, downdating and updating [64].

The downdating has the purpose of eliminating the oldest equation from the previous window. Starting from (4.55) at time $t-1$, we compute

$$\mathbf{A}^{[t-1]}\mathbf{P}^{[t-1]} = \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & \tilde{\mathbf{Q}}^{[t-1]} \end{bmatrix} \begin{bmatrix} \boldsymbol{\alpha}^{[t-w]^T} \\ \tilde{\mathbf{R}}^{[t-1]} \end{bmatrix}, \tag{4.56}$$

where $\boldsymbol{\alpha}^{[t-w]}$ is the input data from time $t-w$ which is to be eliminated. The matrix $\tilde{\mathbf{R}}^{[t-1]} \in \mathbb{R}^{w-1 \times n}$ is upper triangular in its first m columns, hence it corresponds to the data from time $t-1$ produced with a smaller window of size $w-1$.

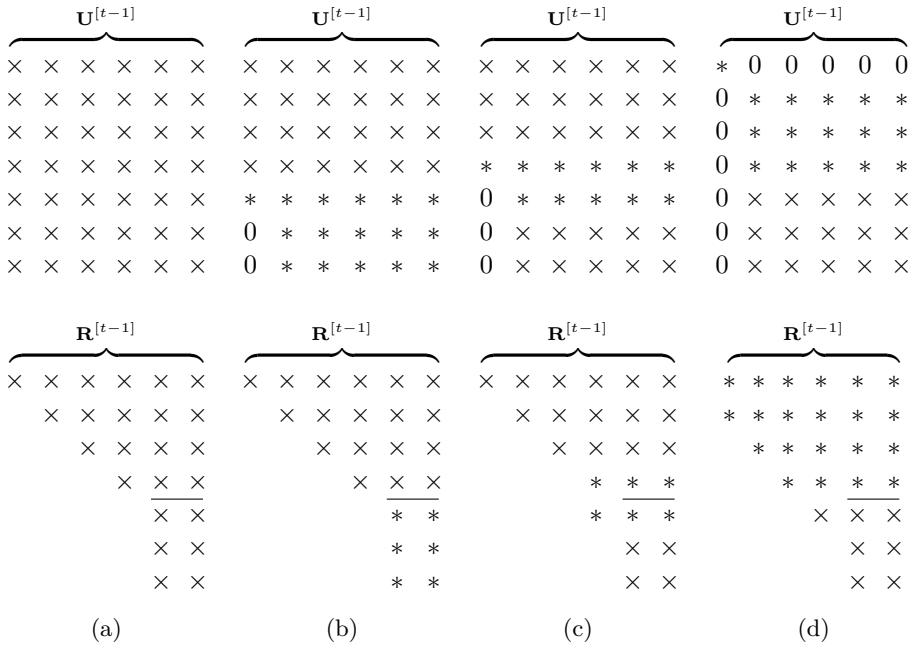


Figure 4.6: Matrices $\mathbf{U}^{[t-1]}$ and $\mathbf{R}^{[t-1]}$ illustrating the operations from Algorithm 4.10 for $m = 4$, $n = 6$ and $w = 7$; the non-zero elements are represented by $\times$ and those modified, to a non-zero value by the current operation, by $*$; the figures show the initial matrices (Figure 4.6a), the matrices after the Householder transform is applied (Figure 4.6b) and after the application of the first Givens rotation (Figure 4.6c); the matrices from (Figure 4.6d) exemplify the changes that occur in $\mathbf{U}^{[t-1]}$ and the upper Hessenberg structure of $\mathbf{R}^{[t-1]}$ after all the Givens transforms are applied.

This is like removing the first row from the permuted $\mathbf{A}^{[t-1]}$ to produce $\tilde{\mathbf{A}}^{[t-1]}$ and then computing its QR factorization $\tilde{\mathbf{A}}^{[t-1]}\mathbf{P}^{[t-1]} = \tilde{\mathbf{Q}}^{[t-1]}\tilde{\mathbf{R}}^{[t-1]}$. Note that the output vector $\boldsymbol{b}^{[t-1]}$ is also modified and its first element is similarly removed to produce the output $\tilde{\boldsymbol{b}}^{[t-1]}$ associated with the smaller window.

The downdating procedure consists of computing the elementary orthogonal transformations that bring $\mathbf{Q}^{[t-1]}$ to the form (4.56) by forcing zeros in its first row and column, in an order chosen to damage the least the triangular form of $\mathbf{R}^{[t-1]}$. We work with the inverse matrix $\mathbf{U}^{[t-1]} = \mathbf{Q}^{[t-1]^T} \in \mathbb{R}^{w \times w}$ since we apply all transformations from the left to produce $\mathbf{R}^{[t-1]} = \mathbf{U}^{[t-1]}\mathbf{A}^{[t-1]}$. The process is similar with that from [64]. We use a Householder reflector to zero the last $w - m + 1$ elements of the first column of $\mathbf{U}^{[t-1]}$ and we update both $\mathbf{R}^{[t-1]}$ and $\boldsymbol{b}^{[t-1]}$. The remaining $w-1$ non-zero elements from positions $2 : w$ are zeroed with several Givens rotations. After applying the last rotation the particular structure from (4.56) is produced because the matrix $\mathbf{U}^{[t]}$ is orthogonal. This operation order ensures that at the end $\mathbf{R}^{[t-1]}$ is upper Hessenberg in the first m columns

Algorithm 4.10: **Sliding window greedy recursive least squares (SW-GRLS) - downdate**

input: $\mathbf{U}^{[t-1]}$; $\mathbf{R}^{[t-1]}$; $\boldsymbol{b}^{[t-1]}$; w; m; n;
output: $\tilde{\mathbf{U}}^{[t-1]}$; $\tilde{\mathbf{R}}^{[t-1]}$; $\tilde{\boldsymbol{b}}^{[t-1]}$;

1 Compute the Householder reflector $\mathbf{H}$ that zeroes $\mathbf{U}^{[t-1]}_{w+1:m,1}$ and apply it on $\mathbf{U}^{[t-1]}$, $\mathbf{R}^{[t-1]}$ and $\boldsymbol{b}^{[t-1]}$ (Theorem A.3.1)

2 For $j = w : -1 : 1$

2.1 Compute the rotation $\mathbf{G}$ that zeroes $\mathbf{U}^{[t-1]}_{j+1,1}$ and modifies $\mathbf{U}^{[t-1]}_{j,1}$ and apply it on $\mathbf{U}^{[t-1]}$, $\mathbf{R}^{[t-1]}$ and $\boldsymbol{b}^{[t-1]}$ (Theorem A.3.2)

and, after eliminating its first row, it becomes upper triangular. An example of the operations used for downdating is given in Figure 4.6 and the whole sequence of operations is summarized in Algorithm 4.10.

The update procedure is the same as for the exponential window greedy recursive least squares algorithm. It starts by adding the current equation to the data factorization

$$\mathbf{A}^{[t]}\mathbf{P}^{[t-1]} = \begin{bmatrix} \begin{bmatrix} \tilde{\mathbf{A}}^{[t-1]}\mathbf{P}^{[t-1]} \end{bmatrix} \\ \boldsymbol{\alpha}^{[t]^T} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{Q}}^{[t-1]} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{R}}^{[t-1]} \\ \boldsymbol{\alpha}^{[t]^T} \end{bmatrix} \tag{4.57}$$

and the solution is revised like in Alg. 4.9 to include the new data. We also note that updating or downdating can be performed singly at time t, hence increasing or decreasing w. Also, the above algorithms can be easily modified for allowing m to increase or decrease and thus to allow the online sparsity estimation.

4.2.3 Group level sparsity estimation

Until now we considered the solution sparsity to be unstructured; the non-zero elements can take any position in the vector $\boldsymbol{x}^{[t]}$. We now assume that the solution has a simple structure produced by having the non-zero elements grouped together in a small number of clusters [40, 74]. In this case we say that the solution $\boldsymbol{x}^{[t]} \in \mathbb{R}^n$ is group sparse. Its number of non-zero elements is still much smaller than n but additionally the non-zero elements are present in few clusters with variable sizes. The difference from the standard sparse problem is that now it is more efficient to work with groups of non-zero elements instead of individual non-zeros. Working on a group level, one has to decide the number of such groups and the size of each group which introduces additional challenges.

We maintain the same assumption about the environment; it is time varying, hence the values and the positions of the non-zero elements of $\boldsymbol{x}^{[t]}$ may change in time. In [P7], we extend the GRLS algorithm to work for group sparse problems and we explore the use of ITC for estimating the group sparsity online. Other algorithms able to produce group sparse solutions are generally based on convex

relaxation techniques and use the $\ell_{1,\infty}$ norm. They are working either in batch mode [56, 72] or have an adaptive implementation [25].

For the group sparsity, the non-zero coefficients locations are supposed to be found in m_g blocks of sizes p_i, $i = 1 : m_g$. Similarly to the GRLS algorithm, at each time t, for computing the sparse solution we employ a greedy strategy to select m active columns $\boldsymbol{a}_i^{[t]}$ from $\mathbf{A}^{[t]}$ forming the m_g active groups. In the group GRLS (G-GRLS) algorithm, the column selection is performed on a group level however. Herein we assume that all groups[1] have the same size $\check{p}$.

We work on groups of columns $\mathcal{G}_i$ containing $\check{p}$ consecutive indices of the non-zero coefficients of the solution. The algorithm maintains the same partial QR factorization with pivoting of the matrix $\mathbf{A}^{[t]}$ like the GRLS algorithm but performs all permutations and searches on a group level. We always consider the groups to be formed by adjacent columns from the original matrix $\mathbf{A}^{[t]}$ before any permutations are applied.

The choice to include a new group $\mathcal{G}_i$ into the active set is based on the minimization of the LS criterion. The group becomes active if, by adding it to the previously selected groups, decreases with the largest amount the error sum from $J(\boldsymbol{x}^{[t]})$. This selection strategy also ensures a group order at time t based on the contribution of each group in decreasing $J(\boldsymbol{x}^{[t]})$. Thus, to work on a group level we only require changes to the column selection criteria from the GRLS algorithm, namely we transform the neighbor permutation and the last search to work on a group level. The basic update from Algorithm 4.6 remains unchanged.

Neighbor group permutations

After receiving new data, the group order is prone to change. Under the assumption that the change is slow, we use a group neighbor permutation strategy to restore the group order.

Given two neighbor groups $\mathcal{G}_i$ and $\mathcal{G}_j$ the decision to permute them is based on the decrease of the residual their permutation produces. If we select from $\mathbf{R}^{[t]}$ and $\boldsymbol{b}^{[t]}$ only the part associated with the two groups we have

$$\begin{bmatrix} \mathbf{R}^{[t]}_{\mathcal{G}_i,\mathcal{G}_i} & \mathbf{R}^{[t]}_{\mathcal{G}_i,\mathcal{G}_j} \\ \mathbf{0} & \mathbf{R}^{[t]}_{\mathcal{G}_j,\mathcal{G}_j} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}^{[t]}_{\mathcal{G}_i} \\ \boldsymbol{x}^{[t]}_{\mathcal{G}_j} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}^{[t]}_{\mathcal{G}_i} \\ \boldsymbol{b}^{[t]}_{\mathcal{G}_j} \end{bmatrix}, \tag{4.59}$$

where $\mathbf{R}^{[t]}_{\mathcal{G}_i,\mathcal{G}_i}$ and $\mathbf{R}^{[t]}_{\mathcal{G}_j,\mathcal{G}_j}$ are upper triangular matrices[2]. The residual decrease

[1] A mechanism for splitting the groups up to a minimum size $\hat{p} \leq \check{p}/2$ and for joining them is present in [P7] where it is assumed that that the real group sizes p_i are bounded by $\hat{p}$ and $\check{p}$,

$$\hat{p} \leq p_i \leq \check{p} \quad \text{with} \quad i = 1 : m_g. \tag{4.58}$$

[2] We remind that for any matrix $\mathbf{X}$, the notation $\mathbf{X}_{\mathcal{G}_i}$ selects the column subset of $\mathbf{X}$ associated with the set $\mathcal{G}_i$ and the actual positions of the columns in $\mathbf{X}$ can vary. For a vector $\boldsymbol{z}^T$, the partition $\boldsymbol{z}^T_{\mathcal{G}_i}$ selects in a similar fashion the elements (columns) associated with the set $\mathcal{G}_i$ and we use the same notation $\boldsymbol{z}_{\mathcal{G}_i}$ for $\boldsymbol{z}$ to select the same elements (corresponding to the rows). The notation $\mathbf{X}_{\mathcal{G}_j,\mathcal{G}_i}$ selects the partition from $\mathbf{X}_{\mathcal{G}_i}$ defined by the rows from the set $\mathcal{G}_j$, $\mathbf{X}_{\mathcal{G}_j,\mathcal{G}_i} = ((\mathbf{X}_{\mathcal{G}_i})^T_{\mathcal{G}_j})^T$.

due to selecting first group $\mathcal{G}_i$ is $\delta_{\mathcal{G}_i} = \|\boldsymbol{b}_{\mathcal{G}_i}^{[t]}\|_2^2$.

Any change in the group order affects the output vector $\boldsymbol{b}^{[t]}$ and we need to compute $\delta_{\mathcal{G}_j}$ after we perform the permutation. By permuting $\mathcal{G}_i$ and $\mathcal{G}_j$ the upper triangular form is destroyed and thus, we need to restore it with the use of Householder reflectors. We start from the left and compute each reflector such that it zeros $|\mathcal{G}_i|$ sub diagonal elements on each column while modifying the diagonal element. After we permute and restore the triangular form we have

$$\begin{bmatrix} \tilde{\mathbf{R}}_{\mathcal{G}_j,\mathcal{G}_j}^{[t]} & \tilde{\mathbf{R}}_{\mathcal{G}_j,\mathcal{G}_i}^{[t]} \\ \mathbf{0} & \tilde{\mathbf{R}}_{\mathcal{G}_i,\mathcal{G}_i}^{[t]} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{x}}_{\mathcal{G}_j}^{[t]} \\ \tilde{\boldsymbol{x}}_{\mathcal{G}_i}^{[t]} \end{bmatrix} = \begin{bmatrix} \tilde{\boldsymbol{b}}_{\mathcal{G}_j}^{[t]} \\ \tilde{\boldsymbol{b}}_{\mathcal{G}_i}^{[t]} \end{bmatrix}, \tag{4.60}$$

and $\delta_{\mathcal{G}_j}$ can be computed, $\delta_{\mathcal{G}_j} = \|\tilde{\boldsymbol{b}}_{\mathcal{G}_i}^{[t]}\|_2^2$. Note that, the permuted columns associated with $\mathcal{G}_i$ and the whole matrix $\mathbf{R}^{[t]}$ may not updated now since we only apply the triangularization to compute $\delta_{\mathcal{G}_j}$ and the permutation is temporary. The permutation is permanent only if

$$\frac{\delta_{\mathcal{G}_j}}{|\mathcal{G}_j|} > \frac{\delta_{\mathcal{G}_i}}{|\mathcal{G}_i|}, \tag{4.61}$$

and so the reflectors used to obtain it are applied to the whole $\mathbf{R}^{[t]}$ to the right of the permuted group.

Last group

To allow changes in the active group set, any possible inactive group can compete for the last active position. An exhaustive search for all possible inactive groups of size $\breve{p}$ is performed. Smaller groups are allowed only if the previously selected active groups limit the number of adjacent inactive columns.

We can write an alternate way of determining the criterion[1] $J(\boldsymbol{x}^{[t]})$,

$$\begin{aligned} J(\boldsymbol{x}_{\mathcal{G}_i}^{[t]}) &= (\boldsymbol{b}_o^{[t]} - \mathbf{A}_{\mathcal{G}_i}^{[t]} \mathbf{A}_{\mathcal{G}_i}^{[t]^+} \boldsymbol{b}_o^{[t]})^T (\boldsymbol{b}_o^{[t]} - \mathbf{A}_{\mathcal{G}_i}^{[t]} \mathbf{A}_{\mathcal{G}_i}^{[t]^+} \boldsymbol{b}_o^{[t]}) \\ &= \boldsymbol{b}_o^{[t]^T} \boldsymbol{b}_o^{[t]} - \boldsymbol{b}_o^{[t]^T} \mathbf{A}_{\mathcal{G}_i}^{[t]} \mathbf{A}_{\mathcal{G}_i}^{[t]^+} \boldsymbol{b}_o^{[t]}, \end{aligned} \tag{4.62}$$

with $\mathbf{A}_{\mathcal{G}_i}^{[t]^+} = (\mathbf{A}_{\mathcal{G}_i}^{[t]^T} \mathbf{A}_{\mathcal{G}_i}^{[t]^{-1}}) \mathbf{A}_{\mathcal{G}_i}^{[t]^T}$. Including the group $\mathcal{G}_i$ in the solution is therefore decreasing the criterion by

$$\delta_{\mathcal{G}_i} = \boldsymbol{b}_o^{[t]^T} \mathbf{A}_{\mathcal{G}_i}^{[t]} \mathbf{A}_{\mathcal{G}_i}^{[t]^+} \boldsymbol{b}_o^{[t]}. \tag{4.63}$$

This holds for any orthogonal transformation of $\mathbf{A}^{[t]}$ and $\boldsymbol{b}_o^{[t]}$, particularly for $\mathbf{R}^{[t]}$ and $\boldsymbol{b}^{[t]}$.

To replace the last group $\mathcal{G}_{m_g}$ with a new group $\mathcal{G}_i$ ($\mathcal{G}_i$ can overlap with $\mathcal{G}_{m_g}$) we require that

$$\frac{\delta_{\mathcal{G}_{m_g}}}{|\mathcal{G}_{m_g}|} < \max_{\mathcal{G}_i} \frac{\delta_{\mathcal{G}_i}}{|\mathcal{G}_i|}. \tag{4.64}$$

[1] The notation $\boldsymbol{b}_o^{[t]}$ is used to denote the original vector $\boldsymbol{b}(t)$ from (2.5) without any orthogonal transforms applied.

Computing the criterion (4.63) requires the inversion from $\mathbf{A}_{\mathcal{G}_i}^{[t]^+}$ which is very computationally demanding. To overcome this we propose to estimate $\delta_{\mathcal{G}_i}$ from the orthogonally transformed data, $\mathbf{R}^{[t]}$ and $\boldsymbol{b}^{[t]}$. We define

$$\delta_{\mathcal{G}_i} = (\mathbf{R}_{\mathcal{G}_{m_g},\mathcal{G}_i}^{[t]^T} \boldsymbol{b}_{\mathcal{G}_i}^{[t]} + \boldsymbol{\phi}_{\mathcal{G}_i}^{[t]^T})\boldsymbol{s}, \tag{4.65}$$

with $\boldsymbol{s}$ being the solution of

$$(\mathbf{R}_{\mathcal{G}_{m_g},\mathcal{G}_i}^{[t]^T} \mathbf{R}_{\mathcal{G}_{m_g},\mathcal{G}_i}^{[t]} + \boldsymbol{\Psi}_{\mathcal{G}_i,\mathcal{G}_i}^{[t]})\boldsymbol{s} = \mathbf{R}_{\mathcal{G}_{m_g},\mathcal{G}_i}^{[t]^T} \boldsymbol{b}_{\mathcal{G}_{m_g}}^{[t]} + \boldsymbol{\phi}_{\mathcal{G}_i}. \tag{4.66}$$

Note that we can use the scalar products $\boldsymbol{\Psi}^{[t]}$ and $\boldsymbol{\phi}^{[t]}$ instead of the whole matrices $\mathbf{R}^{[t]}$ and $\boldsymbol{b}^{[t]}$.

Let the matrix $\mathbf{R}_{\mathcal{G}_{m_g},\mathcal{I}}^{[t]^T} \mathbf{R}_{\mathcal{G}_{m_g},\mathcal{I}}^{[t]} + \boldsymbol{\Psi}^{[t]}$ and the associated $\mathbf{R}_{\mathcal{G}_{m_g},\mathcal{I}}^{[t]^T} \boldsymbol{b}_{\mathcal{G}_{m_g}} + \boldsymbol{\phi}^{[t]}$ be permuted back to the original column order from $\boldsymbol{A}^{[t]}$ and for simplicity lets assume that the set $\mathcal{I}$ is contiguous. Each $\check{p} \times \check{p}$ diagonal block defines a systems from (4.65). To find the solution we transform the system to be upper triangular and use a downdate update procedure [64] to morph between every possible diagonal block.

Group aware online sparsity estimation

The group order introduced by the permutation strategy can be exploited to allow the online estimation of the sparsity level with the use of the PLS and BIC criteria in addition to the column level criteria used so far. While the computation of the column aware ITC is similar to [P5], it is used to infer a group level sparsity estimation instead.

Thus, for the column based estimators BIC^c and PLS^c, the sparsity estimate is the minimizer $\tilde{l}$ of the criterion chosen, either (3.21) or (3.23). To adapt the sparsity estimate for our group problem, we compute the group number estimate l_g by finding the group containing the column $\tilde{l}$.

Since the ordering performed with the permutations is done at the group level without any column ordering in the groups, the column sparsity estimation strategy can potentially provide inaccurate estimates. To alleviate this we propose group aware forms for the ITC. In the BIC case the changes are straightforward, the criterion is computed for solutions $\boldsymbol{x}^{[t]}$ produced by the whole groups $\mathcal{G}_{1:k}$,

$$\text{BIC}_{\mathcal{G}_i}^g = n_{\text{ef}} \ln \|\boldsymbol{r}_{\mathcal{G}_i}\|_2^2 + (1+s) \ln n_{\text{ef}} = \text{BIC}_s. \tag{4.67}$$

where $s = \sum_{j=1}^{i} |\mathcal{G}_j|$ and $\boldsymbol{r}_{\mathcal{G}_i}$ is the residual produced by a solution containing groups $\mathcal{G}_{1:i}$. Note that we consider the penalty for the model complexity

$$\left(1 + \sum_{j=1}^{i} |\mathcal{G}_j|\right) \ln n_{\text{ef}} \tag{4.68}$$

to depend on the number of columns in the groups.

The PLS has a temporal dependence on the previous solutions and the direct generalization to the group case does not maintain the group structure. We propose several ways of extending the PLS criterion to work with groups of columns. The simplest uses the current group structure $\mathcal{G}_{1:m}$ from time t and computes the solution at group level,

$$\mathrm{PLS}^{g}_{\mathcal{G}_i} = \textstyle\sum_{j=0}^{t} \lambda^{t-j} e_s^{[k]^2} = \mathrm{PLS}_s, \tag{4.69}$$

where, similarly to (3.23), $e_s^{[j]}$ is the solution produced by the $s = \sum_{k=1}^{i} |\mathcal{G}_k|$-sparse solution from time j. If the groups are atomic, then we use the past group structure $\mathcal{G}_i^j$ from each time instant $j \leq t$. In this case the group level PLS criterion is given by

$$\mathrm{PLS}^{g_a}_{\mathcal{G}_i} = \sum_{j=1}^{t} \lambda^{t-j} e^{[k]^2}_{\sum_{i=1}^{j} |\mathcal{G}_i^j|}. \tag{4.70}$$

The splitting of groups and the different size of the groups potentially adversely influence the ability of PLS^{g_a} to produce a good sparsity estimate. To overcome these problems in [P7] we also propose several weighted versions of (4.70).

Chapter 5

Conclusions and summary

The research directed towards the development of the two sparsity-aware algorithm families was triggered by the desire of fast, robust and efficient methods for finding sparse solutions. We started with the development of the algorithms derived from the MP algorithm and then focused more on the methods that employ orthogonal transforms. The former are less computationally intensive without performance and robustness degradation.

5.1 Overview of the results

The complexity and the performance of the algorithms are better generally than those of competing methods. We have mainly used for comparison the SPARLS algorithm [12] and the best CD method from [10]. For the distributed scenario we compared the performance against the reweighted zero attracting sparse diffusion LMS (RZA) algorithms developed in [31].

5.1.1 Algorithm complexity

Under the assumption of slow support variation, the greedy methods may modify positions or number of non-zero coefficients less frequently than they estimate the coefficient values. The support essentially changes very slowly; faster changes corrupt the input data and the algorithms behave poorly until the past data associated with the past support is sufficiently forgotten for a chosen forgetting factor or sliding window. Because of this, the permutation operations from the GRLS family of algorithms can be performed less often, once every τ_p input samples. For the G-GRLS algorithm the splitting and the group joining presented in [P7] may similarly be performed every τ_s and τ_j input samples, respectively. For the algorithms derived from the MP, the cyclical update (associated with CAMP, I-CAMP and the I-CAMP-A) may be performed a number of times τ_c per time instant to provide a more accurate solution. Taking this into account, we present in Table 5.1 the summary of the complexity of the algorithms presented in Chapter 4. To

Table 5.1: Summary of the approximate complexity for the studied algorithms; the complexity of the online sparsity estimation is considered separately

Algorithm	Worst case complexity	ITC complexity	
		PLS	BIC
CAMP	$\frac{3}{2}n^2 + (\frac{1}{2} + \tau_c)m^2 + nm$	$\tau_c l^2 + m^2$	$\tau_c l^2 + m^2$
I-CAMP	$\frac{3}{2}n^2 + \frac{\tau_c}{3}m^3 + (\frac{1}{2} + 4\tau_c)m^2 + nm$	m^2	$\frac{3}{2}m^3 + 3m^2$
I-CAMP-A	$\frac{3}{2}(2n - m - p)(m + p)$ $+\frac{\tau_c}{3}m^3 + (\frac{1}{2} + 4\tau_c)m^2 + nm$	m^2	-
CD-AMP	$\frac{3}{2}n^2 + 2mn$	-	-
DCD-AMP	$\frac{3}{2}n^2 + 2mn + 10n$	$5m$	-
D-DCD-AMP[1]	$\frac{3}{2}n^2 + 4mn + 10n$	$5m$	-
GRLS	$(\frac{3}{2} + \frac{2}{\tau_p})(n - m)^2 + \mathcal{O}(mn)$	$2m^2$	$2m$
SW-GRLS	$(4 + \frac{4}{\tau_p})nl + (4 + \frac{4}{\tau_p})l^2$	$2m^2$	$2m$
G-GRLS	$(\frac{3}{2} + \frac{2}{\tau_p})p(n - m)^2 + \mathcal{O}(p^2 n)$	$2m^2$	$2m$
RLS	$\mathcal{O}(4n^2)$	-	-
RZA[1]	$\mathcal{O}(4n)$	-	-
SPARLS[2]	$\mathcal{O}(\tau l n)$	-	-
TNWL[3]	$\mathcal{O}(\frac{3}{2}n^2) + \mathcal{O}(6n^2)$	-	-

provide the means of comparison, we include the SPARLS method from [12] and the TNWL algorithm from [10] together with the RLS algorithm. For comparison in the distributed scenario we use the RZA algorithms [31]. We only count the most significant operations.

We note that in many applications, the matrix $\mathbf{A}^{[t]}$ has a special structure because of time shifted input data as in (2.13). In such case, the complexity of the algorithms[4] is much lower since they can be implemented to take advantage of the special input structure and the costly update of the scalar products that requires $\frac{3}{2}n^2$ computations is implemented using only $3n$ total operations. This is possible because the scalar products $\mathbf{\Psi}^{[t]}$ from (4.3) are computed by copying the upper left block and then only the first row is updated.

[1] We report the number of computations from a single node at time t.

[2] The parameter τ represents the number of times the SPARLS algorithm calls a low complexity expectation maximization routine.

[3] The complexity corresponds to the best performing but also the most complex of the methods from [10] and it includes that of the RLS algorithm which runs in parallel.

[4] This does not apply to the GRLS based algorithms due to the special QR decomposition they maintain.

Table 5.2: Summary of the algorithms used for the performance assessment and their configuration

Algorithm	Description
CAMP CAMP-P CAMP-B	The CAMP algorithm family from [P3]. The CAMP algorithm is running with prior knowledge about the sparsity level $m = l_t$. The two other methods, CAMP-P and CAMP-B use the PLS and BIC criteria, respectively, to estimate online the sparsity level. For both we use a variable[1] parameter $m^{[t]}$ with $\Delta = 5$.
I-CAMP I-CAMP-P I-CAMP-B	The I-CAMP algorithm family from [P3]. The I-CAMP algorithm is running with prior knowledge about the sparsity level $m = l_t$. The I-CAMP-P and I-CAMP-B methods use the PLS and BIC criteria, respectively, to estimate online the sparsity level. For both we use a variable[1] parameter $m^{[t]}$ with $\Delta = 5$.
GRLS GRLS-P GRLS-B	The GRLS algorithm family from [P5]. The GRLS algorithm is running with prior knowledge about the sparsity level $m = l_t$. The two other methods, GRLS-P and GRLS-P use the PLS and BIC criteria, respectively, to estimate online the sparsity level. For both we use a variable[1] parameter $m^{[t]}$ with $\Delta = 5$.
SW-GRLS	The sliding window algorithm from [P6] running with prior knowledge about the sparsity level $m = l_t$.
G-GRLS G-GRLS-P G-GRLS-B	The group algorithms from [P7] running with prior knowledge about the sparsity level $m = l_t$. The two other methods, G-GRLS-P and G-GRLS-P use the group level PLS and BIC criteria from (4.69) and (4.67), respectively, to estimate online the sparsity level. For both we use a variable number[2] of groups $m_g^{[t]}$ with $\Delta_g = 2$ and a minimum group size $\hat{p} = 2$.
CD-AMP	The CD-AMP algorithm from [P1] running with prior knowledge about the sparsity level $m = l_t$.
DCD-AMP	The DCD-AMP algorithms from [P1] using the PLS criterion to estimate online the sparsity level. A variable[1] parameter $m^{[t]}$ is used with $\Delta = 5$.

[1] For a constant value of m the performance is similar thus we only show the results of the more robust version of the algorithms that also estimates $m^{[t]}$ online.

[2] Like for the column based algorithms, for a constant value of m_g the performance is similar thus we only show the results of the more robust version of the algorithms that also estimates $m_g^{[t]}$ online.

Table 5.2: Summary of the algorithms used for the performance assessment and their configuration

Algorithm	Description
D-DCD-AMP	The distributed algorithm from [P2]. It uses as network topology $\mathcal{T}$: a ring $\mathcal{T}_r$ or a fully connected setup $\mathcal{T}_f$. The network size is $\lvert\mathcal{N}\rvert = 10$. Similarly to the centralized case we use a variable[1] $m^{[t]}$ with $\Delta = 5$
RLS RLS-SP	The standard and the sparsity informed RLS algorithms. The RLS-SP algorithm knows the number and positions of the non-zero coefficients and thus estimates only the true coefficients. It provides the best performance that can be achieved by minimizing the LS criterion.
SPARLS	The algorithm from [12] using optimally selected parameters.
TNWL	The algorithm from [10] using optimally selected parameters and a tuned λ for the RLS algorithm that runs in parallel.
RZA-ATC RZA-CTA	The LMS based, diffusion algorithms from [31]. Two strategies are used producing an RZA algorithm that adapts the coefficients to include the new local data and then combines them with the neighbor data (RZA-ATC) and an RZA algorithm that combines the coefficients with the neighbor data and then adapts them to include the new local data (RZA-CTA).

In the general case, the algorithms from the CD-AMP family have good complexity when compared to the GRLS methods. In the GRLS algorithm, the term depending on nm generally involves more operations than the whole DCD-AMP for time shifted input data. If the input data structure can not be exploited, the update of the scalar products raises the complexity of DCD-AMP such that, for low values of m and large n, the complexity of GRLS is generally more than $\frac{5}{3}$ times higher. The TNWL algorithm, which uses an efficient CD coupled with an inner RLS loop, is essentially[1] more than three times more complex than the DCD-AMP and has about twice the complexity of GRLS. Since the computation of the RLS solution can be implemented to consider the time shifted input data, in such case, the complexity of the TNWL is slightly higher than that of the DCD-AMP. When compared to the SPARLS algorithm[2], only DCD-AMP has a similar complexity. For

[1] Depends on the implementation of the RLS algorithm. A stable implementation using the QR decomposition has about $\mathcal{O}(6n^2)$ operations.

[2] Our algorithms have much better performance.

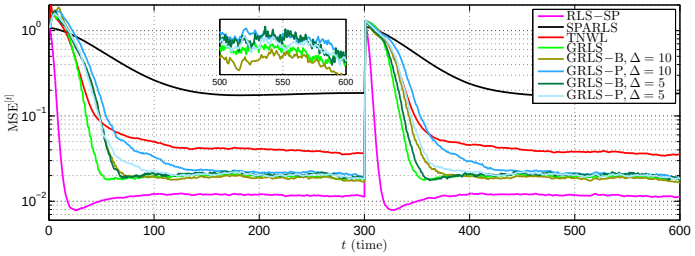


Figure 5.1: The evolution in time of $\text{MSE}^{[t]}$ for a variation speed governed by $f = 0.001$. At time $t = 300$, three of the $l_t = 5$ coefficients change position. The forgetting factor used is $\lambda = 0.92$.

the distributed D-DCD-AMP algorithm, the complexity is higher than that of the RZA algorithms which are based on the NLMS algorithm. If the input is time shifted, then the complexity of the two approaches becomes comparable.

5.1.2 Performance assessment

The performance of our algorithms was validated for an FIR channel identification problem (2.13). We used a sparse filter $\boldsymbol{h}^{[t]}$ with a length n to generate the output data give a random input generated according to a normal distribution with zero mean and unit variance. Each non-zero coefficient has a sinusoidal variation given by

$$h_i^{[t]} = \upsilon_i \cos(2\pi f t + \zeta_i). \tag{5.1}$$

The l_t non-zero coefficient positions i are randomly chosen, each with the amplitude υ_i and the phase ζ_i distributed uniformly in $[0.05, 1]$ and $[0, 2\pi]$. We impose a group structure over a grid of size p only for the test involving G-GRLS. The coefficient vector is normed such that its average squared norm over all t is 1. The outputs, corresponding to the filtered inputs, are corrupted by a zero mean, additive noise, normally distributed with $\sigma^2 = 0.01$.

The performance of the algorithms is measured in terms of the mean square error of the coefficients

$$\text{MSE}^{[t]} = \mathbb{E}\{\|\boldsymbol{h}^{[t]} - \boldsymbol{x}^{[t]}\|_2^2\}, \tag{5.2}$$

where $\boldsymbol{x}^{[t]}$ is the estimate of the true coefficient vector $\mathbf{h}^{[t]}$. The value of $\text{MSE}^{[t]}$ is evaluated by averaging over multiple runs at each time t. We also report the average value of the $\text{MSE}^{[t]}$ over the last 100 samples of the steady state.

The algorithms used for the performance assessment are summarized in Table 5.2. To keep the presentation clear we use the GRLS and the CD-AMP algorithms as a baseline and we compare the other developed algorithms with them. These are also compared with the SPARLS and TNWL algorithms to give insight on how our algorithms perform against other recently developed methods. We use different coefficient variation speeds, namely $f = 0.002$ and $f = 0.001$, to generate fast

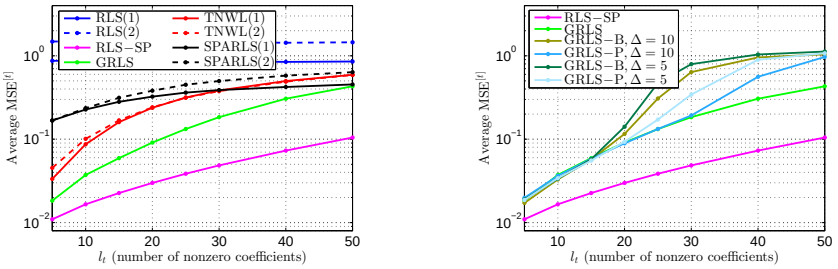


Figure 5.2: The average $\text{MSE}^{[t]}$ as a function of the true number of coefficients l_t and a constant filter of length $n = 200$. The variation speed is governed by $f = 0.001$. The forgetting factor used is $\lambda = 0.92$ for all algorithms except RLS(1) which uses $\lambda = 0.9825$.

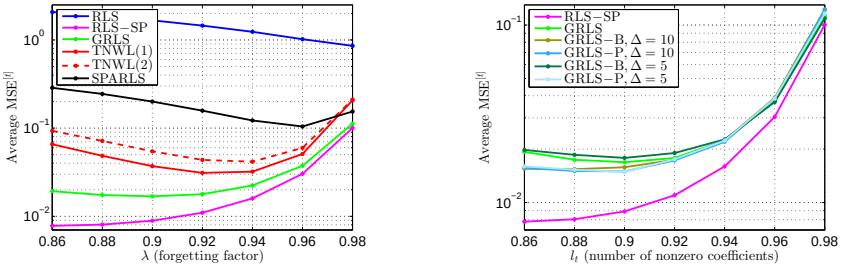


Figure 5.3: The average $\text{MSE}^{[t]}$ as a function of the forgetting factor λ for a number of true coefficients $l_t = 5$ and a constant filter of length $n = 200$. The variation speed is governed by $f = 0.001$.

coefficient changes and $f = 0.0002$ for a slower evolution. We also compare the D-DCD-AMP algorithm with the distributed LMS based algorithms, RZA-ATC and RZA-CTA, introduced by [31].

We begin by presenting the time evolution, in a time varying setup, for the GRLS algorithm family in Figure 5.1 to give an idea about the convergence speed and stationary error. The GRLS algorithm, having prior knowledge of the true sparsity level, has the fastest convergence speed but requires information that is generally unavailable. The fully adaptive algorithms that use the BIC and PLS criteria to estimate online the number of coefficients converge more slowly but are still comparable with the GRLS in terms of convergence speed. The stationary error is essentially the same for all of them proving the ability of the ITC to select to correct sparsity level. When compared to the TNWL and the SPARLS algorithms, our algorithms have a better performance. In [P5], extensive simulations are presented to validate the performance for different channel variations. It was concluded that, for very slow variation speeds or for constant channels, the TNWL algorithm produces good results that approach those produced by the GRLS algorithms. This can also be seen later in Figure 5.6 which contains tests for a slow variations speed.

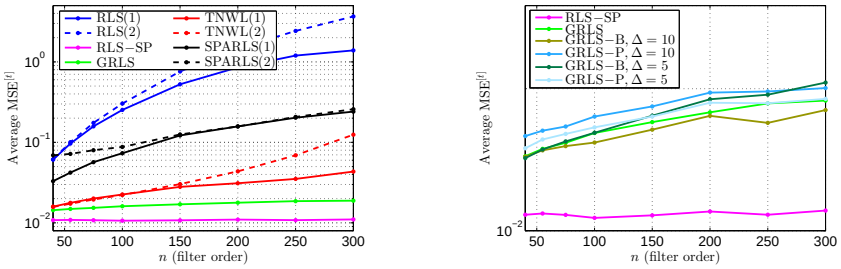


Figure 5.4: The average $\text{MSE}^{[t]}$ as a function of the filter order n for a number of true coefficients $l_t = 5$. The forgetting factor used is $\lambda = 0.92$ for all algorithms besides RLS(1) which uses λ optimized for each test case. The variation speed is governed by $f = 0.001$.

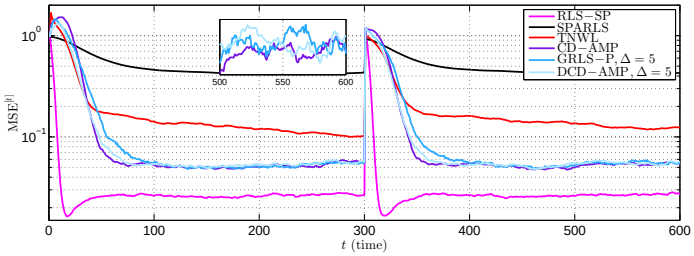


Figure 5.5: The evolution in time of $\text{MSE}^{[t]}$ for a variation speed governed by $f = 0.002$. At time $t = 300$, three of the $l_t = 5$ coefficients change position. The forgetting factor used is $\lambda = 0.90$.

To assess the robustness of the proposed methods we include in Figure 5.2 the evolution of the average $\text{MSE}^{[t]}$ as a function of the true sparsity level l_t. Similarly, Figures 5.3 and 5.4 contain the average $\text{MSE}^{[t]}$ as a function of the forgetting factor λ and of the filter length n, respectively. The SPARLS and TNWL algorithms marked in with a dotted line use the parameters, namely γ for SPARLS and the forgetting factor used by the internal RLS in the case of TNWL, optimally chosen for $l_t = 5$ and $n = 200$ for all other test cases. The versions marked with a continuous line use optimized parameters for each test case. In the case of the RLS algorithm the dotted line uses $\lambda = 0.92$ while the algorithm marked by a continuous line uses λ that produces the best performance.

The GRLS algorithms perform robustly and maintain good performance. The only case when there is a large performance degradation is when the true sparseness of the solution decreases. In such case the ITC fail to produce good estimates. It should be noted that the PLS criterion is more robust.

Figures 5.5 and 5.6 contain the time evolution of the $\text{MSE}^{[t]}$ for the CD-AMP algorithms and show how they compare to the more computationally intensive GRLS algorithms and to the other competing methods, SPARLS and TNWL. We

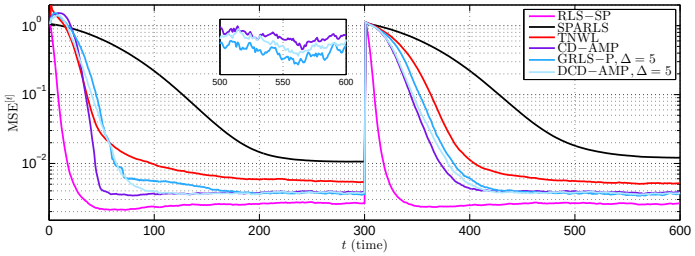
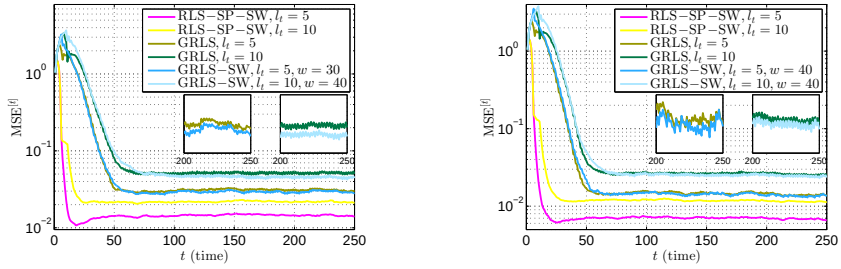


Figure 5.6: The evolution in time of $\text{MSE}^{[t]}$ for a variation speed governed by $f = 0.0002$. At time $t = 300$, three of the $l_t = 5$ coefficients change position. The forgetting factor used is $\lambda = 0.96$.



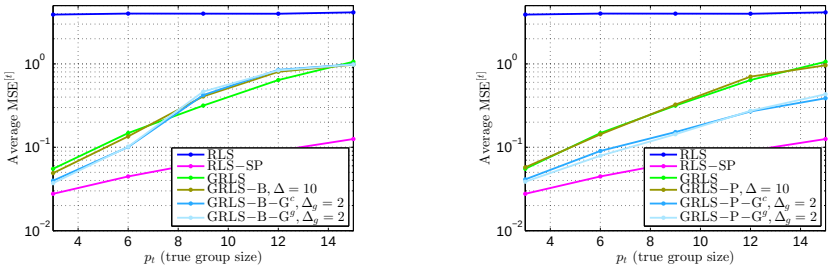
(a) (b)

Figure 5.7: The evolution in time of $\text{MSE}^{[t]}$ for a variation speed governed by $f = 0.001$ in Figure 5.7a and $f = 0.002$ in Figure 5.7b. The forgetting factors are $\lambda = 0.90$ and $\lambda = 0.92$, respectively. Two test cases, i.e. two different data sets with $l_t = 5$ and $l_t = 10$, are presented in each figure. The filter length is $n = 200$.

only use the PLS criterion since it is more robust but a similar comparison is possible also for the BIC criterion. The simulations are performed for both a fast and slow variation speed. They also act to complement the ones performed solely for the GRLS since the setup is different. It can be seen that in both cases the DCD-AMP algorithm performance is very close to that of the GRLS-P and it converges marginally faster.

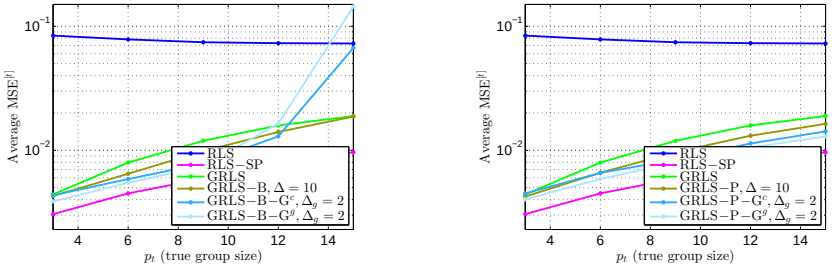
We compare the sliding window and exponential window GRLS algorithms in Figure 5.7. It can be seen that the sliding window approach is able to produce comparable results with those produced using the exponential windowed data. For small window length $w < 0.25n$, like those employed for fast varying channels, the algorithms is less computationally expensive than its forgetting window counterpart.

In case of a setup where the coefficients are grouped together, the performance of the algorithms can be improved by using a group aware algorithm. We present in Figure 5.8 and Figure 5.9 the average $\text{MSE}^{[t]}$ of the group aware GRLS algorithm for the different ITC introduced by [P7]. The group aware G-GRLS algorithms gen-



(a) (b)

Figure 5.8: The average $\text{MSE}^{[t]}$ as a function of the true group size p_t for a number of groups $m_g = 2$. All groups have the same size. The filter length is $n = 180$. Figure 5.9a depicts the evolution of the algorithms that use the BIC criterion while Figure 5.9b contains the algorithms that use the PLS criterion. The forgetting factor used is $\lambda = 0.9$ for all algorithms and the variation speed is fast, governed by $f = 0.002$.



(a) (b)

Figure 5.9: The average $\text{MSE}^{[t]}$ as a function of the true group size p_t for a number of groups $m_g = 2$. All groups have the same size. The filter length is $n = 180$. Figure 5.9a depicts the evolution of the algorithms that use the BIC criterion while Figure 5.9b contains the algorithms that use the PLS criterion. The forgetting factor used is $\lambda = 0.96$ for all algorithms and the variation speed is slow, governed by $f = 0.0002$.

erally outperform the GRLS algorithms for both the BIC and the PLS criteria. The BIC criteria are not very robust and for large number of groups and fast variation speeds they underestimate the true sparsity level. Both group and column based PLS criteria are more robust and perform better than the corresponding BIC counterparts and the original GRLS algorithm.

The performance of the CAMP algorithms can be judged from Figure 5.10. It contains the CAMP and I-CAMP algorithms that use both BIC and PLS criteria and the approximate version of the I-CAMP algorithm with the online sparsity estimation given by the PLS criterion. For comparison we also show the evolution of the GRLS algorithm that knows the true number of non-zero coefficients. It

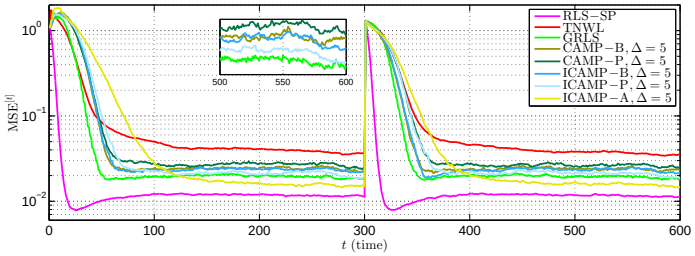


Figure 5.10: The evolution in time of $\text{MSE}^{[t]}$ for a variation speed governed by $f = 0.001$. The forgetting factor is $\lambda = 0.92$, the filter length is $n = 200$ and the true number of coefficients is $l_t = 5$.

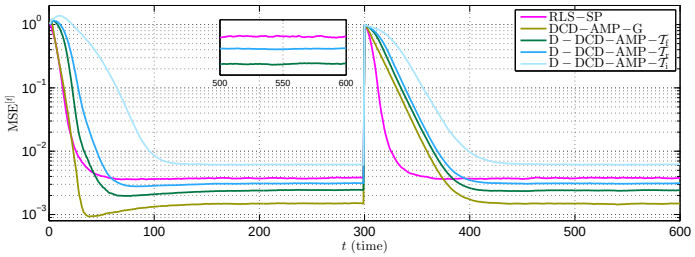


Figure 5.11: The evolution in time of $\text{MSE}^{[t]}$ for a variation speed governed by $f = 0.0002$. The forgetting factor is $\lambda = 0.96$, the filter length is $n = 200$ and the true number of coefficients is $l_t = 10$.

can be seen that the $\text{MSE}^{[t]}$ of the CAMP algorithms approaches that of GRLS showing that the cyclic update scheme is able to provide a good approximation of the true coefficients. The performance of the approximate I-CAMP algorithm is improved because it does not allow fast changes in the support and thus bad positions enter the solution rarely. This affects however the convergence speed which almost doubles.

The performance in time of the distributed algorithm introduced in [P2] is presented in Figure 5.11 and Figure 5.12, respectively. First we compare the D-DCD-AMP algorithm running on two network topologies, a ring $\mathcal{T}_r$ and a fully connected network $\mathcal{T}_f$, with the sparsity informed RLS (RLS-SP) algorithm running on one node and the non distributed DCD-AMP algorithm that has all the data available locally (DCD-AMP-G). We also include a unconnected topology $\mathcal{T}_i$, where each node is independent. This represents the DCD-AMP running on each node without any communication and is included to provide insight into the ability of the distributed algorithm to improve the solution by communicating over the network. Figure 5.12 includes the recent distributed algorithms proposed by [31] and shows how D-DCD-AMP behaves when compared to existing state of the art methods.

Figure 5.13 contains the average $\text{MSE}^{[t]}$ as a function of the configuration

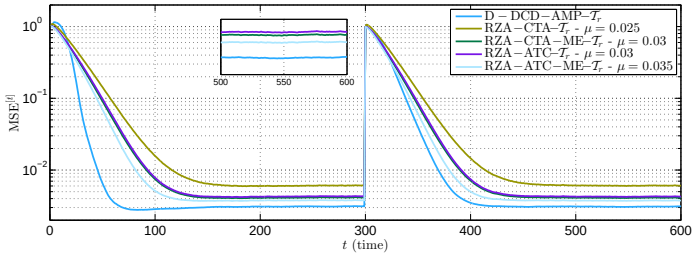
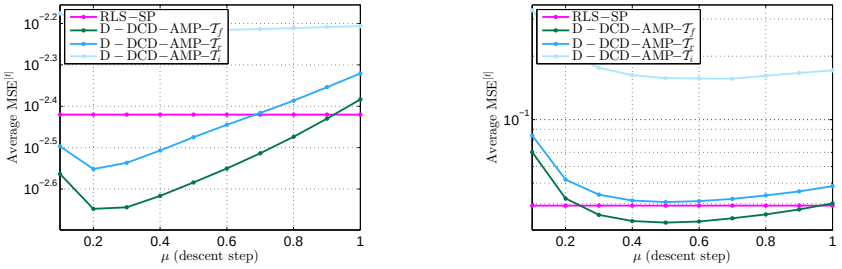


Figure 5.12: The evolution in time of MSE$^{[t]}$ for a variation speed governed by $f = 0.0002$. The forgetting factor is $\lambda = 0.96$, the filter length is $n = 200$ and the true number of coefficients is $l_t = 10$.



(a) (b)

Figure 5.13: The average MSE$^{[t]}$ as a function of the descent step μ. Figure 5.13a depicts the average MSE$^{[t]}$ of the algorithms for $f = 0.002$ and $\lambda = 0.90$ while Figure 5.13b contains the average MSE$^{[t]}$ for $f = 0.0002$ and $\lambda = 0.96$. The filter length used is $n = 200$ and all simulations have the true number of coefficients $l_t = 10$.

parameter μ which acts as a tradeoff between convergence speed and stationary error. It can be seen that for fast channel variation, the best results are produced for a larger μ while for a slow variation a smaller μ allows for a better stationary error.

5.2 Conclusions

We have proposed several greedy adaptive algorithms for finding sparse solutions. Their performance has been extensively tested for a simulated FIR channel identification task. The algorithms belong to two different families of methods. The algorithms from the first one are derived from the MP and use a CD approach to estimate the solution. The latter are based on the batch OLS and compute the solution by maintaining a partial QR triangularization with pivoting. They generate the solution by solving a system of equations defined by the upper triangular part of the data matrix.

All algorithms require little configuration, generally one or two, easy to choose, parameters. No precise a priori information is necessary, conferring robust behavior for unknown environments. Generally the GRLS family has higher computational complexity if compared to the CD-AMP algorithms. However, since it generates the proper LS solution for the given sparsity, it produces slightly better results. The CD-AMP algorithms achieve the same performance and have considerably lower complexity. If compared to competing methods, our algorithms have better performance and generally lower complexity.

The algorithms use ITC to estimate online the sparsity level. Using extensive simulations, it was shown that both criteria proposed, the PLS and BIC, are able to generate reliable estimates. The PLS behaves more robustly than the BIC although it may sometimes produce slightly worse performance.

5.3 Author's contribution

The research which led to the publications presented in this thesis was performed while the author was a researcher at the Department of Signal Processing of Tampere University of Technology. The work was supervised by Prof. Bogdan Dumitrescu[1] and Prof. Ioan Tăbuş. All the publications are a result of the collaboration with the supervisors. The author of the thesis is the main contributor to publications [P1, P2, P3, P4, P6, P7] and also contributed to [P5].

A brief description of the contributions to each article is found below:

[P1] The author of the thesis proposed the main idea of the algorithms, the coordinate descent strategy for the recursive update of the coefficients. He also was responsible for the algorithm implementation, the various experimental simulations and the writing of the publication. The article was finalized with help from B. Dumitrescu.

[P2] The main ideas of the article were proposed by the first author, which was also responsible for the implementation and the writing of the publication. The collaboration with the second author consisted of several constructive discussions which lead towards the final version of the algorithm and article.

[P3] The two algorithms proposed were mainly developed by the author of this thesis but include some suggestions from B. Dumitrescu. The implementation and the writing of the publication were done mainly by the first author. Some improvements to the final form of the publication resulted from the collaboration between the two authors.

[P4] The article introduces a series of approximations, all originally proposed by the author of the thesis. The implementation and writing of the publication were performed independently apart from several constructive discussions with B. Dumitrescu.

[1] FiDiPro (Finland Distinguished Professor) fellow at the Department of Signal Processing

[P5] The author of the thesis contributed to the development of the use of ITC for online sparsity estimation. He was responsible with the implementation of the algorithms and the generation of the simulated tests from [P5, Section VI] as well as the writing of that section. He was also involved in efforts to improve the final version of the article.

[P6] The development of the sliding window algorithm was mainly a joint collaboration between the first two authors. The first author was the main contributor for the development and implementation of the proposed method but received several suggestions from B. Dumitrescu and I. Tăbuş. The writing of the publication was performed by B. Dumitrescu; towards the final stages it consisted of collaboration between all the authors.

[P7] The article extends the GRLS algorithm to produce improvements for group sparse solutions. The first author has the main contributions in the development and implementation of the methods. He also was responsible with writing the publication. Several fruitful discussions with the second author produced the final form of the article and the algorithm.

Appendix A

Mathematical appendix

A.1 Matrix inversion lemma

Lemma A.1.1 (matrix inversion[1]). *For any invertible matrices* $\mathbf{Y} \in \mathbb{R}^{n\times n}$, $\mathbf{Z} \in \mathbb{R}^{m\times m}$ *and* $\mathbf{U} \in \mathbb{R}^{n\times m}$, $\mathbf{V} \in \mathbb{R}^{m\times n}$ *the following equation holds,*

$$(\mathbf{Y} + \mathbf{UZV})^{-1} = \mathbf{Y}^{-1} - \mathbf{Y}^{-1}\mathbf{U}(\mathbf{Z}^{-1} + \mathbf{VY}^{-1}\mathbf{U})^{-1}\mathbf{VY}^{-1}. \tag{A.1}$$

In particular if $\mathbf{B} = \mathbf{Y}^{-1}$, $\mathbf{C} = \mathbf{U} = \mathbf{V}^T$ *and* $\mathbf{D} = \mathbf{Z}^{-1}$ *we have that*

$$\mathbf{A}^{-1} = \mathbf{B} - \mathbf{BC}(\mathbf{D} + \mathbf{C}^T\mathbf{BC})^{-1}\mathbf{C}^T\mathbf{B}, \tag{A.2}$$

where $\mathbf{A} = \mathbf{B}^{-1} + \mathbf{CD}^{-1}\mathbf{C}^T$.

A.2 Sparsity recovery analysis

Theorem A.2.1 (spark lower bound). *For any matrix* $\mathbf{A} \in \mathbb{R}^{n\times m}$ *the following relation holds*

$$\operatorname{spark}(\mathbf{A}) \geq 1 + \frac{1}{\mu(\mathbf{A})}. \tag{A.3}$$

Theorem A.2.2 (uniqueness of a sparse solution). *If the solution* $\boldsymbol{x}$ *of a system of linear equations* $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ *obeys*

$$\|\boldsymbol{x}\|_0 < \frac{1}{2}\left(1 + \frac{1}{\mu(\mathbf{A})}\right), \tag{A.4}$$

then the solution is the sparsest possible.

Theorem A.2.3 ($\operatorname{spark}_\eta$ lower bound). *If the matrix* $\mathbf{A}$ *has normalized columns and a mutual coherence* $\mu(\mathbf{A})$ *then, for any given* η *we have*

$$\operatorname{spark}_\eta(\mathbf{A}) \geq 1 + \frac{1-\eta^2}{\mu(\mathbf{A})}. \tag{A.5}$$

[1] Also known as the Woodbury matrix identity [97].

Theorem A.2.4 (stability of a sparse solution)**.** *If we have a solution $\boldsymbol{x}$ verifying the conditions posed by Theorem A.2.2 then any other solution $\boldsymbol{x}^\epsilon$ verifying $\|\mathbf{A}\boldsymbol{x} - \boldsymbol{b}\|_2 \leq \epsilon$ must obey*

$$\|\boldsymbol{x} - \boldsymbol{x}^\epsilon\|_2^2 \leq \frac{4\epsilon^2}{1 - \mu(\mathbf{A})(2\|\boldsymbol{x}\|_0 - 1)}. \tag{A.6}$$

A.3 Orthogonal transforms

Definition A.3.1. *Let $\boldsymbol{u} \in \mathbb{R}^n$, then the transformation $\mathbf{H} \in \mathbb{R}^{n \times n}$,*

$$\mathbf{H} \triangleq \mathbf{I}_n - \frac{\boldsymbol{u}\boldsymbol{u}^T}{\beta}, \tag{A.7}$$

with $\beta = \frac{1}{2}\|\boldsymbol{u}\|_2^2$, is named a Householder reflector[1]*.*

Property A.3.1. *The Householder reflector is symmetric and orthogonal,*

$$\mathbf{H} = \mathbf{H}^T = \mathbf{H}^{-1}. \tag{A.8}$$

Theorem A.3.1 (introducing zeroes in a vector using a Householder transform[2])**.** *For any vector $\boldsymbol{x} \in \mathbb{R}^n$ and a given integer $1 \leq k \leq n-1$ for which*

$$\sigma \triangleq \operatorname{sgn}(x_k)\sqrt{\sum_{i=k}^{n} x_i^2} \neq 0, \tag{A.9}$$

there exists a vector $\boldsymbol{u}$ with

$$\boldsymbol{u}_i = \begin{cases} 0, & for \quad i < k \\ x_k + \sigma, & for \quad i = k \\ x_i, & for \quad i > k \end{cases}, \tag{A.10}$$

defining a Householder transform $\mathbf{H}$ that zeroes the vector $\boldsymbol{x}$ on positions greater than k. Here we have $\beta = \frac{1}{2}\|\boldsymbol{u}\|_2^2 = \boldsymbol{u}_k\sigma$. Moreover, the transformed vector is

$$\mathbf{H}\boldsymbol{x} = \begin{cases} x_i, & for \quad i < k \\ -\sigma, & for \quad i = k \\ 0, & for \quad i > k \end{cases}. \tag{A.11}$$

Property A.3.2. *Let $\mathbf{H} \in \mathbb{R}^{n \times n}$ be a Householder transform defined with the use of $\boldsymbol{u} \in \mathbb{R}^n$ and $\boldsymbol{x} \in \mathbb{R}^n$. The applied Householder transform $\mathbf{H}\boldsymbol{x}$ is a reflection of $\boldsymbol{x}$ over the hyperspace perpendicular to the vector $\boldsymbol{u}$,*

$$\mathbf{H}\boldsymbol{x} = \boldsymbol{x} - \frac{\boldsymbol{u}^T\boldsymbol{x}}{\beta}\boldsymbol{u}. \tag{A.12}$$

[1] Introduced by Householder in [55].

[2] The use of $\operatorname{sgn}(x_k)$ in the definition of σ is only required for the numerical stability of the transformation.

Note that the application of a Householder does not involve a multiplication by $\mathbf{H}$. *It requires the computation of the scalar*

$$\nu = \frac{\boldsymbol{u}^T \boldsymbol{x}}{\beta} \tag{A.13}$$

and the subtraction of the scaled vector $\nu\boldsymbol{u}$ *from* $\boldsymbol{x}$. *Also, a reflector constructed according to Theorem A.3.1 has an associated vector* $\boldsymbol{u}$ *having elements* $\boldsymbol{u}_{1:k-1}$ *equal to zero and thus does not modify* $\boldsymbol{x}_{1:k-1}$.

Definition A.3.2. *Let* k *and* i *be two integers with* $1 \leq i < k \leq n$. *The matrix* $\mathbf{G}$ *defined as*

$$\mathbf{G} = \begin{bmatrix} \mathbf{I}_{i-1} & & & & \\ & c & & s & \\ & & \mathbf{I}_{k-i-1} & & \\ & -s & & c & \\ & & & & \mathbf{I}_{n-k} \end{bmatrix}, \tag{A.14}$$

with s *and* c *obeying* $s^2 + c^2 = 1$, *is named a Givens rotation* [1,2].

Property A.3.3. *The Givens rotation matrix* $\mathbf{G}$ *is orthogonal,*

$$\mathbf{G}^T = \mathbf{G}^{-1}. \tag{A.15}$$

Property A.3.4. *Let* $\mathbf{G}$ *be a Givens rotation matrix defined for two integers* $i < k$ *and* $\boldsymbol{x} \in \mathbb{R}^n$ *be any vector. Then, by applying the transformation, we modify only the elements* i *and* k; *that is*

$$\mathbf{G}\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_{1:i-1} \\ c\boldsymbol{x}_i + s\boldsymbol{x}_k \\ \boldsymbol{x}_{i+1:k-1} \\ -s\boldsymbol{x}_i + c\boldsymbol{x}_k \\ \boldsymbol{x}_{k+1:n} \end{bmatrix}. \tag{A.16}$$

Theorem A.3.2 (introducing zeroes in a vector using Givens transforms)**.** *Let* k *and* i *be two integers with* $1 \leq i < k \leq n$ *and a vector* $\boldsymbol{x} \in \mathbb{R}^n$. *If*

$$r \stackrel{\Delta}{=} x_i^2 + x_k^2 \neq 0, \tag{A.17}$$

then we can construct a matrix $\mathbf{G}$ *with* $c = \frac{x_i}{r}$ *and* $s = \frac{x_k}{r}$ *that zeroes the vector* $\boldsymbol{x}$ *on position* k,

$$\mathbf{G}\boldsymbol{x} = \begin{bmatrix} \boldsymbol{x}_{1:i-1} \\ r \\ \boldsymbol{x}_{i+1:k-1} \\ 0 \\ \boldsymbol{x}_{k+1:n} \end{bmatrix}. \tag{A.18}$$

[1] The modifications required for $i > k$ are trivial and are not presented herein.
[2] First derived by Jacobi [57]; later introduced as a numerical tool by Givens [50].

Property A.3.5. *Let* $\mathbf{Q} \in \mathbb{R}^{n \times n}$ *be an orthogonal transform and* $\boldsymbol{x}$ *and* $\boldsymbol{y}$ *be two vectors in* $\mathbb{R}^n$*. The multiplication with* $\mathbf{Q}$ *of both vectors preserves the inner product* $\boldsymbol{x}^T\boldsymbol{y}$*,*

$$(\mathbf{Q}\boldsymbol{x})^T(\mathbf{Q}\boldsymbol{y}) = \boldsymbol{x}^T\mathbf{Q}^T\mathbf{Q}\boldsymbol{y} = \boldsymbol{x}^T\boldsymbol{y}. \tag{A.19}$$

Property A.3.6. *Let* $\boldsymbol{x}$ *and* $\mathbf{y}$ *be two vectors in* $\mathbb{R}^n$ *and* $\mathbf{Q} \in \mathbb{R}^{n \times n}$ *be an orthogonal transform. Then if we denote* $\boldsymbol{z} = \mathbf{Q}\boldsymbol{x}$ *and* $\boldsymbol{w} = \mathbf{Q}\boldsymbol{y}$ *we have that*

$$\boldsymbol{z}_{i:n}^T\boldsymbol{w}_{i:n} = \boldsymbol{x}_{i:n}^T\boldsymbol{y}_{i:n} + \boldsymbol{x}_{1:i-1}^T\boldsymbol{y}_{1:i-1} - \boldsymbol{z}_{1:i-1}^T\boldsymbol{w}_{1:i-1}, \tag{A.20}$$

for any $1 \leq i \leq n$*.*

References

[P1] A. Onose and B. Dumitrescu. Adaptive matching pursuit using coordinate descent and double residual minimization. *Signal Processing*, 93(11):3143–3150, November 2013.

[P2] A. Onose and B. Dumitrescu. Distributed coordinate descent using adaptive matching pursuit. In *Proceedings of the International Symposium on Intelligent Signal Processing and Communication Systems*, pages 513–518, Naha, Japan, November 2013.

[P3] A. Onose and B. Dumitrescu. Cyclic adaptive matching pursuit. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 3745–3748, Kyoto, Japan, March 2012.

[P4] A. Onose and B. Dumitrescu. Low complexity approximate cyclic adaptive matching pursuit. In *Proceedings of the European Signal Processing Conference*, pages 2629–2633, Bucharest, Romania, August 2012.

[P5] B. Dumitrescu, A. Onose, P. Helin, and I. Tăbuş. Greedy sparse RLS. *IEEE Transaction on Signal Processing*, 60(5):2194–2207, May 2012.

[P6] A. Onose, B. Dumitrescu, and I. Tăbuş. Sliding window greedy RLS for sparse filters. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3916–3919, Prague, Czech Republic, May 2011.

[P7] A. Onose and B. Dumitrescu. Group greedy RLS sparsity estimation via information theoretic criteria. In *Proceedings of the International Conference on Control Systems and Computer Science*, volume 2, pages 359–364, Bucharest, Romania, May 2013.

[8] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19(6):716–723, 1974.

[9] A.E. Albert and Gardner L.S. JR. *Stochastic Approximation and Nonlinear Regression*. Cambridge, Massachusetts Technology Press of the Massachusetts Institute of Technology, 1967.

[10] D. Angelosante, J.A. Bazerque, and G.B. Giannakis. Online adaptive estimation of sparse signals: Where RLS meets the l_1-norm. *IEEE Transaction on Signal Processing*, 58(7):3436–3447, July 2010.

[11] D. Angelosante and G.B. Giannakis. RLS-weighted LASSO for adaptive estimation of sparse signals. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 3245–3248, Taipei, Taiwan, April 2009.

[12] B. Babadi, N. Kalouptsidis, and V. Tarokh. SPARLS: The sparse RLS algorithm. *IEEE Transaction on Signal Processing*, 58(8):4013–4025, August 2010.

[13] J. Benesty and S.L. Gay. An improved PNLMS algorithm. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 1881–1884, Orlando, USA, May 2002.

[14] A.M. Bruckstein, D.L. Donoho, and M. Elad. From sparse solutions of systems of equations to sparse modeling of signals and images. *Society for Industrial and Applied Mathematics Review*, 51(1):34–81, 2009.

[15] O. Bryt and M. Elad. Compression of facial images using the K-SVD algorithm. *Journal of Visual Communication and Image Representation*, 19(4):270–282, May 2008.

[16] E. J. Candes, J. Romberg, and T. Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, February 2006.

[17] E.J. Candes and T. Tao. Decoding by linear programming. *IEEE Transactions on Information Theory*, 51(12):4203–4215, 2005.

[18] E.J Candes and T. Tao. The Dantzig selector: statistical estimation when p is much larger than n. *Annals of Statistics*, 35(6):2313–2351, 2007.

[19] E.J. Candes, M.B. Wakin, and S.P. Boyd. Enhancing sparsity by reweighted ℓ_1 minimization. *Journal of Fourier Analysis and Applications*, 14(5-6):877–905, 2008.

[20] F.S. Cattivelli, C.G. Lopes, and A.H. Sayed. Diffusion recursive least-squares for distributed estimation over adaptive networks. *IEEE Transactions on Signal Processing*, 56(5):1865–1877, 2008.

[21] F.S. Cattivelli and A.H. Sayed. Diffusion LMS strategies for distributed estimation. *IEEE Transactions on Signal Processing*, 58(3):1035–1048, 2010.

[22] S. Chen, S.A. Billings, and W. Luo. Orthogonal least squares methods and their application to non-linear system identification. *Intentional Journal on Control*, 50(5):1873–1896, 1989.

[23] S.S. Chen, D.L. Donoho, M. Saunders, and A. Saunders. Atomic decomposition by basis pursuit. *SIAM Journal on Scientific Computing*, 20:33–61, 1998.

[24] Y. Chen, Y. Gu, and A.O. Hero. Sparse LMS for system identification. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 3125–3128, Taipei, Taiwan, April 2009.

[25] Y. Chen and A.O. Hero III. Recursive $\ell_{1,\infty}$ group LASSO. *IEEE Transactions on Signal Processing*, 60(8):3978–3987, August 2012.

[26] S. Chouvardas, K. Slavakis, Y. Kopsinis, and S. Theodoridis. A sparsity promoting adaptive algorithm for distributed learning. *IEEE Transactions on Signal Processing*, 60(10):5412–5425, 2012.

[27] M.G. Christensen and S.H. Jensen. The cyclic matching pursuit and its application to audio modeling and coding. In *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, pages 550–554, Nov. 2007.

[28] J.M. Cioffi and T. Kailath. Fast recursive-least-squares transversal filters for adaptive filtering. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 32(2):304–337, 1984.

[29] S.F. Cotter and B.D. Rao. The adaptive matching pursuit algorithm for estimation and equalization of sparse time-varying channels. In *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, volume 2, pages 1772–1776, 2000.

[30] S.F. Cotter and B.D. Rao. Sparse channel estimation via matching pursuit with application to equalization. *IEEE Transactions on Communications*, 50(3):374–377, 2002.

[31] P. Di Lorenzo and A.H. Sayed. Sparse distributed learning based on diffusion adaptation. *IEEE Transactions on Signal Processing*, 61(6):1419–1433, 2013.

[32] D.L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

[33] D.L. Donoho and M. Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via l_1 minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202, 2003.

[34] D.L. Donoho and X. Huo. Uncertainty principles and ideal atomic decomposition. *IEEE Transactions on Information Theory*, 47(7):2845–2862, 2001.

[35] B. Dumitrescu and I. Tăbuş. Greedy RLS for sparse filters. In *Proceedings of the European Signal Processing Conference*, pages 1484–1488, 2010.

[36] D.L. Duttweiler. Proportionate normalized least-mean-squares adaptation in echo cancelers. *IEEE Transactions on Speech and Audio Processing*, 8(5):508–518, 2000.

[37] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *Annals of Statistics*, 32:407–499, 2004.

[38] M. Elad. *Sparse and Redundant Representations - From Theory to Applications in Signal and Image Processing*. Springer, 2010.

[39] M. Elad. Sparse and redundant representation modeling - what next? *IEEE Signal Processing Letters*, 19(12):922–928, 2012.

[40] Y.C. Eldar, P. Kuppinger, and H. Bolcskei. Block-sparse signals: uncertainty relations and efficient recovery. *IEEE Transactions on Signal Processing*, 58(6):3042–3054, June 2010.

[41] M.A. Figueiredo and R.D. Nowak. An EM algorithm for wavelet-based image restoration. *IEEE Transactions on Image Processing*, 12(1):906–916, 2003.

[42] J.H. Friedman, T. Hastie, and R. Tibshirani. Pathwise coordinate optimization. *Annals of Applied Statistics*, 1(2):302–332, 2007.

[43] J.H. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

[44] T. Gansler, S.L. Gay, M. Sondhi, and J. Benesty. Double-talk robust fast converging algorithms for network echo cancellation. *IEEE Transactions on Speech and Audio Processing*, 8(6):656–663, 2000.

[45] C.F. Gauss. *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium*. Hamburg, 1809.

[46] W.M. Gentleman and H.T. Kung. Matrix triangularization by systolic arrays. In *Proceedings of the International Society for Optics and Photonics*, volume 298, Real-Time Signal Processing IV, pages 298–303, 1981.

[47] D. Giacobello, M.G. Christensen, J. Dahl, S.H. Jensen, and M. Moonen. Sparse linear predictors for speech processing. In *Proceedings of the ISCA InterSpeech Conference*, pages 1353–1356, 2008.

[48] D. Giacobello, M.G. Christensen, M.N. Manohar N. Murthi, S.H. Jensen, and M. Moonen. Sparse linear prediction and its applications to speech processing. *IEEE Transactions on Audio, Speech and Language Processing*, 20(5):1644–1657, 2012.

[49] C.D. Giurcaneanu and S.A. Razavi. AR order selection in the case when the model parameters are estimated by forgetting factor least-squares algorithms. *Signal Processing*, 90(2):451–466, 2010.

[50] W. Givens. Computation of plane unitary rotations transforming a general matrix to triangular form. *Journal of the Society for Industrial and Applied Mathematics*, 6(1):26–50, 1958.

[51] I.F. Gorodnitsky and B.D. Rao. Sparse signal reconstruction from limited data using FOCUSS: a re-weighted minimum norm algorithm. *IEEE Transactions on Signal Processing*, 45(3):600–616, 1997.

[52] E.J. Hannan, A.J. McDougall, and D.S. Poskitt. Recursive estimation of autoregressions. *Journal of the Royal Statistical Society. Series B*, 51(2):217–233, 1989.

[53] S. Haykin. *Adaptive Filter Theory*. Prentice Hall, 2002.

[54] Y. Hel-Or and D. Shaked. A discriminative approach for wavelet denoising. *IEEE Transactions on Image Processing*, 17(4):443–457, 2008.

[55] A.S. Householder. Unitary triangularization of a nonsymmetric matrix. *Journal of the Association for Computing Machinery*, 5(4):339–342, October 1958.

[56] K. Hwanjoon and B.D. Rao. On the benefits of the block-sparsity structure in sparse signal recovery. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3685–3688, March 2012.

[57] C.G.J. Jacobi. Über ein leichtes verfahren, die in der theorie der säkularstörungen vorkommenden gleichungen numerisch aufzulösen. *Journal für die reine und angewandte Mathematik*, 30:51–94, 1846.

[58] R.E. Kalman. A new approach to linear filtering and prediction problems. *ASME Journal of Basic Engineering*, 82, series D:35–45, 1960.

[59] N. Kalouptsidis, G. Mileounis, B. Babadi, and V. Tarokh. Adaptive algorithms for sparse system identification. *Signal Processing*, 91(8):1910–1919, 2011.

[60] G.Z. Karabulut and A. Yongacoglu. Estimation of time-varying channels with orthogonal matching pursuit algorithm. In *Proceedings of the IEEE Symposium on Advances in Wired and Wireless Communication*, pages 141–144, 2005.

[61] A.N. Kolmogorov. Sur l'interpolation et l'extrapolation des suites stationnaires. *Comptes Rendus de l'Acadamie des Sciences*, 208:2043–2045, 1939.

[62] Y. Kopsinis, K. Slavakis, and S. Theodoridis. Online sparse system identification and signal reconstruction using projections onto weighted l_1 balls. *IEEE Transaction on Signal Processing*, 59(3):936–952, March 2011.

[63] A.-M. Legendre. Méthode des moindres carrés, pour trouver le milieu le plus probable entre les résultats de différentes observations. *Memoires Présentés par Divers Savants à la l'Académie des Sciences de l'Institut de France*, pages 149–154, 1819.

[64] M.P. Mahon, L.H. Sibul, and H.M. Valenzuela. A sliding window update for the basis matrix of the QR decomposition. *IEEE Transactions on Signal Processing*, 41(5):1951–1953, May 1993.

[65] D. Malioutov, M. Cetin, and A.S. Willsky. A sparse signal reconstruction perspective for source localization with sensor arrays. *IEEE Transactions on Signal Processing*, 53(8):3010–3022, 2005.

[66] S.G. Mallat and Z. Zhang. Matching pursuit with time frequency dictionaries. *IEEE Transactions in Signal Processing*, 41(12):3397–3415, 1993.

[67] G. Mateos, I.D. Schizas, and G.B. Giannakis. Distributed recursive least-squares for consensus-based in-network adaptive estimation. *IEEE Transactions on Signal Processing*, 57(11):4583–4588, 2009.

[68] J.F.C. Mota, J. Xavier, P.M.Q. Aguiar, and M. Puschel. Distributed basis pursuit. *IEEE Transactions on Signal Processing*, 60(4):1942–1956, 2012.

[69] J. Nagumo and A. Noda. A learning method for system identification. *IEEE Transactions on Automatic Control*, 12(3):282–287, 1967.

[70] P.A. Naylor, J. Cui, and M. Brookes. Adaptive algorithms for sparse echo cancellation. *Signal Processing*, 86(6):1182–1192, 2006.

[71] D. Needell and J.A. Tropp. CoSaMP: Iterative signal recovery from incomplete and inaccurate samples. *Applied and Computational Harmonic Analysis*, 26(3):301–321, 2009.

[72] S.N. Negahban and M.J. Wainwright. Simultaneous support recovery in high dimensions: benefits and perils of block ℓ_1/ℓ_∞-regularization. *IEEE Transactions on Information Theory*, 57(6):3841–3863, June 2011.

[73] M. Niedzwiecki. Bayesian-like autoregressive spectrum estimation in the case of unknown process order. *IEEE Transactions on Automatic Control*, 30(10):950–961, 1985.

[74] F. Parvaresh, H. Vikalo, S. Misra, and B. Hassibi. Recovering sparse signals using sparse measurement matrices in compressed DNA microarrays. *IEEE Selected Topics in Signal Processing*, 2(3):275–285, June 2008.

[75] Y.C. Pati, R. Rezaiifar, and P.S. Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Proceedings of the Asilomar Conference on Signals, Systems, and Computers*, pages 40–44, 1993.

[76] R.L. Plackett. Some theorems in least squares. *Biometrika*, 37(1/2):149–157, 1950.

[77] L. Rebollo-Neira and D. Lowe. Optimized orthogonal matching pursuit approach. *IEEE Signal Processing Letters*, 9(4):137–140, 2002.

[78] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[79] J. Rissanen. Order estimation by accumulated prediction errors. *Journal of Applied Probability*, 23:55–61, 1986.

[80] A.H. Sayed. *Fundamentals of Adaptive Filtering*. John Wiley & Sons, 2003.

[81] A.H. Sayed and T. Kailath. A state-space approach to adaptive RLS filtering. *IEEE Signal Processing Magazine*, 11(3):18–60, 1994.

[82] I.D. Schizas, G. Mateos, and G.B. Giannakis. Distributed LMS for consensus-based in-network adaptive processing. *IEEE Transactions on Signal Processing*, 57(6):2365–2382, 2009.

[83] G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 6(2):461–464, 1978.

[84] B.L. Sturm, M. Christensen, and R. Gribonval. Cyclic pure greedy algorithms for recovering compressively sampled sparse signals. In *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, pages 1143–1147, 2011.

[85] B.L. Sturm and M.G. Christensen. Cyclic matching pursuits with multiscale time-frequency dictionaries. In *Proceedings of the Asilomar Conference on Signals, Systems and Computers*, pages 581–585, 2010.

[86] V.N. Temlyakov. Weak greedy algorithms. *Advances in Computational Mathematics*, 12(2-3):213–227, 2000.

[87] R. Tibshirani. Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society, Series B*, 58:267–288, 1994.

[88] A. N. Tikhonov and V. Y. Arsenin. *Solutions of Ill-Posed Problems*. Winston & Sons, Washington, D.C, 1977.

[89] J.A. Tropp. Greed is good: algorithmic results for sparse approximation. *IEEE Transactions on Information Theory*, 50(10):2231–2242, 2004.

[90] J.A. Tropp. Just relax: convex programming methods for identifying sparse signals in noise. *IEEE Transactions on Information Theory*, 52(3):1030–1051, 2006.

[91] L.R. Vega, H. Rey, J. Benesty, and S. Tressens. A new robust variable step-size NLMS algorithm. *IEEE Transactions on Signal Processing*, 56(5):1878–1893, 2008.

[92] L.R. Vega, H. Rey, J. Benesty, and S. Tressens. A family of robust algorithms exploiting sparsity in adaptive filters. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(4):572–581, May 2009.

[93] T. Walzman and M. Schwartz. A projected gradient method for automatic equalization in the discrete frequency domain. *IEEE Transactions on Communications*, 21(12):1442–1446, 1973.

[94] B. Widrow. Adaptive filters. In *Aspects of Network and System Theory*, pages 563–586. Holt, Rinehart and Winston, 1971.

[95] B. Widrow and M.E. Jr. Hoff. Adaptive switching circuits. *Institute of Radio Engineers - Western Electronic Show and Convention Record*, 4:96–104, August 1960.

[96] N. Wiener. *Extrapolation, Interpolation and Smoothing of Stationary Times Series with Engineering Applications*. Cambridge, Massachusetts Technology Press of the Massachusetts Institute of Technology, 1949. *(first published as a classified National Defense Research Report in 1942)*.

[97] Max A. Woodbury. *Inverting Modified Matrices*. Number 42 in Statistical Research Group Memorandum Reports. Princeton University, Princeton, NJ, 1950.

[98] Y.V. Zakharov and V.H. Nascimento. DCD-RLS adaptive filters with penalties for sparse identification. *IEEE Transactions on Signal Processing*, 61(12):3198–3213, 2013.

[99] Y.V. Zakharov, G.P. White, and J. Liu. Low-complexity RLS algorithms using dichotomous coordinate descent iterations. *IEEE Transactions on Signal Processing*, 56(7):3150–3161, 2008.

Publications

Publication 1

Reprinted from

[P1] A. Onose and B. Dumitrescu. Adaptive matching pursuit using coordinate descent and double residual minimization. *Signal Processing*, 93(11):3143–3150, November 2013.

Adaptive Matching Pursuit Using Coordinate Descent and Double Residual Minimization

Alexandru Onose[1], Bogdan Dumitrescu[1]

Abstract

We present a greedy recursive algorithm for computing sparse solutions to systems of linear equations. Derived from adaptive matching pursuit, the algorithm employs a greedy column selection strategy which, combined with coefficient update via coordinate descent, ensures a low complexity. The sparsity level is estimated online using the predictive least squares (PLS) criterion. The key to performance is the minimization of two residuals, corresponding to two solutions with different sparsity levels, one for finding the values of the nonzero coefficients, the other for maintaining a large enough pool of candidates for the PLS criterion. We test the algorithm for a sparse time-varying finite impulse response channel; the performance is comparable with or better than that of the competing methods, while the complexity is lower.

Keywords: sparse filters, matching pursuit, adaptive algorithm, greedy method, channel identification

1. Introduction

In recent years numerous problems like compression, prediction, array processing, channel identification or echo cancellation have generated much interest in the development of algorithms for recursively finding sparse solutions to systems of linear equations [1, 2]. The search for methods that produce sparse solutions began with the development of batch algorithms like the basis pursuit [3], the least absolute shrinkage and selection operator [4] or the orthogonal least squares [5]. In practice however, the data is often available sequentially and thus adaptive algorithms that compute the solution recursively are much more efficient. The different adaptive methods proposed range from convex relaxation techniques [6, 7, 8] to algorithms based on projections onto convex sets [9] or greedy methods [10, 11]. With roots in the traditional adaptive filtering, sparsity aware LMS algorithms have also been developed [12, 13].

The aim of this paper is to present a new adaptive greedy algorithm that uses a selection strategy inspired from [11], but now coupled with coefficient update based on coordinate descent, thus having low complexity without negatively affecting the performance.

Let us consider a typical finite impulse response (FIR) channel identification problem where the input $u(t)$ and the output $d(t)$ are known at each time instance t. We desire to estimate the true coefficients h_j that result from the minimization of the estimation error

$$e(t) = d(t) - \sum_{j=0}^{N-1} h_j u(t-j). \tag{1}$$

Using an exponential window with forgetting factor $0 < \lambda \leq 1$, the least-squares criterion to be minimized is

$$J(t) = \|\boldsymbol{b}^{(t)} - \boldsymbol{A}^{(t)}\boldsymbol{x}^{(t)}\|^2. \tag{2}$$

The solution $\boldsymbol{x}^{(t)}$ resulting from the minimization of $J(t)$ is the estimate of the vector $\boldsymbol{h}$ of true coefficients, which is assumed to be sparse, with $L_t \ll N$ nonzero elements. The matrix $\boldsymbol{A}^{(t)} \in \mathbb{R}^{t\times N}$, constructed using the input data, has the i-th row equal to $\lambda^{\frac{t-i}{2}}\boldsymbol{\alpha}^{(i)T}$ with

$$\boldsymbol{\alpha}^{(i)} = [u(i)\ u(i-1)\ \ldots\ u(i-N+1)]^T. \tag{3}$$

The vector $\boldsymbol{b}^{(t)} \in \mathbb{R}^t$ is composed of the exponentially weighted output data $\boldsymbol{b}_i = \lambda^{\frac{t-i}{2}} d(i)$. The index $^{(i)}$ is used to indicate the data from time i; we define $\beta^{(i)} = d(i)$ for later use. We presume the environment to be slowly varying such that the solution does not change greatly between two consecutive time instances.

We develop an adaptive algorithm for finding a sparse minimizer of the criterion (2). The most remote ancestors of our algorithm are the matching pursuit (MP) algorithm [14] and its adaptive counterpart, the adaptive matching pursuit (AMP) [10]. The solution is updated at each time t via coordinate descent. This technique is used in other adaptive algorithms; it can be tailored for finding either full solutions like in [15] or sparse solutions, for example in [8]. A related algorithm is the (batch) cyclic MP [16, 17],

[1]The authors are with Department of Signal Processing, Tampere University of Technology, Finland. Email: firstname.lastname@tut.fi. B. Dumitrescu is also with Department of Automatic Control and Computers, University Politehnica of Bucharest, Romania. This work has been supported by a Tekes FiDiPro grant and by GETA.

which uses several rounds of coordinate descent at each step of the greedy search. We have presented an adaptive version of cyclic MP [18]; however, the current algorithm is quite different in the organization of the computation and does not repeat the optimization of a coefficient at the same time instance, but spreads it over time; hence, its complexity is lower.

Since we use some techniques from the GRLS algorithm [11], we discuss here the differences and similarities between this work and [11]. Our new algorithm requires a pseudo ordering of the columns from $\boldsymbol{A}$ that contribute to the solution; any method providing such an order may be used but we only present two approaches. The first is inspired from [11] and uses neighbor permutations, while the second, simpler, only tries to find the worst column. The neighbor selection strategy is used in [11] for a completely different underlying algorithm and it is one of the few viable methods that allow low complexity, while for the algorithm herein such restrictions do not apply.

We estimate online the sparsity level L_t using the predictive least squares criterion (PLS) [19], like in [11], although other model selection criteria exist [20, 21] and the Bayesian information criterion [20] is also used in [11]. However, the mechanism that is created to allow the use of PLS is new and is specifically tailored for the algorithm presented here. A distinctive feature is the minimization of two residuals related to criterion (2), corresponding to solutions with different sparsity levels; this also implies some modifications in the column selection strategy, compared to [11]. Finally, the coefficient estimation strategy is completely different from the orthogonalization from [11]; here, the coordinate descent ensures a good approximation of the least-squares solution, with significantly lower complexity than in [11]; the only similarity is that we use a fixed number of scalar products instead of the indefinitely long matrix $\boldsymbol{A}$ and vector $\boldsymbol{b}$, but this is a standard storage technique.

The contents of this paper is as follows. In section 2 we present our proposed algorithm for computing a sparse solution with fixed sparsity level. In section 3 we describe our procedure based on the minimization of two residuals for the online estimation of the sparsity level L_t and the computation of the solution. In section 4 we discuss the complexity of our algorithm, compared to that of other algorithms. Section 5 contains the simulation results used to validate the performance.

2. Fixed-sparsity level adaptive matching pursuit with coordinate descent

We describe the basic operation of our algorithm assuming that, at time t, we have computed an m-sparse solution $\boldsymbol{x}^{(t)}$, permuted such that its nonzero elements are in the first m positions, which are named active. The columns of the matrix $\boldsymbol{A}^{(t)}$ are permuted accordingly; we will not show the permutation explicitly, but only explain how it changes. The value $m \geq L_t$ is assumed to be fixed in this section. The residual corresponding to the solution $\boldsymbol{x}^{(t)}$ is

$$\boldsymbol{r}_m^{(t)} = \boldsymbol{b}^{(t)} - \sum_{i=1}^{m} x_i^{(t)} \boldsymbol{a}_i^{(t)}, \tag{4}$$

where $\boldsymbol{a}_i^{(t)}$ is the i-th column of (the permuted) $\boldsymbol{A}^{(t)}$.

At time $t+1$, the new (permuted) data are appended

$$\boldsymbol{b}^{(t+1)} = \begin{bmatrix} \sqrt{\lambda}\boldsymbol{b}^{(t)} \\ \beta^{(t+1)} \end{bmatrix}, \quad \boldsymbol{A}^{(t+1)} = \begin{bmatrix} \sqrt{\lambda}\boldsymbol{A}^{(t)} \\ \boldsymbol{\alpha}^{(t+1)^T} \end{bmatrix}. \tag{5}$$

To ease the notation, from now on we remove the upper index $t+1$ that marks the variables affected by current computations. Using the solution from time t, the new residual is

$$\boldsymbol{r}_{m,0} = \boldsymbol{b} - \sum_{i=1}^{m} x_i^{(t)} \boldsymbol{a}_i = \begin{bmatrix} \boldsymbol{r}_m^{(t)} \\ \beta - \sum_{i=1}^{m} x_i^{(t)} \alpha_i \end{bmatrix}. \tag{6}$$

At time $t+1$ we perform two tasks: i) we manage the order of the active positions and ii) we update the values of the solution $\boldsymbol{x}$. Task i) is typical to greedy algorithms and MP in particular; it aims to identify the nonzero coefficients and order them based on their importance. To reduce complexity and relying on the slow variability of the channel, we use for now the search scheme from [11] that allows order changes almost only between neighbors; the only exception is the last position in the active set, for which we permit all the inactive columns to compete with the current active column occupying that position. Task ii) is performed by a simple coordinate descent on each of the active positions, optimizing the residual corresponding to the m-sparse solution. The two tasks are intertwined and act successively on each active coefficient.

To update the coefficient i, with $i < m$, we compute a partial residual with the two neighbor positions i and $i+1$ temporarily removed from the active set:

$$\tilde{\boldsymbol{r}}_{m,i} = \boldsymbol{r}_{m,i-1} + x_i^{(t)} \boldsymbol{a}_i + x_{i+1}^{(t)} \boldsymbol{a}_{i+1}. \tag{7}$$

The column that takes position i is decided by looking at the best alignment with this residual, i.e. with the standard MP criterion

$$k = \arg\max_{l \in \mathcal{S}} \frac{|\boldsymbol{a}_l^T \tilde{\boldsymbol{r}}_{m,i}|^2}{\|\boldsymbol{a}_l\|^2}, \tag{8}$$

with $\mathcal{S} = \{i, i+1\}$. If position $i+1$ is better, then positions i and $i+1$ are permuted. The optimization of coefficient x_i is a coordinate descent on the least-squares criterion (2), by projecting the residual (after excluding position i from the active set) on the i-th column:

$$x_i = \frac{\boldsymbol{a}_i^T (\boldsymbol{r}_{m,i-1} + x_i^{(t)} \boldsymbol{a}_i)}{\|\boldsymbol{a}_i\|^2} = x_i^{(t)} + \mu, \tag{9}$$

with $\mu = \frac{\boldsymbol{a}_i^T \boldsymbol{r}_{m,i-1}}{\|\boldsymbol{a}_i\|^2}$. This operation takes place immediately after the index in position i is found via (8). The new residual is $\boldsymbol{r}_{m,i} = \boldsymbol{r}_{m,i-1} - \mu \boldsymbol{a}_i$.

For the last active column ($i = m$), the search for the best column is performed with (8), but now on the index set $\mathcal{S} = m : N$ and with the partial residual

$$\tilde{\boldsymbol{r}}_{m,m} = \boldsymbol{r}_{m,m-1} + x_m^{(t)} \boldsymbol{a}_m, \tag{10}$$

because $x_k^{(t)} = 0$, $k > m$. The optimal coefficient is

$$x_m = \frac{\boldsymbol{a}_k^T \tilde{\boldsymbol{r}}_{m,m}}{\|\boldsymbol{a}_k\|^2} \tag{11}$$

and the new residual is $\boldsymbol{r}_{m,m} = \tilde{\boldsymbol{r}}_{m,m} - x_m \boldsymbol{a}_k$. Note that if an inactive position becomes active ($k > m$), then the influence of the former column m must be removed from the residual, which explains the use of $\tilde{\boldsymbol{r}}_{m,m}$ from (10) in the above formulas. After these computations, the matrix $\boldsymbol{A}$ is permuted accordingly.

Because the matrix $\boldsymbol{A}$ grows indefinitely in time, it is impossible to store it explicitly. We implement all the above operations using only the scalar products $\boldsymbol{\Phi} = \boldsymbol{A}^T \boldsymbol{A}$ between the columns of $\boldsymbol{A}$ and the scalar products $\boldsymbol{\Psi} = \boldsymbol{A}^T \boldsymbol{r}_m$ between the data matrix and the residual of the m-sparse solution. The transformation is simple: each equation involving a residual is multiplied by $\boldsymbol{A}^T$. The main algorithm, called CD-AMP (Coordinate Descent Adaptive Matching Pursuit), is shown in the table titled Alg. 1; the first operation (at current time) is the update (12) of the scalar products, based on (5) and (6). Then, Alg. 2 is called $m - 1$ times for the active positions that are managed with neighbor search; to illustrate the transformation into scalar product operations, note that steps 1 and 2 of the algorithm correspond to (7) and (8), respectively. It can easily be observed that the scalar product form is obtained by multiplying (7) to the left with $\boldsymbol{A}^T$ and that (8) is already expressed using only scalar products. Finally, Alg. 3 is called for setting the last active position; any inactive column may become active at this stage, the search from step 2 being performed over the whole inactive column set.

Fig. 1 gives a graphical image of CD-AMP. Each search and update operation is represented by a horizontal line with one or several ticks showing the columns that compete for the current position, marked with an arrow. The arrow also means an update operation. Active columns are represented by blue bullets, inactive columns by black bullets. The numbers on the left belong to the steps in CD-AMP. On the right, we remind that the (product scalars with the) residual are updated at each step.

The neighbor permutation is not the only column ordering strategy that can be applied. A sweep of any sorting algorithm based on permutations may be used. The main requirement is that it eventually moves the worst active column into the last position m such that it may be replaced when Alg. 3 is called. Thus, we also can use a column ordering strategy that involves permuting between the column m and all other columns from $m-1$ to 1. The modifications to Alg. 1 are straightforward, the loop from step 2 is performed in reversed order and Alg. 2 is called now for i and m. This strategy does not produce an ordered active set; it only ensures that the worst column is moved into the last position.

Alg. 1. (CD-AMP: Coordinate descent adaptive matching pursuit)

1 update scalar products with current data permuted accordingly

$$\begin{aligned} \boldsymbol{\Phi}_{1:N,1:N} &\leftarrow \lambda \boldsymbol{\Phi}_{1:N,1:N} + \boldsymbol{\alpha}_{1:N} \boldsymbol{\alpha}_{1:N}^T \\ \boldsymbol{\Psi}_{1:N} &\leftarrow \lambda \boldsymbol{\Psi}_{1:N} + \boldsymbol{\alpha}_{1:N} (\beta - \textstyle\sum_{i=1}^{m} x_i \alpha_i) \end{aligned} \tag{12}$$

2 for $i = 1 : m-1$

2.1 find and update coefficient on position i*, Alg. 2* $(i, i+1)$

3 estimate coefficient on position m*, Alg. 3* $(m, m+1 : N)$

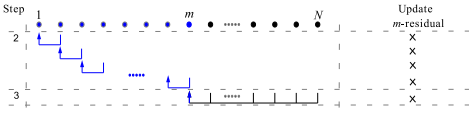


Figure 1: Graphical representation of CD-AMP operations.

3. Support cardinality estimation and full algorithm

In the CD-AMP algorithm, we have made no assumptions on the true number L_t of nonzero elements of the solution $\boldsymbol{x}$. We still assume for a while that we run CD-AMP with $m \geq L_t$, but will later refine the algorithm. After enough data is available, the algorithm will hope-

Alg. 2. (Update coefficient i, search between columns i and j)
inputs: i, j

1 remove column i *and* j *influence from the residual*
$\tilde{\boldsymbol{\Psi}}_{\{i,j\}} = \boldsymbol{\Psi}_{\{i,j\}} + x_i \boldsymbol{\Phi}_{\{i,j\},i} + x_j \boldsymbol{\Phi}_{\{i,j\},j}$
2 find best i*-th column searching between* i *and* j
$k = \arg\max_{l \in \{i,j\}} \tilde{\Psi}_l^2 / \Phi_{l,l}$
3 if $k \neq i$ *(column* j *is better)*

3.1 swap columns (and lines) i *and* j *in* $\boldsymbol{\Phi}$
swap elements i *and* j *in* $\boldsymbol{x}$ *and* $\boldsymbol{\Psi}$

4 compute coefficient update $\mu = \Psi_i / \Phi_{i,i}$
5 update coefficient and corresponding scalar products
$x_i \leftarrow x_i + \mu, \quad \boldsymbol{\Psi}_{1:N} \leftarrow \boldsymbol{\Psi}_{1:N} - \mu \boldsymbol{\Phi}_{1:N,i}$

Alg. 3. (Estimate coefficient i, search in the set $\mathcal{S}$)

inputs: i, $\mathcal{S}$

1 remove last active column influence from the residual
$\boldsymbol{\Psi}_{1:N} \leftarrow \boldsymbol{\Psi}_{1:N} + x_i \boldsymbol{\Phi}_{1:N,i}$
2 find best candidate column searching all columns in $\mathcal{S}$
$k = \arg\max_{l \in \{i,\mathcal{S}\}} \Psi_l^2 / \Phi_{l,l}$
3 if $k > i$ *(another column is better than i)*
3.1 swap columns (and lines) i *and* k *in* $\boldsymbol{\Phi}$
swap elements i *and* k *in* $\boldsymbol{\Psi}$
4 evaluate coefficient value $x_i = \Psi_i / \Phi_{i,i}$
5 update scalar product based $\boldsymbol{\Psi}$
$\boldsymbol{\Psi}_{1:N} \leftarrow \boldsymbol{\Psi}_{1:N} - x_i \boldsymbol{\Phi}_{1:N,i}$

Alg. 4. (DCD-AMP: Double residual CD-AMP)

1 update scalar products with current data permuted accordingly, equation (12) with $m = L$
2 update scalar product differences between L-residual and M-residual scalar products
$\boldsymbol{\delta}_{1:N} \leftarrow \lambda \boldsymbol{\delta}_{1:N} + \boldsymbol{\alpha}_{1:N} \sum_{i=L+1}^{M} x_i \alpha_i$
3 for $i = 1 : L-1$
3.1 find and update coefficient on position i*, Alg. 2* $(i, i+1)$
4 estimate coefficient on position L, Alg. 3 $(L, L+1)$*; if column* $L+1$ *is better than* L
before step 3.1 *in Alg. 3, update scalar product difference* $\boldsymbol{\delta}$
$\boldsymbol{\delta} = \boldsymbol{\delta} + x_{L+1} \boldsymbol{\Phi}_{1:N,L+1} - x_L \boldsymbol{\Phi}_{1:N,L}$
in Alg. 3, step 3.1*, permute also* $\boldsymbol{x}$
5 construct the M-residual scalar product $\boldsymbol{\Omega}$
$\boldsymbol{\Omega} = \boldsymbol{\Psi} - \boldsymbol{\delta}$
6 estimate coefficient on position $L+1$*, Alg. 5* $(L+1, L+2 : N)$
7 for $i = L+2 : M-1$
7.1 find and update coefficient on position i*, Alg. 2* $(i, i+1)$ *with* $\boldsymbol{\Omega}$ *instead of* $\boldsymbol{\Psi}$
8 for position M only apply Alg. 2, steps 4 and 5, with $i = M$ *and* $\boldsymbol{\Omega}$ *instead of* $\boldsymbol{\Psi}$
9 estimate L using the PLS criterion; if L or M change, update scalar products $\boldsymbol{\Psi}$ *and* $\boldsymbol{\Omega}$
10 compute the differences between L-residual and M-residual scalar products
$\boldsymbol{\delta} = \boldsymbol{\Psi} - \boldsymbol{\Omega}$

fully select the nonzero locations of the true solution on the first L_t positions. Relying on this inherent order and similarly to [11], we use the predictive least squares (PLS) criterion to compute an estimate $\hat{L}$ of L_t. The PLS criterion [19], at time t, is defined as

$$\mathbf{PLS}_c^{(t)} = \sum_{i=0}^{t} \lambda^{t-i} e_c^{(i)2}, \tag{13}$$

where $e_c^{(i)} = \beta^{(i)} - \sum_{j=1}^{c} \alpha_j^{(i)} x_j^{(i-1)}$ is the a priori estimation error at time i produced by the c-sparse solution at time $i-1$; note that this solution is made of the first c elements of the current m-sparse solution, hence the error can be cheaply computed. Only $O(m)$ operations are necessary at each time instance if the criterion is recursively computed as

$$\mathbf{PLS}_c^{(t)} = \lambda \cdot \mathbf{PLS}_c^{(t-1)} + e_c^{(t)2}. \tag{14}$$

An estimate of the true sparsity level is given by

$$\hat{L} = \arg \min_{c=1:m} \mathbf{PLS}_c^{(t)}. \tag{15}$$

However, unlike [11] where arbitrary changes are allowed, we only change the estimated sparsity level by at most 1 at each time, setting $\hat{L}^{(t+1)} = \hat{L}^{(t)} + \text{sign}(\hat{L} - \hat{L}^{(t)})$. It was noticed that this choice ensures smoother tracking. The estimate $\hat{L}^{(t+1)}$, denoted simply L in what follows, is used in the algorithm at time $t+1$, as discussed below.

We present now the main features of our AMP algorithm with support estimation, named Double CD-AMP (DCD-AMP), because it consists of running a modified version of CD-AMP with the goal of minimizing two residuals, $\boldsymbol{r}_L$, corresponding to an L-sparse solution, and $\boldsymbol{r}_M$, corresponding to an M-sparse solution, with $M > L$. The minimization of the residual $\boldsymbol{r}_L$ aims to compute an approximation of the true nonzero coefficients of the sparse solution $\boldsymbol{x}$, given the estimated sparsity level. The approximation is good when L is a good estimate of L_t. The residual $\boldsymbol{r}_M$ allows the calculation of the PLS criterion for solutions with assumed higher sparsity level, covering potential increases of L_t. The optimization of $\boldsymbol{r}_M$ alone would give less accurate estimation of the true nonzero coefficients, since other coefficients are also involved. However, the accuracy is sufficient to obtain good estimates L with the PLS criterion. So, the two residual optimizations work in tandem: through $\boldsymbol{r}_L$ we get accurate coefficients and through $\boldsymbol{r}_M$ an accurate sparsity level.

The residual $\boldsymbol{r}_L$ is used for computing the coefficients on positions $1:L$, while the residual $\boldsymbol{r}_M$ is used to estimate only the coefficients on positions $L+2:M$. The coefficient on position $L+1$ is also estimated using the residual $\boldsymbol{r}_L$, in order to provide a good value in case an increase of the sparsity level is required at the next time instance. However, not being included in the L-sparse solution, the $L+1$ coefficient does not modify $\boldsymbol{r}_L$, it only affects $\boldsymbol{r}_M$. To ensure computational efficiency, we use the residual difference $\boldsymbol{r}_L - \boldsymbol{r}_M$ to avoid the update of both residuals when a coefficient is updated. Since the M-sparse solution includes the L-sparse solution, updating the L-sparse solution does not change the residual difference. We will show in section 5 that, despite optimizing two residuals,

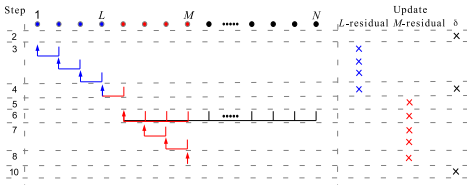


Figure 2: Graphical representation of DCD-AMP operations.

the complexity of DCD-AMP is only marginally higher than that of CD-AMP.

The positions $1:L$ are now named active, since they refer to the estimated nonzero coefficients; the positions $L+1:M$ are named pending and the remaining positions are inactive. We describe below the details that differentiate DCD-AMP from the the basic algorithm from Section 2. The full algorithm is listed as Alg. 4. It uses scalar products similarly to CD-AMP, namely $\boldsymbol{\delta} = \boldsymbol{A}^T(\boldsymbol{r}_L - \boldsymbol{r}_M)$, the scalar product of the residual difference $\boldsymbol{r}_L - \boldsymbol{r}_M$ with the columns from $\boldsymbol{A}$, $\boldsymbol{\Psi} = \boldsymbol{A}^T\boldsymbol{r}_L$, and $\boldsymbol{\Omega} = \boldsymbol{A}^T\boldsymbol{r}_M$. A graphical representation of DCD-AMP is given in Fig. 2, using the same conventions as in Fig. 1; color red is reserved for the pending positions. Additionally, the Matlab source code can be found at `http://www.cs.tut.fi/~bogdand/Software/cdamp.zip`.

The minimization of the residual $\boldsymbol{r}_L$ can be seen as the minimization of the least-squares criterion (2) based on the current estimate of the sparsity level. The modifications of CD-AMP for working with $\boldsymbol{r}_L$ are minimal. In particular, the computations for positions $1:L-1$ are not changed (Alg. 4, step 3). For position L, the operations are as in Alg. 3, but with position $L+1$ as the only alternative candidate. Since the column on position $L+1$ belongs to the M-sparse solution, if we include it in the L-sparse solution and move the column L to the M-sparse solution, we need to update the residual difference resulting in the extra operations from Alg. 4, step 4. Once the coefficient on position L is chosen and computed, $\boldsymbol{r}_L$ is not modified anymore and we can reconstruct the residual $\boldsymbol{r}_M$ based on the residual difference (Alg. 4, step 5).

For position $L+1$, all pending and inactive positions are candidates and Alg. 5, which is a modified version of Alg. 3 for the two residuals case, is used. The (updated) residual $\boldsymbol{r}_L$ is used in the MP criterion (8), for the best column search, and also for computing the coefficient value (Alg. 5, steps 1 and 3). The updated coefficient modifies only $\boldsymbol{r}_M$ since the first L coefficients were previously chosen (and $\boldsymbol{r}_L$ was properly updated). If a new position $k > L+1$ is found best, then the pending positions are shifted accordingly (if $k \leq M$, then positions $L+1:k-1$ move into $L+2:k$; if $k > M$, then positions $L+1:M-1$ move into $L+2:M$, hence the former position M is discarded from the pending set (Alg 5, step 2.3). Thus, depending on the position of the best candidate, the removed or recomputed coefficients modify the residual as in Alg. 5, steps 2 and 4. The remaining coefficients on positions $L+2:M-1$ are computed similarity to the CD-AMP algorithm but using and updating the residual $\boldsymbol{r}_M$ (Alg. 4, step 7). There is no open selection for position M, i.e. Alg. 3 is not called.

Finally, we estimate the current sparsity level L using the PLS criterion (Alg. 4, step 9). The parameter M is either fixed to a sufficiently large value, or variable, being computed as $M = L + \Delta$, where Δ is a small positive integer constant. If the value L is changed, then both residuals are changed in the variable M case (when M is also changed), but only $\boldsymbol{r}_L$ is changed in the fixed M case. As a last step, the residual difference is stored after which a new sample can be processed.

In addition to the DCD-AMP algorithm, we introduce two other versions of CD-AMP for the purpose of offering performance insight for the double residual scheme. The first algorithm, named 1CD-AMP, is the naive approach equivalent to running CD-AMP with $m = M$ but having the sparsity level $L \leq M$ estimated online using the PLS criterion; a single residual is optimized. The solution is composed of the first L out of the M computed coefficients. The second algorithm, named MCD-AMP, involves running CD-AMP M times, with $m = 1:M$, respectively; it generates a solution for each sparsity level independently. Similarly, the sparsity level estimate L is found online, but the solution is now given by the CD-AMP algorithm having $m = L$. Since it optimizes a distinct residual for each sparsity level, the MCD-AMP algorithm produces the best solution given by this class of algorithms. Note that 1CD-AMP requires an ordered solution, thus the second permutation strategy presented in Section 2 may not be applied in this case. For the MCD-AMP there are no difficulties in this regard.

4. Complexity

For the CD-AMP algorithm, the main computational burden is given by the update (12) of the scalar products $\boldsymbol{\Phi}$, which needs about $\mathcal{C}_{\boldsymbol{\Phi}} = \frac{3}{2}N^2$ operations due to the matrix symmetry (additions and multiplications are counted separately). In many applications, the matrix $\boldsymbol{A}$ has special structure because of time shifted input data as in (3) (e.g. the FIR channel identification problem presented herein). In such case, the complexity is much lower, only $\mathcal{C}_{\boldsymbol{\Phi}} = 3N$ operations are required for the scalar product update. The scalar products $\boldsymbol{\Phi}$ are computed by copying the upper left block, $\boldsymbol{\Phi}_{2:N,2:N} \leftarrow \boldsymbol{\Phi}_{1:N-1,1:N-1}$, and by updating the first row $\boldsymbol{\Phi}_{1,1:N} \leftarrow \lambda\boldsymbol{\Phi}_{1,1:N} + \alpha_1\boldsymbol{\alpha}^T$ (the matrix is symmetric, so the first column has the same values). Algorithms 2 and 3 and the update of $\boldsymbol{\Psi}$ require additionally about $2mN + 3N$ operations.

Despite the double bookkeeping of the residuals, the complexity of DCD-AMP is only slightly larger than that of a single CD-AMP of size M. The update of the scalar products between the columns of $\boldsymbol{A}$ in (12) is performed

Alg. 5. (Estimate coefficient i, search in the set $\mathcal{S}$, two residuals)
inputs: *i, $\mathcal{S}$*

1 find best candidate column searching all columns in $\mathcal{S}$
$k = \arg\max_{l\in\{i,\mathcal{S}\}} \Psi_l^2/\Phi_{l,l}$
2 if $k > i$ (another column is better than i)
2.1 if $k > M$ (the best column is inactive)
2.1.1 remove old coefficient i from the M-sparse solution associated with $\boldsymbol{\Omega}$
$\boldsymbol{\Omega}_{1:N} \leftarrow \boldsymbol{\Omega}_{1:N} + x_i\boldsymbol{\Phi}_{1:N,i}$, $x_i \leftarrow 0$
2.2 else (the best column is pending)
2.1.1 remove old coefficient k from the M-sparse solution associated with $\boldsymbol{\Omega}$
$\boldsymbol{\Omega}_{1:N} \leftarrow \boldsymbol{\Omega}_{1:N} + x_k\boldsymbol{\Phi}_{1:N,k}$, $x_k \leftarrow 0$
2.3 insert column k on position i; push columns $L + 1 : k - 1$ one position to the right
apply the order changes in $\boldsymbol{x}$, $\boldsymbol{\Phi}$, $\boldsymbol{\Psi}$, $\boldsymbol{\delta}$ and $\boldsymbol{\Omega}$
3 evaluate coefficient value $x_i = \Psi_i/\Phi_{i,i}$
4 update scalar product $\boldsymbol{\Omega}$
$\boldsymbol{\Omega}_{1:N} \leftarrow \boldsymbol{\Omega}_{1:N} - x_i\boldsymbol{\Phi}_{1:N,i}$

like in CD-AMP. The cost of the update of the scalar products between the columns of $\boldsymbol{A}$ and the residuals is indeed increased, but this is only $3N$. Using a judicious update of the residuals (Alg. 4) via the scalar product difference $\boldsymbol{\delta}$, the complexity of DCD-AMP is about $\mathcal{C}_{\boldsymbol{\Phi}} + 2MN + 10N$ operations, which is only about $10N$ more than for a CD-AMP of size M. The complexity introduced by the PLS criterion is negligible, only about $5M$. The simpler CD algorithm 1CD-AMP has about the same complexity as CD-AMP with $m = M$ (the PLS complexity being negligible) and thus comparable also with that of DCD-AMP. The MCD-AMP has a higher complexity, approximately $\mathcal{C}_{\boldsymbol{\Phi}} + M^2N$; the PLS criterion is this case in computed independently for each m-sparse solution with $m = 1 : M$, thus its added complexity is about $\frac{5}{2}M^2$. This translates to MCD-AMP being $\frac{M}{2}$ times more complex than DCD-AMP.

When compared with other algorithms, DCD-AMP has good complexity without sacrificing the performance (extensive simulations are presented in Section 5). In the GRLS algorithm [11], the coefficient computation is more complex, requiring approximately $\frac{5}{2}(N - M)^2 + O(MN)$ operations (in the worst case, with the parameter $\tau_0 = 2$, see relation (20) from [11]); furthermore, the term depending on MN generally involves more operations than the whole DCD-AMP for time shifted input data. If the input data structure can not be exploited, the update of $\boldsymbol{\Phi}$ raises the complexity of DCD-AMP such that, for low values of M and large N, the complexity of GRLS is generally more than $\frac{5}{3}$ times higher than that of DCD-AMP. The TNWL algorithm from [8] uses an efficient coordinate descent coupled with an inner RLS algorithm for finding the solution estimate. The computation of the RLS solution can be implemented to consider the time shifted input data and thus, in this case, the complexity of the TNWL is slightly higher than that of the DCD-AMP. For general input data without any particular exploitable structure, the TNWL is essentially more than three times more complex than the DCD-AMP (depending on the implementation of the RLS algorithm). The SPARLS algorithm [6] has a complexity similar to that of DCD-AMP (but produces much worse results).

5. Simulations

We validate the performance of our algorithms in the simulation setup from [11]. The sparse filter in the FIR channel identification problem (1) has length $N = 200$. Its $L_t = 5$ nonzero coefficients have a sinusoidal variation according to

$$h_i^{(t)} = a_i \cos(2\pi f T_s t + \alpha_i). \tag{16}$$

The nonzero coefficient positions i are randomly chosen; the amplitude a_i and the phase α_i are distributed uniformly in $[0.05, 1]$ and $[0, 2\pi]$. The variation speed is controlled by the product fT_s between the frequency f and the sampling interval T_s. We norm the coefficient vector so that its average squared norm over all t is 1. The input test data are generated from $\mathcal{N}(0, 1)$; the outputs, corresponding to the filtered inputs, are corrupted by a zero mean additive noise, normally distributed with $\sigma^2 = 0.01$. The datasets are exactly as in [11].

We assess the performance of the algorithms in terms of the mean square error of the coefficients

$$\text{MSE}^{(t)} = E\{||\boldsymbol{h}^{(t)} - \boldsymbol{x}^{(t)}||_2^2\}, \tag{17}$$

where $\boldsymbol{h}^{(t)}$ is the true coefficient vector and $\boldsymbol{x}^{(t)}$ its estimate. The value of $\text{MSE}^{(t)}$ is evaluated by averaging over 1000 runs at each time t, while the average MSE is defined as the mean $\text{MSE}^{(t)}$ over the last 100 values of t.

Our algorithms are CD-AMP, in the form presented in Alg. 3 with known sparsity level $m = L_t$, and DCD-AMP-F/V, the algorithm using the PLS criterion and the double residual optimization presented in Section 3. Since it requires a priori information about the sparsity level, CD-AMP is used only as a reference for the performance of DCD-AMP. To provide further insight into the performance we also include the two algorithms, 1CD-AMP-F/V and MCD-AMP-F/V, presented at the end of Section 3. The final letter means a fixed (F) or variable (V) upper bound M for all algorithms that use such a bound.

We compare them with the following algorithms: RLS-SP, the sparsity informed RLS algorithm knowing a priori the number of nonzero coefficients and their position, as reference for the best attainable performance; GRLS-PLS-F/V, the algorithms from [11], implementing an adaptive version of the greedy least squares algorithm; SPARLS,

Table 1: Average MSE for the studied algorithms, $L_t = 5$.

fT_s	0.005	0.002	0.001	0.0005	0.0002	0.0001
λ	0.86	0.90	0.92	0.94	0.96	0.98
RLS-SP	0.089	0.0267	0.0110	0.00518	0.00246	0.00306
SPARLS	0.824	0.4417	0.1578	0.04225	0.01120	0.00767
TNWL-OPT	0.498	0.1491	0.0311	0.01248	0.00471	0.00386
CD-AMP	0.401	0.0514	0.0177	0.00773	0.00340	0.00343
$M = 20$						
GRLS-PLS-F	0.213	0.0511	0.0189	0.00853	0.00356	0.00357
DCD-AMP-F	0.365	0.0556	0.0184	0.00806	0.00349	0.00358
MCD-AMP-F	0.304	0.0473	0.0168	0.00766	0.00342	0.00340
1CD-AMP-F	0.879	0.1484	0.0415	0.01501	0.00558	0.00545
$M = 40$						
GRLS-PLS-F	0.194	0.0500	0.0180	0.00824	0.00374	0.00373
DCD-AMP-F	0.373	0.0551	0.0185	0.00801	0.00351	0.00353
MCD-AMP-F	0.319	0.0478	0.0168	0.00770	0.00343	0.00340
1CD-AMP-F	1.671	0.2940	0.0614	0.02016	0.00722	0.00705
$\Delta = 5$						
GRLS-PLS-V	0.285	0.0569	0.0187	0.00762	0.00330	0.00340
DCD-AMP-V	0.339	0.0525	0.0179	0.00786	0.00342	0.00353
MCD-AMP-V	0.265	0.0456	0.0165	0.00754	0.00338	0.00339
1CD-AMP-V	0.519	0.0843	0.0266	0.01063	0.00431	0.00433
$\Delta = 10$						
GRLS-PLS-V	0.242	0.0570	0.0196	0.00845	0.00341	0.00346
DCD-AMP-V	0.362	0.0545	0.0183	0.00801	0.00349	0.00355
MCD-AMP-V	0.279	0.0464	0.0167	0.00762	0.00341	0.00340
1CD-AMP-V	0.689	0.1164	0.0343	0.01292	0.00501	0.00495

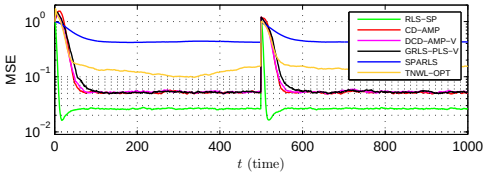


Figure 3: $\text{MSE}^{(t)}$ for $L_t = 5$, $fT_s = 0.002$ and $\lambda = 0.9$.

the algorithm presented in [6]; TNWL-OPT, the best of the algorithms presented in [8] using an optimized forgetting factor λ for the inner RLS loop. Their configuration parameters are set as explained in [11].

Table 1 contains the average MSE for the above algorithms for different variation speeds of the coefficients values. The tests are performed for variation speeds ranging from (the highest) $fT_s = 0.005$, corresponding to a period of 200 samples, to (the lowest) $fT_s = 0.0001$, corresponding to a period of 10000 samples. The forgetting factor λ was chosen smaller for higher variation speeds, to allow the coefficient tracking. We group the algorithms based on the value of the threshold M or parameter Δ to allow easy comparison between algorithms having the same configuration. We note that for $\Delta = 5$, DCD-AMP-V which has the best average MSE among the DCD-AMP versions and also the lowest complexity, performs almost as well as CD-AMP, which knows the number of nonzero coefficients; hence, PLS gives accurate information on the sparsity level. Also, DCD-AMP-V has similar performance with GRLS-PLS-V, despite its simpler optimization technique. The simpler algorithm, 1CD-AMP-F/V, performs poorly because it can not simultaneously provide consistent sparsity level estimates and good values for the coefficients, due to the use of a single residual. By using two residuals, DCD-AMP produces a good L-sparse solution estimate and, at the same time, allows the estimation of the PLS criterion for large enough sparsity. Opposed to the GRLS algorithm where, due to the orthogonalization technique, solutions for different sparsity levels can be computed, the CD-AMP gives good approximates only for the m-sparse solution. This motivates the use of the two residuals $\boldsymbol{r}_L$ and $\boldsymbol{r}_M$ in the DCD-AMP instead of the naive approach from 1CD-AMP. The more complex MCD-AMP-F/V algorithm, which should provide the best possible performance for this class of algorithms, does not show big performance improvements when compared to the DCD-AMP-F/V algorithms; still, because it has accurate solutions for all sparsity levels, MCD-AMP-F/V is consistently better when compared to all the other algorithms for low and medium variation speeds. Thus, the double residual optimization can be seen as a good compromise between performance and algorithm complexity.

The new introduced algorithms also outperform SPARLS and TNWL-OPT in all simulations; the simplest algorithm, 1CD-AMP, provides comparable results, while the others are consistently better. For fast time-varying environments with $fT_s = 0.005$, DCD-AMP-F/V shows a performance degradation due to the large differences in the coefficient values between consecutive time instances. The performance can be improved by running multiple times the coordinate descent part in steps 2 and 3 of Alg. 1, in the spirit of cyclic matching pursuit [16, 18]. For the lower variations speeds, the performance gap between the GRLS algorithm and the DCD-AMP becomes small, DCD-AMP even managing to outperform GRLS.

We present in Fig. 3 and 4 the $\text{MSE}^{(t)}$ of the studied algorithms as a function of time t, giving an example of the evolution of the algorithms for both fast and slow variation speeds fTs. The simulations are performed for a channel sparsity level $L_t = 5$; at time $t = 500$, three of the five nonzero coefficients randomly change position. The algorithms GRLS-PLS-V and DCD-AMP-V use the parameter $\Delta = 5$. We note that DCD-AMP-V has better convergence speed than all other previous algorithms and is only barely slower than CD-AMP. Both the SPARLS and TNWL algorithms perform worse in terms of both the convergence speed and the stationary MSE.

We do not include explicitly the results of the simulations with the different column ordering strategy (introduced at the end of Section 2). The performance is similar to that of the algorithms employing the neighbor permutations, the MSE is about $1-2\%$ worse and the covergence speed marginally slower.

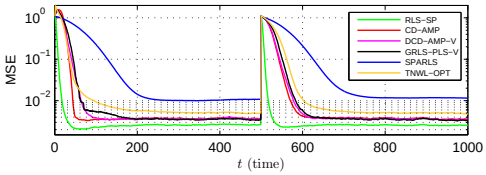


Figure 4: $\mathrm{MSE}^{(t)}$ for $L_t = 5$, $fT_s = 0.0002$ and $\lambda = 0.96$.

6. Conclusions

We have proposed an adaptive algorithm that, using a coordinate descent update strategy coupled with a greedy column selection derived from the adaptive matching pursuit algorithm, is able to compute online, sparse solutions to systems of linear equations. The complexity is low, while its performance is similar or better than that of other, more complex methods. There is a single, easy to choose, configuration parameter (M or Δ) and no precise a priori information is required, conferring robust behavior for unknown environments.

References

[1] A. M. Bruckstein, D. L. Donoho, M. Elad, From Sparse Solutions of Systems of Equations to Sparse Modeling of Signals and Images, SIAM Review 51 (1) (2009) 34–81.

[2] D. Giacobello, M. Christensen, M. Murthi, S. Jensen, M. Moonen, Sparse Linear Prediction and Its Applications to Speech Processing, IEEE Trans. Audio Speech Lang. Proc. 20 (5) (2012) 1644–1657.

[3] S. S. Chen, D. L. Donoho, M. A. Saunders, Atomic decomposition by basis pursuit, SIAM J. Sci. Comput. 20 (1998) 33–61.

[4] R. Tibshirani, Regression Shrinkage and Selection via the LASSO, J. Royal Stat. Soc., Ser. B 58 (1994) 267–288.

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

[6] B. Babadi, N. Kalouptsidis, V. Tarokh, SPARLS: The Sparse RLS Algorithm, IEEE Trans. Sign. Proc. 58 (8) (2010) 4013–4025.

[7] N. Kalouptsidis, G. Mileounis, B. Babadi, V. Tarokh, Adaptive Algorithms for Sparse System Identification, Sign. Proc. 91 (8) (2011) 1910–1919.

[8] D. Angelosante, J. Bazerque, G. Giannakis, Online Adaptive Estimation of Sparse Signals: Where RLS Meets the l_1-Norm, IEEE Trans. Sign. Proc. 58 (7) (2010) 3436–3447.

[9] Y. Kopsinis, K. Slavakis, S. Theodoridis, Online Sparse System Identification and Signal Reconstruction Using Projections Onto Weighted l_1 Balls, IEEE Trans. Sign. Proc. 59 (3) (2011) 936–952.

[10] S. Cotter, B. Rao, The Adaptive Matching Pursuit Algorithm for Estimation and Equalization of Sparse Time-Varying Channels, in: 34th Asilomar Conf. Sign. Syst. Comp., Vol. 2, 2000, pp. 1772–1776.

[11] B. Dumitrescu, A. Onose, P. Helin, I. Tăbuş, Greedy Sparse RLS, IEEE Trans. Sign. Proc. 60 (5) (2012) 2194–2207.

[12] Y. Chen, Y. Gu, A. Hero, Sparse LMS for system identification, in: ICASSP, Taipei, Taiwan, 2009, pp. 3125–3128.

[13] L. Vega, H. Rey, J. Benesty, S. Tressens, A Family of Robust Algorithms Exploiting Sparsity in Adaptive Filters, Audio, Speech, and Language Processing, IEEE Transactions on 17 (4) (2009) 572–581.

[14] S. Mallat, Z. Zhang, Matching Pursuit with Time Frequency Dictionaries, IEEE Trans. Sgn. Proc. 41 (12) (1993) 3397–3415.

[15] Y. Zakharov, G. White, J. Liu, Low-Complexity RLS Algorithms Using Dichotomous Coordinate Descent Iterations, IEEE Trans. Sign. Proc. 56 (7) (2008) 3150–3161.

[16] M. Christensen, S. Jensen, The Cyclic Matching Pursuit and its Application to Audio Modeling and Coding, in: 41th Asilomar Conf. Sign. Syst. Comp., 2007, pp. 550–554.

[17] B. L. Sturm, M. Christensen, Cyclic matching pursuit with multiscale time-frequency dictionaries, in: 44th Asilomar Conf. Sign. Syst. Comp., 2010, pp. 581–585.

[18] A. Onose, B. Dumitrescu, Cyclic Adaptive Matching Pursuit, in: ICASSP, Kyoto, Japan, 2012, pp. 3745–3748.

[19] J. Rissanen, Order Estimation by Accumulated Prediction Errors, J. Appl. Probab. 23 (1986) 55–61.

[20] G. Schwarz, Estimating the dimension of a model, J. Appl. Probab. 6 (2) (1978) 461–464.

[21] J. Rissanen, Modeling by shortest data description, Automatica 14 (5) (1978) 465–471.

Publication 2

[P2] A. Onose and B. Dumitrescu. Distributed coordinate descent using adaptive matching pursuit. In *Proceedings of the International Symposium on Intelligent Signal Processing and Communication Systems*, pages 513–518, Naha, Japan, November 2013.

Distributed Coordinate Descent Using Adaptive Matching Pursuit

Alexandru Onose*, Bogdan Dumitrescu*†
*Tampere University of Technology,
Department of Signal Processing,
FI-33101 Tampere, Finland
†'Politehnica' University of Bucharest,
Department of Automatic Control and Computers,
RO-060042 Bucharest, Romania
emails: alexandru.onose@tut.fi, bogdan.dumitrescu@tut.fi

***Abstract*—We propose a distributed adaptive algorithm for finding sparse solutions to systems of linear equations. The algorithm is greedy in nature. At each time moment, it first combines the current nonzero elements of the solution received from neighbor nodes by averaging them and then adapts the solution via a coordinate descent update using the local data. The column selection strategy, derived from adaptive matching pursuit, also fuses the received neighbor information with local data. The algorithm provides good performance with limited inter node communication and relatively low computational complexity.**

***Keywords*—coordinate descent, matching pursuit, sparse filters, distributed algorithm, greedy algorithm**

I. Introduction

The need for algorithms able to find sparse solutions is present in many areas of signal processing like channel identification, prediction, acoustics, image processing or classification [1]. Considerable effort has been directed into the development of batch methods where all the information is locally stored and processed [2], [3]. Since usually data are available sequentially, adaptive methods recently attracted increased interest resulting in the development of fast sparse algorithms [4], [5], [6].

In a distributed setup (e.g. a sensor network), where at the same time instant t new data are available locally at a number of nodes, a distributed algorithm operating over the network needs to produce good estimates (better than the ones possible with only the local node data). If we view the network as a graph with vertices between computing nodes that can communicate with each other, then the performance improvements are generally achieved by exchanging information between these connected neighbor nodes and fusing it with the local measurements. Traditional methods proposed for distributed scenarios are generally based on LMS [7], [8] or on RLS [9], [10]. When sparsity is required, distributed algorithms have been developed to work either in batch mode [11] or to estimate the solution in an adaptive fashion (e.g. the sparsity aware, projection based distributed algorithm from [12] or the sparse distributed LMS algorithm from [13]).

This work was supported by Tekes FiDiPro—Finland Distinguished Professor Programme and by GETA Finland.

In this paper we aim to develop a greedy adaptive distributed algorithm that is derived from [14], which in turn is related to the adaptive matching pursuit (AMP) [15]. The algorithm requires limited inter node communication and has a low complexity. To our knowledge this is the first greedy adaptive distributed algorithm for finding sparse solutions to linear systems.

We use the coordinate descent strategy proposed by [14] to compute in each node a local solution estimate (coefficient values, sparsity level and solution support) and we allow this data to be exchanged between neighbor nodes. Each node performs two tasks: first it combines the neighbor estimates with the local solution; then, performs an adaptation step to update the solution with newly available measurements.

The first task is achieved by averaging the neighbor solution coefficients with the local solution. This is a standard strategy and is employed in other distributed algorithms [12], [13]. To allow for all nodes to converge towards a common sparsity level, the neighbor sparsity level estimates influence the local sparsity level. The second task involves a coordinate descent on the direction given by the matching pursuit (MP) criterion [16] and is governed by a descent step size γ that allows for a tradeoff between the local adaptation speed and the convergence towards a common neighbor estimate. Additionally, to achieve fast convergence to a common solution, we account for the neighbor coefficient data when deciding the local support.

Let us begin by describing a typical finite impulse response (FIR) channel identification scenario at any existing node. At time instant t, the node has the local input and output data, $u(t)$ and $d(t)$. We need to estimate the values of the filter coefficients $x_j(t)$ that minimize the estimation error

$$e(t) = d(t) - \sum_{j=0}^{N-1} x_j(t)u(t-j) = \beta^{[t]} - \boldsymbol{\alpha}^{[t]T}\boldsymbol{x}^{[t]}, \quad (1)$$

where $\boldsymbol{\alpha}^{[t]} \in \mathbb{R}^N$ contains input data, $\beta^{[t]}$ is the output data and $\boldsymbol{x}^{[t]} \in \mathbb{R}^N$ is the vector of coefficients, all at current time t.

In a slow varying environment, when the solution changes slightly between consecutive samples, all past data from time

$\tau \leq t$ are exponentially windowed with a forgetting factor $\lambda \leq 1$ and produce the least-squares criterion

$$J^{[t]} = \left\| \boldsymbol{b}^{[t]} - \boldsymbol{A}^{[t]} \boldsymbol{x}^{[t]} \right\|^2. \quad (2)$$

The input matrix $\boldsymbol{A}^{[t]} \in \mathbb{R}^{t \times N}$ and the output vector $\boldsymbol{b}^{[t]} \in \mathbb{R}^t$ are constructed using $\boldsymbol{\alpha}^{[\tau]}$ and $\beta^{[\tau]}$ from each time instant $\tau \leq t$. The minimization of $J^{[t]}$ produces the solution estimate $\boldsymbol{x}^{[t]}$; we assume $\boldsymbol{x}^{[t]}$ to be sparse with $L^{[t]} \ll N$ nonzero coefficients. In a distributed environment each node k minimizes a local criterion $J^{[t,k]}$ but also exchanges some local information with other nodes over the network accelerating the convergence speed and improving the local estimate of an unique global solution. Henceforward we use the index $^{[t,k]}$ for data from time t available at a node k and the index $^{[t]}$ to refer to data from time t in the non distributed scenario.

The content of this paper is organized as follows: in Section II we present the general ideas of the non distributed version on the algorithm presented in [14]; Section III contains the proposed strategy for the distributed algorithm; Sections IV and V contain details regarding the algorithm complexity and the simulations used to validate the performance, respectively.

II. Adaptive matching pursuit with coordinate descent

We provide a brief description of the coordinate descent AMP (CD-AMP) [14] algorithm in the non distributed case, when only a single node exists. Lets suppose that at time $t-1$ the algorithm has an L-sparse solution $\boldsymbol{x}^{[t-1]}$; we also assume that the input data matrix $\boldsymbol{A}^{[t-1]}$ is permuted such that, ideally the nonzero coefficients are on the first L positions ($L = L^{[t]}$ is fixed for now for any t and the positions corresponding to nonzero elements are named active).

The solution $\boldsymbol{x}^{[t-1]}$ produces a residual

$$\boldsymbol{r}^{[t-1]} = \boldsymbol{b}^{[t-1]} - \sum_{i=1}^{L} x_i^{[t-1]} \boldsymbol{a}_i^{[t-1]}; \quad (3)$$

the i-th column of $\boldsymbol{A}^{[t-1]}$ is denoted by $\boldsymbol{a}_i^{[t-1]}$.

After a new input vector $\boldsymbol{\alpha}^{[t]}$ and the corresponding output $\beta^{[t]}$ are received at time t, the input matrix and the output vector become

$$\boldsymbol{A}^{[t]} = \begin{bmatrix} \sqrt{\lambda}\boldsymbol{A}^{[t-1]} \\ \boldsymbol{\alpha}^{[t]T} \end{bmatrix}, \quad \boldsymbol{b}^{[t]} = \begin{bmatrix} \sqrt{\lambda}\boldsymbol{b}^{[t-1]} \\ \beta^{[t]} \end{bmatrix}, \quad (4)$$

and the residual $\boldsymbol{r}^{[t]}$ has initially the form

$$\boldsymbol{r}^{[t]} = \begin{bmatrix} \boldsymbol{r}^{[t-1]} \\ \beta^{[t]} - \sum_{i=1}^{L} x_i^{[t-1]} \alpha_i^{[t]} \end{bmatrix}. \quad (5)$$

The solution $\boldsymbol{x}^{[t-1]}$ is then updated to take advantage of the newly acquired data. We perform this in two steps: i) we manage the support, i.e. the order of the active positions; ii) we update the coefficient values to incorporate the new data.

Step i) identifies the nonzero coefficients and orders them using a permutation scheme [14], [17]. For each active position $i = 1 : L-1$ we decide the best candidate column $\boldsymbol{a}_k^{[t]}$, with $k \in \mathcal{S} = \{i, i+1\}$, based on the alignment of the partial residual

$$\tilde{\boldsymbol{r}}^{[t]} = \boldsymbol{r}^{[t]} + x_i^{[t-1]} \boldsymbol{a}_i^{[t]} + x_{i+1}^{[t-1]} \boldsymbol{a}_{i+1}^{[t]}, \quad (6)$$

with the two neighbor columns $\boldsymbol{a}_i^{[t]}$ and $\boldsymbol{a}_{i+1}^{[t]}$ (this is similar to the MP column selection),

$$k = \arg\max_{l \in \mathcal{S}} \frac{|\boldsymbol{a}_l^{[t]T} \tilde{\boldsymbol{r}}^{[t]}|^2}{\|\boldsymbol{a}_l^{[t]}\|^2}. \quad (7)$$

If column $\boldsymbol{a}_{i+1}^{[t]}$ is found better than $\boldsymbol{a}_i^{[t]}$, we permute them in the matrix $\boldsymbol{A}^{[t]}$ (the corresponding coefficients $x_i^{[t-1]}$ and $x_{i+1}^{[t-1]}$ are also permuted).

Step ii) represents the optimization of the coefficient i, selected by the permutation associated with (7), via a coordinate descent. The residual is projected on column $\boldsymbol{a}_i^{[t]}$ to produce the update step

$$\mu = \frac{\boldsymbol{a}_i^{[t]T} \boldsymbol{r}^{[t]}}{\|\boldsymbol{a}_i^{[t]}\|^2}, \quad (8)$$

that would minimize $J^{[t]}$ in the context given by the other coefficients $x_j^{[t]}$, $j < i$ and $x_j^{[t-1]}$, $j > i$,

$$x_i^{[t]} = x_i^{[t-1]} + \mu = \frac{\boldsymbol{a}_i^{[t]T} (\boldsymbol{r}^{[t]} + x_i^{[t-1]} \boldsymbol{a}_i^{[t]})}{\|\boldsymbol{a}_i^{[t]}\|^2}. \quad (9)$$

Since we change the coefficient value, the residual needs to be updated also,

$$\boldsymbol{r}^{[t]} \leftarrow \boldsymbol{r}^{[t]} - \mu \boldsymbol{a}_i^{[t]}. \quad (10)$$

For the last position L we perform step i), the search for a candidate column $\boldsymbol{a}_k^{[t]}$, in the index set $\mathcal{S} = L : N$. The partial residual is now $\tilde{\boldsymbol{r}}^{[t]} = \boldsymbol{r}^{[t]} + x_L^{[t-1]} \boldsymbol{a}_L^{[t]}$, since $x_j^{[t-1]} = 0$, $j > L$. If a new column is found better by (7), we permute in $\boldsymbol{A}^{[t]}$ columns L and k (implicitly we also permute $\boldsymbol{x}^{[t]}$). The coefficient update for step ii) is then given by

$$x_L^{[t]} = \frac{\boldsymbol{a}_L^{[t]T} \tilde{\boldsymbol{r}}^{[t]}}{\|\boldsymbol{a}_L^{[t]}\|^2}. \quad (11)$$

The associated updated residual is

$$\boldsymbol{r}^{[t]} \leftarrow \tilde{\boldsymbol{r}}^{[t]} - x_L^{[t]} \boldsymbol{a}_L^{[t]}. \quad (12)$$

A summary of the CD-AMP algorithm is presented in Alg. 1. Before introducing the distributed algorithm we give the main ideas on how CD-AMP is modified to work with online sparsity estimation (L is allowed to change). In [14], the predictive least squares (PLS) [18] is used to estimate the sparsity level L but the Bayesian information criterion (BIC) [19] may serve the same purpose.

The fully adaptive algorithm that estimates online the sparsity, named Double CD-AMP (DCD-AMP) [14], minimizes two residuals $\boldsymbol{r}^{[t]}$ and $\boldsymbol{g}^{[t]}$, corresponding to an L-sparse and to an M-sparse (with $M > L$) solution, respectively. The M-sparse solution is not very accurate, but allows the computation of the PLS/BIC criteria to give good estimates L of the true number of nonzero coefficients. The L-sparse solution is then computed with much better accuracy, since it often has the correct support.

The two solutions overlap such that the L-sparse solution is included in the M-sparse one. Essentially, the CD-AMP algorithm is used for the computation of both solutions. There

Alg. 1. ***CD-AMP***

input: $\boldsymbol{r}^{[t-1]}$; $\boldsymbol{x}^{[t-1]}$; $\beta^{[t]}$; $\boldsymbol{\alpha}^{[t]}$; L; N;
output: $\boldsymbol{r}^{[t]}$; $\boldsymbol{x}^{[t]}$;

1 *Update the residual with the new available data like in (5).*
2 *For* $i = 1 : L-1$
  2.1 *Find the best candidate column for position* i *using (7).*
  2.2 *Update the associated coefficient like in (9) and the residual* $\boldsymbol{r}^{[t]}$ *like in (10).*
3 *For last position* $i = L$
  3.1 *Find the best candidate column using (7) but searching over the set* $\mathcal{S} = \{L : N\}$.
  3.2 *Update the associated coefficient like in (11) and the residual* $\boldsymbol{r}^{[t]}$ *like in (12).*

are however important differences. The candidates for position L are now only L and its neighbor $L+1$. In turn, the competition for position $L+1$ is open to all remaining columns, i.e. $\mathcal{S} = L+1 : N$ in (7). Moreover, the selection and the update of the coefficient on position $L+1$ are based on the residual $\boldsymbol{r}^{[t]}$. All these operations are meant to not only quickly bring into position $L+1$ the most likely nonzero coefficients that are not yet included in the solution, but to estimate them accurately even before they enter the solution.

We consider M fixed during the presentation, for simplicity. However, a change in the value of M can be cheaply implemented by a proper update of the residuals. Note that the algorithms are efficiently implemented using only the scalar products between the input data and the residual $\boldsymbol{A}^{[t]T}\boldsymbol{r}^{[t]}$, $\boldsymbol{A}^{[t]T}\boldsymbol{g}^{[t]}$ and between the input data matrix $\boldsymbol{A}^{[t]T}\boldsymbol{A}^{[t]}$. The complexity is presented in section IV for the scalar product implementation.

III. Distributed algorithm

We study the channel identification problem on a network $\mathcal{N}$ consisting of P nodes. We assume it to be static (it has a fixed topology $\mathcal{T}$) and connected (there exists a network path between any two nodes $n^{[k]}$ and $n^{[l]}$ in $\mathcal{N}$). The true coefficient vector $\boldsymbol{x}^{[t]}$ is the same over the network, $\boldsymbol{x}^{[t,k]} = \boldsymbol{x}^{[t,l]}$, $\forall n^{[k]}, n^{[l]} \in \mathcal{N}$. The goal is to jointly improve the local node estimates of $\boldsymbol{x}^{[t]}$ by using, besides the locally available data, also information received from neighbors.

We present in what follows the extension of DCD-AMP to the distributed case; we explain the main ideas and changes from the non distributed algorithm and how they affect a generic node $n^{[k]}$ at time t. The following notations for the data belonging to the k-th node are used: $\boldsymbol{x}^{[t,k]}$ is the solution computed locally at each time instant t; it contains $L^{[t,k]}$ coefficients associated with the local active positions (thus named active solution) but also the other coefficients computed to allow the sparsity estimation (in particular the coefficient $L^{[t,k]}+1$); $\ell^{[t,k]}$ represents the sparsity level estimate computed locally using the information theoretic criteria. The number $L^{[t,k]}$ of active positions is computed using also neighbor data and hence, $L^{[t,k]} \neq \ell^{[t,k]}$ in general.

At each time instant t, a node $n^{[k]}$ receives two sets of data:

- *The local data:* The input data $\beta^{[t,k]}$ and output data $\boldsymbol{\alpha}^{[t,k]}$, like (1) but different for each node.
- *The neighbor nodes data:* Since the bandwidth is limited, the node $n^{[k]}$ only receives from each neighbor node $n^{[j]}$, $j \in \mathcal{N}_k$ ($\mathcal{N}_k$ is the set of all nodes neighbor with $n^{[k]}$), data corresponding to the solution from time $t-1$:
  i) The sparsity level estimate $\ell^{[t-1,j]}$ computed using only the local data.
  ii) The $L^{[t-1,j]}$ coefficients of the current active solution $\boldsymbol{x}^{[t-1,j]}$.
  iii) The coefficient $L^{[t-1,j]}+1$, the best of the other coefficients not included in the current active solution.
  iv) The absolute coefficient positions.

The coefficient $L^{[t-1,j]}+1$ is transmitted to cover a possible increase of the sparsity estimate at the next time instance and also to allow better averaging in case it is actually in the solution support in other nodes; this produces a faster convergence. All other coefficient values (on positions $L^{[t-1,j]}+2 : N$) of $\boldsymbol{x}^{[t-1,j]}$ do not contribute directly to the solution and are implicitly considered 0. After receiving the new data we minimize a criterion $J^{[t,k]}$ like in (2) based on the local data $\boldsymbol{A}^{[t,k]}$ and $\boldsymbol{b}^{[t,k]}$ but we also use the neighbors data to improve the solution and to force the estimates to converge towards a common value, as explained below.

To provide a reliable number $L^{[t,k]}$ of active positions we average the estimated sparsity level of the current node with the one received from its neighbors. To prevent large fluctuations, we allow only changes by at most one:

$$L^{[t,k]} = L^{[t-1,k]} + \operatorname{sign}\left(\left\lceil \frac{\ell^{[t,k]} + \sum_{j\in\mathcal{N}_k} \ell^{[t-1,j]}}{|\mathcal{N}_k|+1} \right\rceil - L^{[t-1,k]}\right). \quad (13)$$

Any change of the values of $L^{[t,k]}$ when compared to $L^{[t-1,k]}$ produces an update of both local residuals, $\boldsymbol{r}^{[t,k]}$ and $\boldsymbol{g}^{[t,k]}$.

We compute the average coefficient estimate,

$$\bar{\boldsymbol{x}}^{[t-1,k]} = \frac{\boldsymbol{x}^{[t-1,k]} + \sum_{j\in\mathcal{N}_k} \boldsymbol{x}^{[t-1,j]}}{|\mathcal{N}_k|+1}. \quad (14)$$

For positions where the support differs we average with the implicit coefficient 0. Since we will use the average coefficient values, the residual update (5) changes,

$$\boldsymbol{r}^{[t,k]} = \begin{bmatrix} \boldsymbol{r}^{[t-1,k]} + \sum_{i=1}^{L^{[t,k]}} \delta_i \boldsymbol{a}_i^{[t-1,k]} \\ \beta^{[t]} - \sum_{i=1}^{L^{[t,k]}} \bar{x}_i^{[t-1,k]} \alpha_i^{[t]} \end{bmatrix}, \quad (15)$$

where $\delta_i = x_i^{[t-1,k]} - \bar{x}_i^{[t-1,k]}$. Thus we replace the local coefficient values existing on the local support by averaging them with the neighbor data. This data, $\bar{\boldsymbol{x}}^{[t-1,k]}$, represents the new solution from time $t-1$ that has to be updated to include new available local measurements similarly to the update of $\boldsymbol{x}^{[t-1]}$ in the non distributed scenario. Thus, to keep the same notations we replace $\boldsymbol{x}^{[t-1,k]} \leftarrow \bar{\boldsymbol{x}}^{[t-1,k]}$ and we update $\boldsymbol{x}^{[t-1,k]}$ as presented below.

The coefficient selection rule (7) is changed to use the neighbor coefficients and thus to improve the accuracy of finding the true support. A coefficient, received from one of the neighbor nodes $n^{[j]}$, has associated a partial residual $\tilde{\boldsymbol{r}}^{[t,j]}$ based on the node's local data. A selection criterion based on each $\tilde{\boldsymbol{r}}^{[t,j]}$ can be written as

$$k = \arg\max_{l \in \mathcal{S}} \frac{|\boldsymbol{a}_l^{[t,k]T} \tilde{\boldsymbol{r}}^{[t,k]}|^2}{\|\boldsymbol{a}_l^{[t,k]}\|^2} + \sum_{j \in \mathcal{N}_k} \frac{|\boldsymbol{a}_l^{[t,j]T} \tilde{\boldsymbol{r}}^{[t,j]}|^2}{\|\boldsymbol{a}_l^{[t,j]}\|^2}. \quad (16)$$

The neighbor residuals $\tilde{\boldsymbol{r}}^{[t,j]}$ are not available locally and we substitute them with the available coefficient data. Since the underlying process generating the input data matrix is the same for all nodes, with $\|\boldsymbol{a}^{[t,k]}\|^2 \approx \|\boldsymbol{a}^{[t,j]}\|^2$, we approximate (see (9) and (11))

$$\frac{|\boldsymbol{a}^{[t,j]T} \tilde{\boldsymbol{r}}^{[t,j]}|^2}{\|\boldsymbol{a}^{[t,j]}\|^2} \approx \left(\boldsymbol{x}_l^{[t-1,j]}\right)^2 \|\boldsymbol{a}_l^{[t,k]}\|^2, \quad (17)$$

and incorporate the neighbor residual data into the local selection criterion,

$$k = \arg\max_{l \in \mathcal{S}} \frac{|\boldsymbol{a}_l^{[t,k]T} \tilde{\boldsymbol{r}}^{[t,k]}|^2}{\|\boldsymbol{a}_l^{[t,k]}\|^2} + \sum_{j \in \mathcal{N}_k} \left(\boldsymbol{x}_l^{[t-1,j]}\right)^2 \|\boldsymbol{a}_l^{[t,k]}\|^2. \quad (18)$$

We also change the update rule (9) to

$$x_i^{[t,k]} = x_i^{[t-1,k]} + \gamma\mu, \quad (19)$$

creating a slower coordinate descent governed by the descent step $\gamma \in (0, 1]$. A smaller value of γ means a lower influence of the local data. This allows the algorithm to remain close to the mean coefficient values computed from neighbor data but also allows the adaptation to take advantage of new local data. The residual update is modified in a similar fashion,

$$\boldsymbol{r}^{[t,k]} = \boldsymbol{r}^{[t,k]} - \gamma\mu \boldsymbol{a}_i^{[t,k]}. \quad (20)$$

For the additional coefficient $L^{[t,k]} + 1$ we use the same averaging strategy (14) and the updated residual $\boldsymbol{r}^{[t,k]}$; the selection criterion (18) is used for all column searches to introduce a preference for positions existing in neighbor nodes. For the other positions $L^{[t,k]} + 2 : M$, since there is a low probability that the columns are in the active set of the neighbor nodes, we use the full length coordinate descent (with $\gamma = 1$) to compute their associated coefficients. This achieves the fastest convergence given the local data. An overview of the full distributed algorithm presenting the main steps is available in Alg. 2.

Alg. 2. ***Distributed DCD-AMP***

local input: $\boldsymbol{r}^{[t-1,k]}$; $\boldsymbol{x}^{[t-1,k]}$; $\beta^{[t,k]}$; $\boldsymbol{\alpha}^{[t,k]}$; $L^{[t-1,k]}$; M; N;
neighbor input: $\boldsymbol{x}^{[t-1,j]}$; $\ell^{[t-1,j]}$; $j \in \mathcal{N}_k$
output: $\boldsymbol{r}^{[t,k]}$; $\boldsymbol{x}^{[t,k]}$; $\ell^{[t,k]}$; $L^{[t,k]}$

1 *Compute the local $L^{[t,k]}$ from (13) considering the data received from the neighbors.*
2 *Average the coefficients like in (14).*
3 *Update the residual $\boldsymbol{r}^{[t-1,k]}$ to include the averaged coefficients and the new available data like in (15) (a similar operation is performed for $\boldsymbol{g}^{[t-1,k]}$).*
4 *For $i = 1 : L^{[t,k]}$*
  4.1 *Find the best candidate column for position i using (18) and searching over the set $\mathcal{S} = \{i : i+1\}$.*
  4.2 *Update the associated coefficient like in (19) and the residual $\boldsymbol{r}^{[t,k]}$ like in (20).*
5 *For position $i = L^{[t,k]} + 1$*
  5.1 *Find the best candidate column using (18) searching over the set $\mathcal{S} = \{L^{[t,k]} + 1 : N\}$. Moving it on position $L^{[t,k]} + 1$ pushes the old columns one position to the right.*
  5.2 *Compute coefficient $L^{[t,k]} + 1$ using $\boldsymbol{r}^{[t,k]}$ but modify the residual $\boldsymbol{g}^{[t,k]}$ since the coefficient is not in the active set.*
6 *For $i = L^{[t,k]} + 2 : M$*
  6.1 *Find the best candidate column for position i using (18) searching over the set $\mathcal{S} = \{i : i+1\}$. For $i = M$ this step is not performed.*
  6.2 *Update the associated coefficient like in (9) but based on the residual $\boldsymbol{g}^{[t,k]}$.*
  6.3 *Update the residual $\boldsymbol{g}^{[t,k]}$ to account for the new coefficient value.*
7 *Estimate the new sparsity level $\ell^{[t,k]}$ using the local data.*

IV. Complexity and communication

The complexity of the DCD-AMP algorithm is reported in [14] to be about $\frac{3}{2}N^2 + 2MN + 10N$. The complexity of the distributed algorithm is slightly higher due to the residual update from (15). This accounts for about $2LN$ operations per time instant with L being the current sparsity estimate, $L < M$. The other operations are cheaply performed. Thus the overall complexity grows to about $\frac{3}{2}N^2 + 4MN + 10N$.

The algorithm requires limited communication between nodes since it transmits almost only the nonzero coefficients and their positions. Each node transmits $2(L + 1) + 1$ values.

V. Simulations

We test the performance of the introduced distributed DCD-AMP algorithm for a network of 10 sensors. We use three topologies; $\mathcal{T}_R$, a ring configuration, $\mathcal{T}_F$, a full broadcast configuration where each node is connected to all the others and $\mathcal{T}_I$, which has no communication paths between nodes, each node using only its local data.

We test the algorithms for a time varying FIR channel identification task like in (1). The filter length $N = 200$ and the filter has 10 true nonzero coefficients randomly distributed. Each nonzero coefficient i changes in time according to a sinusoidal law

$$h_i = \alpha_i cos(2\pi f T_s + \eta_i), \quad (21)$$

with the amplitude α_i and the phase η_i distributed uniformly in $[0.05, 1]$ and $[0, 2\pi]$. The simulations contain a sudden change of the position of half of the coefficients to allow the evaluation of the convergence speed.

The coefficient vector is normed such that its average squared norm is 1. The input data is generated at each node according to a zero mean Gaussian distribution with variance 1. The outputs are affected by white noise with $\sigma^2 = 0.01$. The performance is given by the coefficient mean squared error,

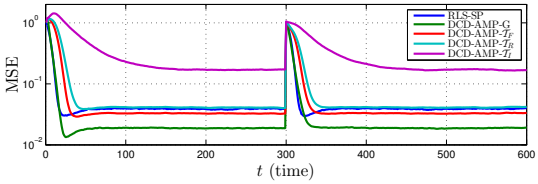


Fig. 1. Average MSE for $fT_s = 0.002$; $\lambda = 0.9$.

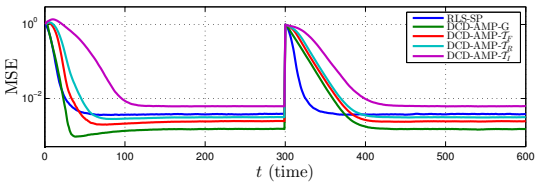


Fig. 2. Average MSE for $fT_s = 0.0002$; $\lambda = 0.96$.

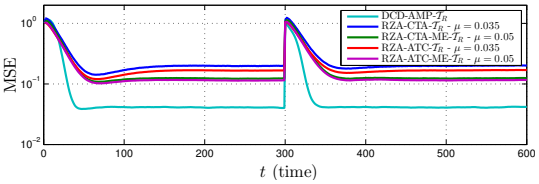


Fig. 3. Comparison with the RZA algorithms for $fT_s = 0.002$; $\lambda = 0.9$.

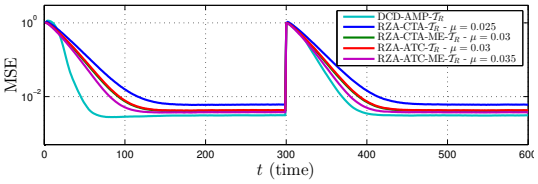


Fig. 4. Comparison with the RZA algorithms for $fT_s = 0.0002$; $\lambda = 0.96$.

$\text{MSE} = E\{||\boldsymbol{h} - \boldsymbol{x}||^2\}$. We estimate its value by averaging over 1000 tests and over all nodes in the network.

For comparison we include the sparsity informed RLS algorithm (RLS-SP) running on one node and the global DCD-AMP algorithm from [14] that has all the data from across the network available at each time instance (DCD-AMP-G). We present the performance for two variation speeds, $fT_s = 0.002$ and $fT_s = 0.0002$ in Fig. 1 and 2, respectively. The distributed algorithms DCD-AMP-$\mathcal{T}_R$ and DCD-AMP-$\mathcal{T}_F$ use $\gamma = 0.4$ while DCD-AMP-$\mathcal{T}_I$ has $\gamma = 1$. In all experiments M is adapted such that $M = L + 5$, where L is the current sparsity estimate.

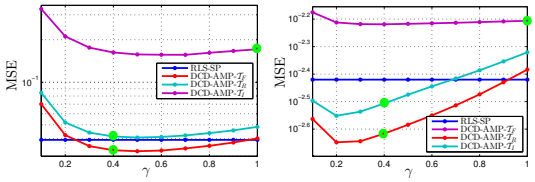
For the ring network configuration, $\mathcal{T}_R$, we compare our algorithms with the RZA-ATC and RZA-CTA algorithms from [13] in Fig. 3 and 4. In [13] it was concluded that the RZA-ATC algorithms have similar performance with the projection based algorithms from [12] and to our knowledge these are the best performing sparsity aware adaptive distributed algorithms. A direct comparison is not immediate since the algorithms from [13] are LMS based while ours minimizes the RLS criterion. For this purpose we searched for the best adaptation step μ_{RZA} on a grid with step of 0.005. The parameter γ_{RZA} was estimated online as specified in [13] and we chose $\epsilon_{\text{RZA}} = 0.001$. The combination strategy averages the neighbor estimates (our algorithm uses the same strategy). Additionally, the performance of the RZA-ATC and RZA-CTA algorithms can be improved by exchanging local measurements over the network. The algorithms using this strategy are named RZA-ATC-ME and RZA-CTA-ME and assign the same weight, $\frac{1}{|\mathcal{N}_k|+1}$, for the measurements received at each node k.

In Fig. 5 we present the performance of the algorithms as a function of coordinate descent adaptation step γ. It has marked with green dots the simulation conditions for the algorithms presented in Fig. 1 and 2.

We also present in Fig 6 a simulation setup where the coefficients remain constant in time ($fT_s = 0$). We only include the best of the algorithms from [13], namely RZA-ATC-ME, with the parameter $\mu_{\text{RZA}} = 0.015$ chosen such that it has approximately the same stationary error like DCD-AMP-$\mathcal{T}_R$ with $\gamma = 0.4$.

The performance of the distributed DCD-AMP algorithms is dependent on the network topology but remains robust. The two used topologies are a showcase of the best and worst performance, respectively. Thus, although a loosely connected network suffers performance degradation the performance will be at worst the equal to that of DCD-AMP-$\mathcal{T}_R$. The parameter γ is a tradeoff between adaptation speed and the steady state error, the larger the value the more the local data is emphasized. For small values of γ more of the influence received from neighbor nodes is retained. In Fig. 6 it can be seen that for very small values of γ the algorithm approaches the solution given by DCD-AMP-G which has all data available locally.

When compared with the algorithms from [13] our algorithm always achieves better performance. The RZA-ATC and RZA-CTA algorithms are unable to properly track the variable FIR channels (Fig. 3 and 4) due to the low convergence speed achieved by the methods (which can be assessed in Fig. 6). It should be noted that while we explicitly searched for the near optimal parameter μ_{RZA}, the parameter γ for the distributed DCD-AMP is not optimal as can be seen from Fig. 5. Moreover, the forgetting factor λ is chosen in the spirit of [14], [17] such that it is lower for faster coefficient variations but no search for the optimal parameter is performed. While our algorithm is more complex, in order to achieve good performance RZA-ATC-ME requires the exchange of more local data and two communication sessions per time moment, which can be prohibitive. The distributed DCD-AMP algorithm only requires limited communication between neighbor nodes. If the input data is time shifted like in the FIR channel identification problem such that the communication of the neighbor measurements is cheaper, the complexity of the distributed DCD-AMP also becomes of the same order with that of the algorithms from [13].



a) $fT_s = 0.002$, $\lambda = 0.9$. a) $fT_s = 0.0002$, $\lambda = 0.96$.

Fig. 5. Average MSE as a function of γ

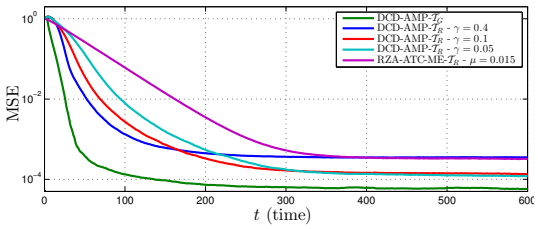


Fig. 6. Average MSE for constant channel and $\lambda = 0.99$.

VI. Conclusions

We have presented a distributed sparse adaptive algorithm that uses a coordinate descent strategy on local data coupled with the averaging of neighbor node information to provide good performance. It requires little communication in the network and it is robust with regard to its topology. The complexity is low being comparable to that of the base algorithm DCD-AMP. The algorithm requires little prior information; the configuration parameter γ is easy to choose as a tradeoff between convergence speed and stationary error.

References

[1] A.M. Bruckstein, D.L. Donoho, and M. Elad, "From Sparse Solutions of Systems of Equations to Sparse Modeling of Signals and Images," *SIAM Rev.*, vol. 51, no. 1, pp. 34–81, 2009.

[2] S. S. Chen, D. L. Donoho, and M. A. Saunders, "Atomic decomposition by basis pursuit," *SIAM J. Sci. Comput.*, vol. 20, pp. 33–61, 1998.

[3] R. Tibshirani, "Regression Shrinkage and Selection via the LASSO," *J. Royal Stat. Soc., Ser. B*, vol. 58, pp. 267–288, 1994.

[4] B. Babadi, N. Kalouptsidis, and V. Tarokh, "SPARLS: The Sparse RLS Algorithm," *IEEE Trans. Sign. Proc.*, vol. 58, no. 8, 2010.

[5] D. Angelosante, J.A. Bazerque, and G.B. Giannakis, "Online Adaptive Estimation of Sparse Signals: Where RLS Meets the l_1-Norm," *IEEE Trans. Sign. Proc.*, vol. 58, no. 7, pp. 3436–3447, Jul. 2010.

[6] Y. Kopsinis, K. Slavakis, and S. Theodoridis, "Online Sparse System Identification and Signal Reconstruction Using Projections Onto Weighted l_1 Balls," *IEEE Trans. Sign. Proc.*, vol. 59, no. 3, pp. 936–952, Mar. 2011.

[7] I.D. Schizas, G. Mateos, and G.B. Giannakis, "Distributed LMS for Consensus-Based In-Network Adaptive Processing," *IEEE Trans. Sign. Proc.*, vol. 57, no. 6, pp. 2365–2382, June.

[8] F.S. Cattivelli and A.H. Sayed, "Diffusion lms strategies for distributed estimation," *IEEE Tran. Sign. Proc.*, vol. 58, no. 3, pp. 1035–1048, 2010.

[9] G. Mateos, I.D. Schizas, and G.B. Giannakis, "Distributed Recursive Least-Squares for Consensus-Based In-Network Adaptive Estimation," *IEEE Trans. Sign. Proc.*, vol. 57, no. 11, pp. 4583–4588, 2009.

[10] F.S. Cattivelli, C.G. Lopes, and A.H. Sayed, "Diffusion recursive least-squares for distributed estimation over adaptive networks," *IEEE Trans. Sign. Proc.*, vol. 56, no. 5, pp. 1865–1877, 2008.

[11] J.F.C. Mota, J. Xavier, P.M.Q. Aguiar, and M. Puschel, "Distributed Basis Pursuit," *IEEE Trans. Sign. Proc.*, vol. 60, no. 4, pp. 1942–1956, 2012.

[12] S. Chouvardas, K. Slavakis, Y. Kopsinis, and S. Theodoridis, "A Sparsity Promoting Adaptive Algorithm for Distributed Learning," *IEEE Trans. Sign. Proc.*, vol. 60, no. 10, pp. 5412–5425, 2012.

[13] P. Di Lorenzo and A.H. Sayed, "Sparse distributed learning based on diffusion adaptation," *IEEE Tran. Sign. Proc.*, vol. 61, no. 6, pp. 1419–1433, 2013.

[14] A. Onose and B. Dumitrescu, "Adaptive Matching Pursuit Using Coordinate Descent and Double Residual Minimization," *Sig. Proc.*, vol. 93, no. 11, pp. 3143–3150, 2013.

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

[16] S.G. Mallat and Z. Zhang, "Matching Pursuit with Time Frequency Dictionaries," *IEEE Trans. Sgn. Proc.*, vol. 41, no. 12, pp. 3397–3415, 1993.

[17] B. Dumitrescu, A. Onose, P. Helin, and I. Tăbuş, "Greedy Sparse RLS," *IEEE Trans. Sign. Proc.*, vol. 60, no. 5, pp. 2194–2207, 2012.

[18] J. Rissanen, "Order Estimation by Accumulated Prediction Errors," *J. Appl. Probab.*, vol. 23, pp. 55–61, 1986.

[19] G. Schwarz, "Estimating the dimension of a model," *Ann. Stat.*, vol. 6, no. 2, pp. 461–464, 1978.

Publication 3

[P3] A. Onose and B. Dumitrescu. Cyclic adaptive matching pursuit. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 3745–3748, Kyoto, Japan, March 2012.

CYCLIC ADAPTIVE MATCHING PURSUIT

Alexandru Onose, Bogdan Dumitrescu

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

ABSTRACT

We present an improved Adaptive Matching Pursuit algorithm for computing approximate sparse solutions for overdetermined systems of equations. The algorithms use a greedy approach, based on a neighbor permutation, to select the ordered support positions followed by a cyclical optimization of the selected coefficients. The sparsity level of the solution is estimated on-line using Information Theoretic Criteria. The performance of the algorithm approaches that of the sparsity informed RLS, while the complexity remains lower than that of competing methods.

Index Terms— matching pursuit, adaptive algorithm, sparse filters, channel identification

1. INTRODUCTION

In recent years, sparse approximation problems have been of a major interest due to their practical applicability in array processing, compression, denoising and many other tasks. The aim of this paper is to present a series of improvements to an adaptive version of Matching Pursuit (MP) [1] making it a viable alternative to more complex methods like [2, 3].

Let us consider a typical example of an FIR channel identification task. At time t the channel input $u(t)$ and output $d(t)$ are measured and our aim is to find the coefficients h_i, $i = 0 : N$, such that the estimation error

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

is minimized. In a slow time varying environment, with λ as the forgetting factor, this can be translated into minimizing

$$J(t) = \sum_{i=1}^{t} \lambda^{t-i} |e(i)|^2. \tag{2}$$

This is equivalent to minimizing, at each time instance t, the norm of the residual $||\mathbf{b} - \mathbf{A}\mathbf{x}||_2$, where the matrix $\mathbf{A} \in \mathbb{R}^{t \times N}$ and the vector $\mathbf{b} \in \mathbb{R}^t$ are built with the input and output data, respectively. Furthermore, we consider that the coefficient vector has at most $M \ll N$ non-zero coefficients ($\mathbf{x}$ is sparse), fact usually valid in many practical applications. The number of significant coefficients and their positions are not known a priori.

For finding sparse approximate solution to the minimization problem presented in (2) we use an Adaptive Matching Pursuit (AMP) [4] algorithm combined with a Cyclic Matching Pursuit approach [5]. The algorithm provides models with different sparsity levels and we apply Information Theoretic Criteria (ITC) in a similar way as in [6] to choose the model that best fits the data. We present two algorithms, each assuming either a fixed or variable upper sparsity level bound M, and we show how the ITC can be computed; we prove, by empirical simulations, that the performance of these low-complexity algorithms is good.

This paper is organized as follows: in section 2 we present an improved adaptive matching pursuit algorithm; section 3 presents details regarding the ITC that are used to selecting the best support size while section 4 details their use in conjunction with our algorithms; section 5 contains the results of our simulations.

2. CYCLIC ADAPTIVE MATCHING PURSUIT

We begin by presenting an extension to the classic MP algorithm adjusted for an adaptive context. At the base of our method resides an improved AMP that uses a cyclic coefficient re-computation [5] to minimize the prediction error.

Like in the MP case, the algorithm selects one by one columns from the matrix $\mathbf{A}$ (named active columns) such that they are best aligned with the residual. Upon selecting the first column $\mathbf{a}_{k_1}$ best aligned with $\mathbf{b}_0 = \mathbf{b}$, the projection of the current residual, $\mathbf{b}_0$, on the direction of $\mathbf{a}_{k_1}$ is removed from itself, thus resulting a new residual $\mathbf{b}_1$. At step i, the search for the best aligned column $\mathbf{a}_{k_i}$ continues in the set $\mathcal{I}$ of columns not yet chosen; a new column is selected such that is best aligned with the current residual $\mathbf{b}_{i-1}$

$$k_i = \arg\max_{l \in \mathcal{I}} \frac{|\mathbf{a}_l^T \mathbf{b}_{i-1}|^2}{||\mathbf{a}_l||^2}; \tag{3}$$

the new coefficient represents the alignment of the column with the residual

$$x_{k_i} = \frac{\mathbf{a}_{k_i}^T \mathbf{b}_{i-1}}{||\mathbf{a}_{k_i}||^2}; \tag{4}$$

the residual $\mathbf{b}_i$ is then computed by removing the influence of the column $\mathbf{a}_{k_i}$ form $\mathbf{b}_{i-1}$

$$\mathbf{b}_i = \mathbf{b}_{i-1} - x_{k_i} \mathbf{a}_{k_i}. \tag{5}$$

Work supported by Tekes FiDiPro – Finland Distinguished Professor Programme grant. B. Dumitrescu is also with Department of Automatic Control and Computers, "Politehnica" University of Bucharest, Romania.

The selection of active columns stops once a given number of columns M is reached.

The selection of the new column based on (3) guarantees that the residual is decreased by the largest amount at each step, without changing the values of the previously computed coefficients. Once a set of M_i columns is selected, the value of the coefficients can be updated by cyclically optimizing one coefficient at a time while holding the other $M_i - 1$ coefficients fixed. An update of the coefficient x_{k_i} is performed by removing the associated column $\mathbf{a}_{k_1}$ from the active set, restoring the influence it had in decreasing the residual

$$\mathbf{b}^{'}_{M_i} = \mathbf{b}_{M_i} + x_{k_i}\mathbf{a}_{k_i} \tag{6}$$

and reintroducing it in the active set again. By plugging equation (6) into (4), the new coefficient value is

$$x^{'}_{k_i} = \frac{\mathbf{a}_{k_i}^T \mathbf{b}^{'}_{M_i}}{||\mathbf{a}_{k_i}||^2} = x_{k_i} + \gamma, \quad \text{with } \gamma = \frac{\mathbf{a}_{k_i}^T \mathbf{b}_{M_i}}{||\mathbf{a}_{k_i}||^2}. \tag{7}$$

Using the coefficient expression and the residual updates (5) and (6), the cyclic update, made in place, is summarized by

$$\mathbf{b}_{M_i} \leftarrow \mathbf{b}^{'}_{M_i} - (x_{k_i} + \gamma)\mathbf{a}_{k_i} = \mathbf{b}_{M_i} - \gamma\mathbf{a}_{k_i} \tag{8}$$

$$x_{k_i} \leftarrow x^{'}_{k_i}. \tag{9}$$

Cyclically performing the update for each coefficient a number of times further minimizes the residual.

We propose two methods for estimating the values of M coefficients; the first, named Cyclic Adaptive Matching Pursuit (CAMP) and presented in Alg. 3, cyclically updates the coefficients after all the active columns are chosen; the second, named Iterated Cyclic Adaptive Matching Pursuit (ICAMP) and presented in Alg. 4, cyclically improves the coefficients after the introduction of each new column in the active set.

The algorithms are efficiently implemented using only information about the scalar products $\Phi_{i,j} = \mathbf{a}_i^T\mathbf{a}_j$ between the columns of the matrix $\mathbf{A}$ and the scalar products $\Psi_i = \mathbf{a}_i^T\mathbf{b}$ between the the columns of $\mathbf{A}$ and output vector $\mathbf{b}$. Equations (5), (6) and (8) can be expressed with the use of the scalar products $\mathbf{\Psi}$ and $\mathbf{\Phi}$ by multiplying on the left with $\mathbf{A}^T$, while for the others, the introduction of the scalar products is immediate.

At time t, upon receiving a new input data vector $\boldsymbol{\alpha}$ and output data β, the scalar products $\mathbf{\Phi}$ and $\mathbf{\Psi}$ are updated in place and a copy, $\tilde{\mathbf{\Psi}}$ of $\mathbf{\Psi}$, is used to store the scalar products with the current residual (Alg. 3 and 4, equation $(*)$). Due to the slow varying nature of the considered problem, when we choose the active column set (Alg. 3 and 4, step 2.1) we reuse the previous selection computed at time $t-1$ and allow changes in the column order only between neighbor positions. The only exception is the last position, for which all the remaining columns compete.

For the column selection and the coefficient estimation (Alg. 1) we use the scalar products $\tilde{\mathbf{\Psi}}$ between $\mathbf{A}$ and the current residual and update them in place (Alg. 1, steps 1.1, 2.1, 5). This is performed after each new column is selected, according to a vectorial version of (5). We note that, although the algorithm can be implemented without permuting the matrix $\mathbf{A}$, to make the presentation simpler we consider that the columns in $\mathbf{A}$ (and the other corresponding matrices) are ordered according to their influence on the residual (Alg. 1, step 3). The computation of the coefficient value (Alg. 1, step 4), if we consider the permutation influence on $\mathbf{x}$, follows directly from (4).

For both CAMP and ICAMP algorithms, after the column selection and the initial coefficient estimation, N_{it} cyclical updates are performed such that the residual is further

Alg. 1 (**Estimate the coefficient** i).

1 if $i < M$

1.1 update scalar product for next column $\tilde{\mathbf{\Psi}}_{i+1} \leftarrow \tilde{\mathbf{\Psi}}_{i+1} - \mathbf{\Phi}_{i+1,1:i-1}\mathbf{x}_{1:i-1}$

1.2 $k_i = \arg\max_{l\in[i,i+1]} \frac{\tilde{\Psi}_l^2}{\Phi_{l,l}}$ (find best candidate column searching between neighbors)

2 if $i == M$

2.1 update remaining scalar products $\tilde{\mathbf{\Psi}}_{M+1:N} \leftarrow \tilde{\mathbf{\Psi}}_{M+1:N} - \mathbf{\Phi}_{M+1:N,1:M-1}\mathbf{x}_{1:M-1}$

2.2 $k_i = \arg\max_{l\in[M:N]} \frac{\tilde{\Psi}_l^2}{\Phi_{l,l}}$ (find the best k_i candidate column searching all remaining columns)

3 swap columns (and lines) i and k_i in $\mathbf{\Phi}$ swap elements i and k_i in $\tilde{\mathbf{\Psi}}$, $\mathbf{x}$ and $\mathbf{\Psi}$

4 $x_i = \frac{\tilde{\Psi}_i}{\Phi_{i,i}}$ (evaluate the coefficient value)

5 $\tilde{\mathbf{\Psi}}_{1:i+1} \leftarrow \tilde{\mathbf{\Psi}}_{1:i+1} - x_i\mathbf{\Phi}_{1:i+1,i}$ (update the scalar product considering the new residual)

Alg. 2 (**Cyclic update of** M_i **coefficients**).

1 for $i = 1 : M_i$

1.1 $\gamma = \frac{\tilde{\Psi}_i}{\Phi_{i,i}}$

1.2 $x_i \leftarrow x_i + \gamma$ (update the coefficient)

1.3 $\tilde{\mathbf{\Psi}}_{1:M_i+1} \leftarrow \tilde{\mathbf{\Psi}}_{1:M_i+1} - \gamma\mathbf{\Phi}_{1:M_i+1,i}$

Alg. 3 (**CAMP: Cyclic adaptive matching pursuit**).

1 update scalar products with current data; save a copy of scalar products $\mathbf{\Psi}_{1:N}$

$$\left.\begin{array}{l}\mathbf{\Phi}_{1:N,1:N} \leftarrow \lambda\mathbf{\Phi}_{1:N,1:N} + \boldsymbol{\alpha}_{1:N}\boldsymbol{\alpha}_{1:N}^T \\ \mathbf{\Psi}_{1:N} \leftarrow \lambda\mathbf{\Psi}_{1:N} + \beta\boldsymbol{\alpha}_{1:N} \\ \tilde{\mathbf{\Psi}}_{1:N} = \mathbf{\Psi}_{1:N}\end{array}\right\} \quad (*)$$

2 for $i = 1 : M$ (select coefficients one by one)

2.1 estimate the coefficient i as in Alg. 1

3 for $l = 1 : N_{\text{it}}$

3.1 update $M_i = M$ coefficients as in Alg. 2

Alg. 4 (**ICAMP: Iterated cyclic adaptive matching pursuit**).

1 update scalar products like in $(*)$

2 for $i = 1 : M$ (select coefficients one by one)

2.1 estimate the coefficient i as in Alg. 1

2.2 for $l = 1 : N_{\text{it}}$

2.2.1 update $M_i = i$ coefficients as in Alg. 2

2.3 $\check{\mathbf{x}}_{i,1:i} = \mathbf{x}_{1:i}$ (store the current coefficients)

decreased. The procedure is presented in Alg. 2 and follows closely (7) and (8), the main difference consisting in the use of the scalar products $\tilde{\mathbf{\Psi}}$ defined with the residual $\mathbf{b}_{M_i}$ from (8).

The number of operations necessary for implementing the selection of active columns and computation of the coefficients is the same for both CAMP and ICAMP $\nu \approx \frac{3}{2}N^2 + \frac{1}{2}M^2 + NM$. The added complexity due to the cyclical update is different for the two algorithms; in case of CAMP it is $\rho_0 \approx N_{\text{it}}M^2$; for ICAMP the computational burden is greater, $\tau_0 \approx \frac{1}{3}N_{\text{it}}M^3 + 4N_{\text{it}}M^2$.

3. INFORMATION THEORETIC CRITERIA

The algorithms presented so far order the elements of the solution based on their contribution in decreasing the residual. If we consider that the real cardinality L_t of the solution support is unknown and we apply the algorithms for a number of non-zero elements $M \geq L_t$ then, it is plausible to assume that, when enough data are available, the first L_t positions chosen by the CAMP and ICAMP algorithms correspond with high probability to the non-zero locations of the true solution.

To find an estimate L for the true cardinality of the support L_t we employ information theoretic criteria (ITC). We use two model selection methods, Bayesian information criterion (BIC) [7] and Predictive least squares criterion (PLS) [8] as suggested by [6] for a similar order selection task.

3.1. Bayesian information criterion (BIC)

Consider, at time t, the squared norm of the residual $\Gamma_k = \mathbf{b}_k^T\mathbf{b}_k$ computed for a solution $\mathbf{x}$ with a support cardinality k; if we use the fact that $\mathbf{b}_k = \mathbf{b}_0 - \sum_{i=1}^k x_i\mathbf{a}_i$, it results that

$$\begin{aligned}\mathbf{b}_k^T\mathbf{b}_k &= \left(\mathbf{b}_0 - \textstyle\sum_{i=1}^k x_i\mathbf{a}_i\right)^T\left(\mathbf{b}_0 - \sum_{i=1}^k x_i\mathbf{a}_i\right) \\ &= \Gamma_0 - 2\textstyle\sum_{i=1}^k x_i\Phi_i + \sum_{i=1}^k\sum_{j=1}^k x_ix_j\Psi_{i,j}.\end{aligned} \tag{10}$$

Using the norm of the residual, we can define the BIC criterion [7] at time t as

$$\mathbf{BIC}_{k,t} = n_{\text{ef}}\ln\Gamma_k + (k+1)\ln n_{\text{ef}}, \tag{11}$$

where $n_{\text{ef}} = \sum_{i=0}^t \lambda^{t-i}$ is the effective number of samples used to determine the solution $\mathbf{x}$.

For the ICAMP algorithm, the criterion (11) can be computed directly with (10) and the stored values for the coefficients $\check{\mathbf{x}}_{k,1:k}$, while for CAMP the following recursion is used

$$\Gamma_k = \Gamma_{k-1} - 2x_k\Phi_k + 2x_k\mathbf{x}_{1:k}^T\Psi_{k,1:k} - x_k^2\Psi_{k,k}^2. \tag{12}$$

3.2. Predictive least squares criterion (PLS)

The PLS criterion [8] at time t is defined as

$$\mathbf{PLS}_{k,t} = \sum_{i=0}^{t}\lambda^{t-i}e_{k,i}^2, \tag{13}$$

where $e_{k,i} = \beta - \boldsymbol{\alpha}_{1:k}^T\mathbf{x}_{1:k}$ is the a priori estimation error at time i produced by the k-sparse solution $\mathbf{x}$ computed at time $i-1$, $\boldsymbol{\alpha}$ and β are the input and output at time i. The PLS criterion is given by

$$\mathbf{PLS}_{k,t} = \lambda\mathbf{PLS}_{k,t-1} + e_{k,t}^2. \tag{14}$$

Alg. 5 (**ITC CAMP**).

1 estimate the coefficients as in Alg. 3
2 estimate the support size L using BIC or PLS
3 $\tilde{\Psi}_{1:L} \leftarrow \tilde{\Psi}_{1:L} + \Phi_{1:L,L+1:M}\mathbf{x}_{L+1:M}$ (remove the contribution of the other $M-L$ columns)
4 for $l = 1 : N_{\text{it}}$
  4.1 update $M_i = L$ coefficients as in Alg. 2
5 if the variable M algorithm is used, increase or decrease M by 1 such that it approaches $L + \Delta$

Alg. 6 (**ITC ICAMP**).

1 estimate the coefficients as in Alg. 4
2 estimate the support size L using BIC or PLS
3 use the stored values $\check{\mathbf{x}}_{L,1:L}$ for the coefficients
4 if the variable M algorithm is used, increase or decrease M by 1 such that it approaches $L + \Delta$

4. ITC AND THE CAMP ALGORITHMS

We propose two variants of the algorithms. For the first we define a maximum, fixed, sparsity level M chosen large enough to accommodate all possible values for the true support size L_t and apply the ITC to choose an estimate L for the support size. The second uses a variable upper sparsity level M which is increased or decreased by one at each sample time such that it approaches $L + \Delta$, where Δ is a parameter guaranteeing that there are enough candidates for finding the best number of non-zero elements and L is the estimate for the support size. The use of a variable M ensures a lower number of operations and an improved robustness to unknown sparsity levels.

The estimation L of the true number of non-zero elements L_t results from minimizing the criterion

$$L = \arg\min_{k=1:M}\mathbf{BIC}_{k,t} \quad \text{or} \quad \mathbf{PLS}_{k,t}. \tag{15}$$

By using the BIC and PLS criteria in conjunction with CAMP and ICAMP, two algorithms result (Alg. 5 and Alg. 6).

The use of the ITC increases the complexity of the base algorithms. For CAMP, the additional complexity is $\rho_{1,\text{bic}} \approx \rho_{1,\text{pls}} \approx N_{\text{it}}L^2 + M^2$ when using BIC and PLS; for ICAMP, the added complexity is $\tau_{1,\text{bic}} \approx \frac{2}{3}M^3 + 3M^2$ for BIC or $\tau_{1,\text{pls}} \approx M^2$ for PLS.

5. SIMULATIONS

The algorithms were tested for a FIR channel identification problem (1) with $L_t = 5$ nonzero coefficients and a filter order $N = 200$. The coefficient positions are randomly chosen while their variation is described by

$$\tilde{x}_i(t) = a_i\cos(2\pi fT_st + \phi_i), \tag{16}$$

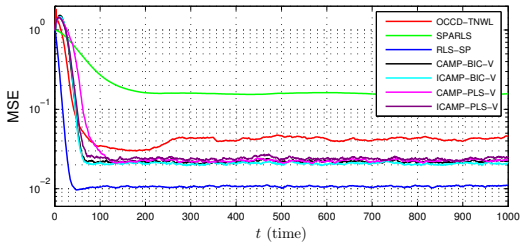


Fig. 1. Squared coefficient error for M = 5.

where the amplitude a_i and phase ϕ_i are distributed uniformly in $[0.05, 1]$ and $[0, 2\pi]$, respectively. Afterwards, the filter is normed so that its average norm is $E\{||\tilde{\mathbf{x}}||_2^2\} = 1$. The inputs are normally distributed according to $\mathcal{N}(0, 1)$ and the outputs are affected by an additive Gaussian noise with $\sigma^2 = 0.01$. The channel variation speed is controlled by the product fT_s. The measure for the performance is the coefficient mean squared error (MSE)

$$\mathbf{MSE}(t) = E\{||\tilde{\mathbf{x}} - \mathbf{x}||_2^2\}, \tag{17}$$

where $\tilde{\mathbf{x}}$ contains the real value of the coefficients and $\mathbf{x}$ their estimate. The estimate of the MSE is computed over 1000 runs for each of the variation speeds fT_s.

In Table 1 we present the MSE of several algorithms, averaged over the last 100 samples, for different variation speeds fT_s and forgetting factors λ: RLS-SP, the sparsity informed RLS algorithm with prior knowledge of the nonzero filter coefficients positions; RLS, the standard algorithm that considers a full filter of order N; GRLS, the greedy algorithm with prior knowledge of the support size from in [6]; CAMP and ICAMP, the variants of our algorithms with prior knowledge of the support size; (I)CAMP-PLS/BIC-F/V, the fixed and variable M variants of the algorithms presented in Alg. 5 and 6; SPARLS, the algorithm presented in [3]; OCCD-TNWL, the best of the algorithms from [2].

The configuration of the GRLS, SPARLS and OCCD-TNWL algorithms is done according to the recommendations from the corresponding articles. The parameter γ, used in SPARLS and presented in Table 1, was optimized using a grid search. For GRLS, CAMP and ICAMP algorithms the value for the sparsity was chosen $M = L_t$; the algorithms that employ ITC use $M = 20$ for the fixed threshold version and $\Delta = 5$ for the variable threshold algorithm; for all our algorithms the number of cyclic optimization rounds was $N_{\text{it}} = 5$.

In Fig. 1 we present the time evolution of our algorithms with a variable upper sparsity level limit together with evolution of OCCD-TNWL, SPARLS and RLS-SP for $fT_s = 0.001$.

6. CONCLUSIONS

We proposed two adaptive MP algorithm families (CAMP and ICAMP) employing a cyclical coefficient re-computation and using ITC (BIC and PLS) to estimate on-line the support size. The complexity of the algorithms is lower than that of competing methods; for the ICAMP algorithms the MSE is in general comparable with that of GRLS, while for the less complex CAMP algorithms the MSE slightly degrades; the additional complexity due to the ITC is low if M is small while giving increased robustness.

Table 1. MSE for the studied algorithms.

fT_s	0.002	0.001	0.0005	0.0002	0.0001
λ	0.90	0.92	0.94	0.96	0.98
RLS	5.3012	1.4565	0.36427	0.09774	0.04734
RLS-SP	0.0267	0.0110	0.00518	0.00246	0.00306
GRLS	0.0501	0.0178	0.00785	0.00343	0.00343
CAMP	0.0700	0.0266	0.01161	0.00453	0.00352
ICAMP	0.0534	0.0181	0.00790	0.00346	0.00343
CAMP-BIC-F	0.0805	0.0244	0.00928	0.00380	0.00367
CAMP-BIC-V	0.0649	0.0225	0.00984	0.00445	0.00379
CAMP-PLS-F	0.0899	0.0288	0.01117	0.00430	0.00371
CAMP-PLS-V	0.0745	0.0246	0.01030	0.00430	0.00366
ICAMP-BIC-F	0.0769	0.0220	0.00856	0.00377	0.00377
ICAMP-BIC-V	0.0639	0.0210	0.00992	0.00487	0.00397
ICAMP-PLS-F	0.0881	0.0249	0.00898	0.00343	0.00348
ICAMP-PLS-V	0.0683	0.0194	0.00762	0.00326	0.00340
SPARLS	0.4417	0.1578	0.04225	0.01120	0.00767
γ	170	110	75	50	75
OCCD-TNWL	0.4802	0.0436	0.01231	0.00447	0.00372

7. REFERENCES

[1] S.G. Mallat and Z. Zhang, "Matching Pursuit with Time Frequency Dictionaries," *IEEE Trans. Sgn. Proc.*, vol. 41, no. 12, pp. 3397–3415, 1993.

[2] D. Angelosante, J.A. Bazerque, and G.B. Giannakis, "Online Adaptive Estimation of Sparse Signals: Where RLS Meets the l_1-Norm," *IEEE Trans. Sign. Proc.*, vol. 58, no. 7, pp. 3436–3447, July 2010.

[3] B. Babadi, N. Kalouptsidis, and V. Tarokh, "SPARLS: The Sparse RLS Algorithm," *IEEE Trans. Sign. Proc.*, vol. 58, no. 8, 2010.

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

[5] M.G. Christensen and S.H. Jensen, "The Cyclic Matching Pursuit and its Application to Audio Modeling and Coding," in *41th Asilomar Conf. Sign. Syst. Comp.*, nov. 2007, pp. 550–554.

[6] B. Dumitrescu, A. Onose, P. Helin, and I. Tăbuş, "Greedy Sparse RLS," *Submitted to IEEE Trans. Sign. Proc.*, July 2011.

[7] G. Schwarz, "Estimating the Dimension of a Model," *Ann. Stat.*, vol. 6, no. 2, pp. 461–464, 1978.

[8] J. Rissanen, "Order Estimation by Accumulated Prediction Errors," *J. Appl. Probab.*, vol. 23, pp. 55–61, 1986.

Publication 4

[P4] A. Onose and B. Dumitrescu. Low complexity approximate cyclic adaptive matching pursuit. In *Proceedings of the European Signal Processing Conference*, pages 2629–2633, Bucharest, Romania, August 2012.

LOW COMPLEXITY APPROXIMATE CYCLIC ADAPTIVE MATCHING PURSUIT

Alexandru Onose, Bogdan Dumitrescu

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

ABSTRACT

Based on the iterated cyclic adaptive matching pursuit algorithm, we construct a low complexity approximate variant for finding sparse solutions to systems of linear equations. We employ a greedy neighbor permutation strategy coupled with an approximate scalar product matrix to ensure that the complexity of the algorithm remains low. The sparse solution is cyclically updated improving the performance while the sparsity level is estimated online using the predictive least squares criterion. The performance of the algorithm is similar to that of the non approximate variants while the complexity can be considerably lower.

Index Terms— matching pursuit, sparse filters, greedy algorithm, channel identification

1. INTRODUCTION

Sparse approximation problems have arisen in recent years from many practical application like compression, denoising or array processing. Several algorithms [1, 2, 3] have been proposed with applications to typical signal and image processing tasks. We aim to present in this paper a low complexity version of the iterated cyclic adaptive matching pursuit (ICAMP) [4] algorithm that employs a series of approximations and adaptation steps to provide a lower complexity without negatively influencing the estimation error.

Consider a typical FIR channel identification problem where, at time t, the input $u(t)$ and the output $d(t)$ are measured. The goal is to estimate the true coefficients h_j that minimize the estimation error

$$e(t) = d(t) - \sum_{j=0}^{N-1} h_j u(t-j), \tag{1}$$

where N is the filter length.

Work supported by Tekes FiDiPro – Finland Distinguished Professor Programme and by GETA – Graduate School in Electronics, Telecommunications and Automation. B. Dumitrescu is also with Department of Automatic Control and Computers, "Politehnica" University of Bucharest, Romania.

Minimizing the estimation error at each time instance can be transformed towards minimizing a least squares criterion

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

where $0 < \lambda \leq 1$ is a forgetting factor. Written in matrix form, the criterion is equivalent to minimizing the norm of the residual $||\mathbf{b} - \mathbf{A}\mathbf{x}||^2$; the solution vector $\mathbf{x}$ corresponds to the estimated coefficients h_j that result from the minimization of (2). We consider the solution $\mathbf{x}$ to be sparse, henceforth it has at most $M \ll N$ non zero coefficients. Moreover, the number and the positions of the coefficients are a priori unknown. The matrix $\mathbf{A} \in \mathbb{R}^{t \times N}$ is built with the input data, its i-th row being equal to $\lambda^{\frac{t-i}{2}} \boldsymbol{\alpha}^{(i)T}$ where

$$\boldsymbol{\alpha}^{(i)} = [u(i), u(i-1), \ldots, u(i-N+1)]^T. \tag{3}$$

The vector $\mathbf{b} \in \mathbb{R}^t$ contains the weighted output data $b_i = \lambda^{\frac{t-i}{2}} d(i)$. We denote hereafter $\beta^{(i)} = d(i)$ and, for simplicity, we drop the index (i) from $\boldsymbol{\alpha}^{(i)}$ and $\beta^{(i)}$ when they refer to the current available data at time t.

We develop an approximate variant of the ICAMP algorithm by selectively storing the information about the input data matrix $\mathbf{A}$. The algorithm updates and uses only the scalar products between the full matrix $\mathbf{A}$ and a set of columns from $\mathbf{A}$ that are associated with the current solution support. When a decision to modify the support is made, we delay the actual change by introducing a constant length buffer to temper any rapid support variation. The scalar products associated with the buffer are also updated and can contribute directly to changes in the support. We propose two algorithms that use a fixed respectively a variable upper sparsity bound and estimate the sparsity level with the predictive least squares (PLS) criterion.

The content of this paper is as follows: in section 2 we present the proposed low complexity approximate algorithms; a brief overview of the ICAMP algorithm and the PLS criterion in conjunction with the approximate algorithms is included in section 3; section 4 contains the results of our simulations proving the performance of the algorithms.

2. LOW COMPLEXITY APPROXIMATE ALGORITHMS

We begin by briefly reviewing the classic matching pursuit (MP) algorithm [5] and its adaptive counterpart, the adaptive matching pursuit (AMP) [6]. Starting from AMP, we introduce a series of approximations that provide lower complexity which, used in conjunction with the iterated cyclic adaptive matching pursuit algorithm [4] coupled with a buffer that mitigates fast changes in the solution support, form the basis of our proposed low complexity iterated adaptive matching pursuit algorithm.

The algorithm follows the MP column selection strategy selecting one by one columns (named active columns) from the matrix $\mathbf{A}$ based on their alignment with the residual. The selection starts by finding the best column $\mathbf{a}_{k_1}$ aligned with output data vector representing the first residual $\mathbf{b}_0 = \mathbf{b}$ (the solution $\mathbf{x}$ is null); the column $\mathbf{a}_{k_1}$ is included in the active column set $\mathcal{A}$. The projection of the current residual $\mathbf{b}_0$ on $\mathbf{a}_{k_1}$ is then removed from $\mathbf{b}_0$, producing the new residual $\mathbf{b}_1$ used in the selection of the next active column. After $i-1$ columns are selected, the search for the column $\mathbf{a}_{k_i}$, best aligned with the current residual $\mathbf{b}_{i-1}$, continues in the remaining column set $\mathcal{I}$ composed by columns not yet included in $\mathcal{A}$,

$$k_i = \arg\max_{l \in \mathcal{I}} \frac{|\mathbf{a}_l^T \mathbf{b}_{i-1}|^2}{||\mathbf{a}_l||^2}. \tag{4}$$

The new residual is computed by projecting $\mathbf{b}_{i-1}$ on $\mathbf{a}_{k_i}$,

$$\mathbf{b}_i = \mathbf{b}_{i-1} - x_{k_i} \mathbf{a}_{k_i}, \tag{5}$$

where the coefficient x_{k_i} represents the alignment of the column $\mathbf{a}_{k_i}$ with the residual $\mathbf{b}_{i-1}$,

$$x_{k_i} = \frac{\mathbf{a}_{k_i}^T \mathbf{b}_{i-1}}{||\mathbf{a}_{k_i}||^2}. \tag{6}$$

The selection continues until M columns are chosen.

The above equations are implemented using only the scalar products $\mathbf{\Psi} = \mathbf{A}^T\mathbf{b}$ between the input matrix and the output vector $\mathbf{b}$ and $\mathbf{\Phi} = \mathbf{A}^T\mathbf{A}$ between the columns of the matrix $\mathbf{A}$. The norms of the columns from the matrix $\mathbf{A}$ are stored separately in $\mathbf{\Theta}$ to simplify the presentation, otherwise they are present on the diagonal of $\mathbf{\Phi}$. Equation (5) can be easily expressed using the scalar products by multiplying it with $\mathbf{A}^T$ on the left.

Additionally, a neighbor search strategy is employed for the column selection. Relying on the slow variation of the channel, we reuse the selection order found at time $t-1$ when we perform the search at time t. Thus, starting from the old active set $\mathcal{A}$ we constrain the search for each new column position j between neighbor positions j and $j+1$ in the set. The resulting permutations are performed only between neighbor positions and produce the updated active set at current time t. Changes in the composition of the active set $\mathcal{A}$ are achieved by allowing all remaining inactive columns to compete for the last position M.

At each time instance t, when new input vector $\boldsymbol{\alpha}$ and output data β are available, all the scalar products are updated in place,

$$\begin{array}{lll} \mathbf{\Phi}_{1:N,1:N} & \leftarrow & \lambda \mathbf{\Phi}_{1:N,1:N} + \boldsymbol{\alpha}_{1:N} \boldsymbol{\alpha}_{1:N}^T \\ \mathbf{\Psi}_{1:N} & \leftarrow & \lambda \mathbf{\Psi}_{1:N} + \beta \boldsymbol{\alpha}_{1:N} \\ \mathbf{\Theta}_j & \leftarrow & \lambda \mathbf{\Theta}_j + \alpha_j^2 \ \text{ with } \ j = 1:N, \end{array} \tag{7}$$

and the solution and column selection are revised.

Consider the true sparse solution having $L_t \ll N$ non zero coefficients; by choosing $M \geq L_t$ small enough the main computational burden is due to the update of the scalar products $\mathbf{\Phi}$. They are however not all necessary when computing the solution at a given time. In a stationary regime, where the active set $\mathcal{A}$ composition does not change as new samples are received, the only scalar products required are the column norms $\mathbf{\Theta}$ and the scalar products between the active columns $\mathbf{a}_{k_{1:M}}$ and the full matrix $\mathbf{A}$. Thus, in the ideal scenario where no support changes are made, storing and updating only the partial scalar product matrix $\bar{\mathbf{\Phi}} = \mathbf{A}^T \mathbf{a}_{k_{1:M}}$ instead of the whole $\mathbf{\Phi} = \mathbf{A}^T\mathbf{A}$ ensures that all the required data are available.

When an inactive column $\mathbf{a}_{k_j}$ becomes active instead of the old column $\mathbf{a}_{k_M}$, the associated coefficient $x_{k_j}^{'}$ can still be computed precisely like in (6) using the stored column norms $\mathbf{\Theta}$; it produces an updated residual

$$\mathbf{b}_M^{'} = \mathbf{b}_{M-1} - x_{k_j}^{'} \mathbf{a}_{k_j}. \tag{8}$$

Because the channel is slow varying, there are usually no sudden changes, either in the support or in the coefficient values. Thus, when a coefficient becomes inactive it does so gradually until the associated column reaches the last position $\mathbf{a}_{k_M}$ in the active set $\mathcal{A}$ and is then replaced by the new column $\mathbf{a}_{k_j}$. If $M > L_t$, then we can assume that any coefficient x_{k_i} above the sparsity level L_t ($i > L_t$) is small since the true solution contains all the relevant coefficients. When the support changes, the coefficient $x_{k_j}^{'}$ has a negligible influence in decreasing the residual and the scalar products associated with (8) can be approximated by

$$\mathbf{A}^T \mathbf{b}_M^{'} \approx \mathbf{A}^T \mathbf{b}_{M-1} \tag{9}$$

This does not require any information regarding the unknown scalar products $\mathbf{A}^T\mathbf{a}_{k_j}$. When the new coefficient gradually become significant, if the unknown scalar products $\mathbf{A}^T\mathbf{a}_{k_j}$ are far from their true values, the performance is negatively influenced even leading to instability. To circumvent this we propose to set $\mathbf{A}^T\mathbf{a}_{k_j}$ to zero and allow a number of updates to be performed before the column may be introduced into the active set. For this purpose we use a buffer $\mathcal{B}$ of length P (containing columns $\mathbf{a}_{k_{M+1:M+P}}$) to delay the introduction of any new column in $\mathcal{A}$. The scalar products are updated

for all the columns associated with $\mathcal{B}$, hence diminishing the approximation errors. Furthermore, since the scalar products with the columns from the active set are computed exactly, the errors introduced by the unknown scalar products after the column is included in $\mathcal{A}$ on position i do not affect significantly the coefficients $x_{k_{1:i}}$.

Thus, if the column $\mathbf{a}_{k_j}$, selected to be introduced in the active set $\mathcal{A}$ on position M, belongs to $\mathcal{I} \setminus \mathcal{B}$, it replaces the last column $\mathbf{a}_{k_{M+P}}$ in $\mathcal{B}$ instead of being introduced directly in the active set $\mathcal{A}$. The associated scalar products are set to zero and the solution is still computed with the old column $\mathbf{a}_{k_M}$. If $\mathbf{a}_{k_j}$ belongs to $\mathcal{B}$, it is promoted one position in the set, $k_{j-1} \leftrightarrow k_j$. It becomes active replacing $\mathbf{a}_{k_M}$ in the active set, $k_M \leftrightarrow k_{M+1}$, only when it is on the first position in $\mathcal{B}$. This ensures that a certain column is selected at least P times before becoming active which, coupled with the update of their associated scalar products as new samples are received, reduces the approximation errors.

We present in Alg. 1 the algorithm that constructs the set $\mathcal{B}$ and performs the neighbor permutations. We consider that the elements in $\mathbf{x}$, the columns in the matrix $\mathbf{A}$ and all the associated scalar products are permuted according to the column selection order implicitly present in the active set $\mathcal{A}$ and in the buffer $\mathcal{B}$. Additionally, the partial matrix $\bar{\mathbf{\Phi}}$ is stored in the full matrix $\mathbf{\Phi}$ to simplify the notations. Also, note that the scalar products between the active columns (and columns from $\mathcal{B}$) and all the other columns have exact values and are never set to zero (Alg. 1, step 1.3).

The complexity due to the scalar product update $\mathbf{\Phi}$ is reduced from $\frac{3}{2}N^2$ to $3N(M+P) - \frac{3}{2}(M+P)^2$. The scalar product update complexity is decreased by a factor proportional with $\frac{1}{2}\frac{N}{M+P}$.

3. APPROXIMATE ITERATED CYCLIC ADAPTIVE MATCHING PURSUIT

Selecting the column $\mathbf{a}_{k_i}$ and computing the solution coefficient x_{k_i} like in the AMP algorithm decreases the residual by the largest amount when the previously computed coefficients, $x_{k_{1:i-1}}$, are kept fixed. To further minimize the residual, a cyclical update [7] is performed by optimizing one coefficient at a time while keeping the rest $i-1$ coefficients fixed. The coefficient x_{k_j}, $j \in \{1, \ldots, i\}$ is updated by removing the corresponding column $\mathbf{a}_{k_j}$ from the active set, hence producing the residual

$$\mathbf{b}_i^{'} = \mathbf{b}_i + x_{k_j}\mathbf{a}_{k_j}, \tag{10}$$

and then reincluding it in the active set. The updated coefficient is

$$x_{k_j}^{'} = \frac{\mathbf{a}_{k_j}^T \mathbf{b}_i^{'}}{||\mathbf{a}_{k_j}||^2} = x_{k_j} + \gamma, \tag{11}$$

Alg. 1 (**Introduce the candidate column j in $\mathcal{B}$, perform the necessary neighbor permutations**).

1 if $j > M+P$ (the column is not in buffer $\mathcal{B}$)

- 1.1 Swap columns (and rows) $M+P$ and j in $\mathbf{\Phi}$
  Swap elements $M+P$ and j in $\tilde{\mathbf{\Psi}}$, $\mathbf{x}$ and $\mathbf{\Psi}$
  The information on row j in $\mathbf{\Phi}$ is discarded
- 1.2 Set to zero the unknown scalar products for column $M+P$
  $\Phi_{M+P+1:N,M+P} = 0$
- 1.3 Keep the known scalar products
  $\Phi_{M+P,M+P} = \Theta_{M+P}$
  $\Phi_{1:M+P-1,M+P} = \Phi_{M+P,1:M+P-1}^T$

2 if $1 < j \leq M+P$ (the column is in $\mathcal{B}$ or in $\mathcal{A}$)

- 2.1 Swap columns (and rows) $j-1$ and j in $\mathbf{\Phi}$
  Swap elements $j-1$ and j in $\tilde{\mathbf{\Psi}}$, $\mathbf{x}$ and $\mathbf{\Psi}$

where $\gamma = \frac{\mathbf{a}_{k_j}^T \mathbf{b}_i}{||\mathbf{a}_{k_j}||^2}$. This produces a residual update similar to (5),

$$\begin{aligned} \mathbf{b}_i \leftarrow \mathbf{b}_i^{'} - (x_{k_j} + \gamma)\mathbf{a}_{k_j} = \mathbf{b}_i - \gamma \mathbf{a}_{k_j} \\ x_{k_j} \leftarrow x_{k_j}^{'}. \end{aligned} \tag{12}$$

The cyclic update step, performed N_{it} times for all columns considered by the solution, further minimizes the residual. All the above updates can be expressed with the use of the scalar products $\mathbf{\Psi}$ and the partial scalar products $\bar{\mathbf{\Phi}}$ such that the approximate version of the ICAMP algorithm only requires the approximation steps presented in section 2. The column selection and coefficient estimation is presented in Alg. 2 while the cyclic update is included in Alg. 3.

The true sparsity level L_t of the solution $\mathbf{x}$ can be estimated with the use of information theoretic criteria [3, 4]. If L_t is unknown, we can apply the relations presented so far to find M, $M \geq L_t$, columns $\mathbf{a}_{k_{1:M}}$ that correspond to a M-sparse solution $\mathbf{x}$. Because the selection process introduces an inherent column order, we assume that after enough data are available the first selected positions correspond with high probability to the true support of the solution. The threshold M can be either fixed or can change such that it approaches $L_t + \Delta$, where Δ is a predefined constant ensuring that there are enough candidates for estimating the sparsity level [4]. For clarity we present only the fixed version of the algorithm, the variable one being rather straightforward. Note that for the variable version, due to the changes of the threshold M, the buffer $\mathcal{B}$ needs to be adjusted such that its size is always P. Thus, if M decreases, the last column from the active set $\mathcal{A}$ will be moved to $\mathcal{B}$ while the last column in $\mathcal{B}$ is discarded. If M increases, the first column from $\mathcal{B}$ moves to $\mathcal{A}$ and on the last position in $\mathcal{B}$ a new (random) column is promoted.

The estimate L of the true sparsity level L_t is computed as the point for which the predictive least squares (PLS) criterion

Alg. 2 (**Estimate the coefficient** i).

1 if $i < M$

1.1 Update scalar product for next column
$\tilde{\mathbf{\Psi}}_{i+1} \leftarrow \tilde{\mathbf{\Psi}}_{i+1} - \mathbf{\Phi}_{i+1,1:i-1}\mathbf{x}_{1:i-1}$

1.2 Find best candidate column between neighbors
$k_i = \arg\max_{l\in[i,i+1]} \frac{\tilde{\Psi}_l^2}{\Theta_l}$

2 if $i == M$

2.1 Update remaining scalar products
$\tilde{\mathbf{\Psi}}_{M+1:N} \leftarrow \tilde{\mathbf{\Psi}}_{M+1:N} - \mathbf{\Phi}_{M+1:N,1:M-1}\mathbf{x}_{1:M-1}$

2.2 Find best k_i candidate column between all remaining columns
$k_i = \arg\max_{l\in[M:N]} \frac{\tilde{\Psi}_l^2}{\Theta_l}$

3 Permute necessary columns (Alg. 1)

4 $x_i = \frac{\tilde{\Psi}_i}{\Phi_{i,i}}$ (evaluate coefficient value)

5 Update scalar product considering new residual
$\tilde{\mathbf{\Psi}}_{1:i+1} \leftarrow \tilde{\mathbf{\Psi}}_{1:i+1} - x_i\mathbf{\Phi}_{1:i+1,i}$

Alg. 3 (**Cyclic update of** j **coefficients**).

1 for $i = 1 : j$ (cyclically update each coefficient)

1.1 $\gamma = \frac{\tilde{\Psi}_i}{\Theta_i}$

1.2 $x_i \leftarrow x_i + \gamma$ (update the coefficient)

1.3 $\tilde{\mathbf{\Psi}}_{1:M_i+1} \leftarrow \tilde{\mathbf{\Psi}}_{1:M_i+1} - \gamma\mathbf{\Phi}_{1:M_i+1,i}$

[8] attains its minimum. At time instance t, the PLS criterion is

$$\mathbf{PLS}_j^{(t)} = \sum_{i=0}^{t} \lambda^{t-i} {e_j^{(i)}}^2, \tag{13}$$

where $e_j^{(i)} = \beta^{(i)} - \boldsymbol{\alpha}_{k_{1:j}}^{(i)T}\check{\mathbf{x}}_{j,k_{1:j}}$; the index (i) denotes the time at which the data are considered, $\check{\mathbf{x}}_{j,k_{1:j}}^{(i)}$ contains the coefficients of the j-sparse solution. The PLS criterion can be therefore computed recursively by

$$\mathbf{PLS}_j^{(t)} = \lambda \cdot \mathbf{PLS}_j^{(t-1)} + {e_j^{(t)}}^2. \tag{14}$$

The approximate ICAMP algorithm using the PLS criterion is summarized in Alg. 4. The number of operations required by the column selection and coefficient computation is $3N(M+P) - M^2 + NM$ while the cyclic update adds $\frac{1}{3}N_{\text{it}}M^3 + 4N_{\text{it}}M^2$ operation. The computation of the PLS criterion is fast requiring only M^2 operations.

4. SIMULATIONS

The performance of the algorithms was tested using a sparse FIR channel identification problem (1) with $N = 200$ and the true solutions having $L_t = 5$ non zero coefficients. The channel non zero coefficients are described by

$$h_j(t) = a_j \cos(2\pi f T_s t + \phi_j). \tag{15}$$

Alg. 4 (**Iterated cyclic adaptive matching pursuit with PLS column selection, approximate version**).

1 Update needed scalar products between columns of $\mathbf{A}$
$\mathbf{\Phi}_{1:N,1:M+P} \leftarrow \lambda\mathbf{\Phi}_{1:N,1:M+P} + \boldsymbol{\alpha}_{1:N}\boldsymbol{\alpha}_{1:M+P}^T$
Update scalar products $\mathbf{\Theta}$ and $\mathbf{\Psi}$ like in (7)
Save copy of scalar products $\tilde{\mathbf{\Psi}}_{1:N} = \mathbf{\Psi}_{1:N}$

2 for $i = 1 : M$ (select coefficients one by one)

2.1 Estimate coefficient i as in Alg. 2

2.2 for $l = 1 : N_{\text{it}}$ (improve the j-sparse solution)

2.2.1 Update $j = i$ coefficients as in Alg. 3

2.3 $\check{\mathbf{x}}_{i,1:i} = \mathbf{x}_{1:i}$ (store current coefficients)

3 Estimate the support size L using PLS

4 Use the stored values $\check{\mathbf{x}}_{L,1:L}$ for the coefficients

The coefficient positions j are chosen randomly, the amplitude a_j and the initial phase ϕ_j are uniformly distributed in $[0.05, 1]$ and $[0, 2\pi]$. The variation speed is given by the product fT_s. The filter is normed such that the average norm is $E\{||h_i||_2^2\} = 1$. The inputs $d(t)$ are normally distributed according to $\mathcal{N}(0, 1)$ and the outputs are corrupted by an additive Gaussian noise with $\sigma^2 = 0.01$. We measure the performance of the algorithms in terms of the coefficient mean square error

$$\mathbf{MSE}(t) = E\{||\mathbf{h} - \mathbf{x}||_2^2\}, \tag{16}$$

where $\mathbf{h}$ contains the actual values of the coefficients and $\mathbf{x}$ their estimates; 1000 test runs were used to estimate the MSE.

The tested algorithms are as follows: RLS, the standard algorithm using the full filter of length N; RLS-SP, the sparsity aware RLS algorithm with prior knowledge of the position and number of coefficients; GRLS and ICAMP algorithms from [3] respectively from [4] and ICAMP-A, the approximate algorithm presented herein, with prior knowledge of the support size, $M = L_t$; GRLS/ICAMP(-A)-F/V, the versions of the above algorithms that estimate online the sparsity level using the PLS criterion in conjunction with a fixed (-F) threshold $M = 20$ or a variable (-V) threshold with $\Delta = 5$; SPARLS, the algorithm presented in [2]; TNWL, the best of the algorithms from [1], using an optimized forgetting factor for the inner RLS loop.

The GRLS, SPARLS and TNWL algorithms are configured according to the recommendations from the corresponding articles. The parameter γ used in SPALRS and the forgetting factor f_{RLS} used in the inner RLS loop in TNWL were optimized using a grid search. For all the cyclic algorithms the number of optimization rounds was $N_{\text{it}} = 5$.

In Table 1 we present the average MSE (averaged over the last 100 samples) for all the studied algorithms. Fig. 1 contains the time evolution of our algorithm using a fixed M for $fT_s = 0.001$; at time $t = 500$, three of the $L_t = 5$ coefficients randomly change positions. In Fig. 2 we present the

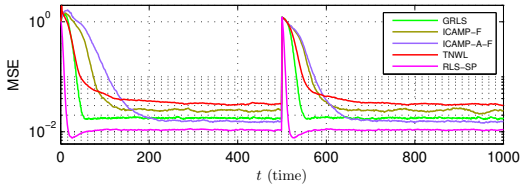


Fig. 1. MSE for $L_t = 5$ and $fT_s = 0.001$.

Table 1. Average MSE for the studied algorithms.

fT_s	0.002	0.001	0.0005	0.0002	0.0001
λ	0.90	0.92	0.94	0.96	0.98
RLS	5.3012	1.4565	0.36427	0.09774	0.04734
RLS-SP	0.0267	0.0110	0.00518	0.00246	0.00306
GRLS	0.0501	0.0178	0.00785	0.00343	0.00343
GRLS-F	0.0511	0.0189	0.00853	0.00356	0.00357
GRLS-V	0.0569	0.0187	0.00762	0.00330	0.00340
ICAMP	0.0534	0.0181	0.00790	0.00346	0.00343
ICAMP-F	0.0881	0.0249	0.00898	0.00343	0.00348
ICAMP-V	0.0683	0.0194	0.00762	0.00326	0.00340
ICAMP-A	0.0321	0.0139	0.00673	0.00310	0.00338
ICAMP-A-F	0.0409	0.0155	0.00706	0.00320	0.00342
ICAMP-A-V	0.0370	0.0146	0.00703	0.00323	0.00340
SPARLS	0.4417	0.1578	0.04225	0.01120	0.00767
γ	170	110	75	50	75
TNWL	0.1491	0.0311	0.01248	0.00471	0.00386
$\lambda_{\mathrm{RLS_{opt}}}$	0.9975	0.9825	0.9800	0.9850	0.9900

evolution of the MSE and the convergence time as a function of the buffer length P, also for $fT_s = 0.001$. For low values of the buffer length P the algorithms (ICAMP-A-F for $P = 2$, ICAMP-A-V for $P = 2, 3$) exhibit large errors for a small number of tests; we removed the associated test runs from the data presented in the figures. The buffer $\mathcal{B}$ mitigates any unwanted rapid changes in the support and allows the algorithms to achieve a lower MSE. The length P can be chosen as a trade off between the complexity and the MSE; it should be selected sufficiently large to reject the possible instabilities due to the approximations made; too large values have a negative impact on the convergence time (Fig. 2).

5. CONCLUSIONS

We propose an adaptive low complexity algorithm, employing a fixed length buffer and a series of approximations, able to outperform the non approximate counterpart and other competing adaptive algorithms. The decrease in complexity due to the update of the scalar products is proportional with

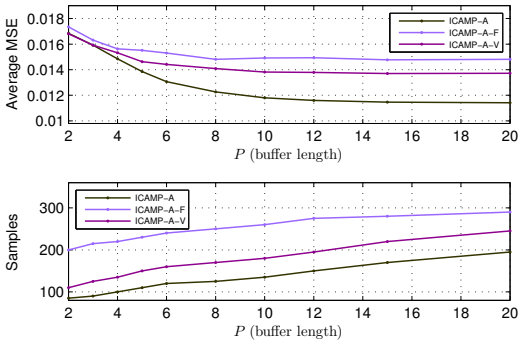


Fig. 2. Top: Average MSE as a function of the buffer length P. Bottom: Convergence time (number of samples until the MSE is within 5% of the average MSE) as a function of the buffer length P. The variation time is given by $fT_s = 0.001$.

$\frac{1}{2}\frac{N}{M+P}$ which, for low sparsity levels and small buffers, becomes substantial. The choice of buffer length P is a trade off between complexity and robustness.

6. REFERENCES

[1] D. Angelosante, J.A. Bazerque, and G.B. Giannakis, "Online Adaptive Estimation of Sparse Signals: Where RLS Meets the l_1-Norm," *IEEE Trans. Sign. Proc.*, vol. 58, no. 7, pp. 3436–3447, Jul. 2010.

[2] B. Babadi, N. Kalouptsidis, and V. Tarokh, "SPARLS: The Sparse RLS Algorithm," *IEEE Trans. Sign. Proc.*, vol. 58, no. 8, 2010.

[3] B. Dumitrescu, A. Onose, P. Helin, and I. Tăbuş, "Greedy Sparse RLS," *IEEE Trans. Sign. Proc. (to appear)*, 2012.

[4] A. Onose and B. Dumitrescu, "Cyclic Adaptive Matching Pursuit," in *ICASSP, Kyoto, Japan*, Mar. 2012.

[5] S.G. Mallat and Z. Zhang, "Matching Pursuit with Time Frequency Dictionaries," *IEEE Trans. Sgn. Proc.*, vol. 41, no. 12, pp. 3397–3415, 1993.

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

[7] M.G. Christensen and S.H. Jensen, "The Cyclic Matching Pursuit and its Application to Audio Modeling and Coding," in *41th Asilomar Conf. Sign. Syst. Comp.*, Nov. 2007, pp. 550–554.

[8] J. Rissanen, "Order Estimation by Accumulated Prediction Errors," *J. Appl. Probab.*, vol. 23, pp. 55–61, 1986.

Publication 5

[P5] B. Dumitrescu, A. Onose, P. Helin, and I. Tăbuş. Greedy sparse RLS. *IEEE Transaction on Signal Processing*, 60(5):2194–2207, May 2012.

Greedy Sparse RLS

Bogdan Dumitrescu, *Member, IEEE*, Alexandru Onose, *Student Member, IEEE*, Petri Helin, Ioan Tăbuş, *Senior Member, IEEE*

***Abstract*—Starting from the orthogonal (greedy) least squares method, we build an adaptive algorithm for finding online sparse solutions to linear systems. The algorithm belongs to the exponentially windowed RLS family and maintains a partial orthogonal factorization with pivoting of the system matrix. For complexity reasons, the permutations that bring the relevant columns into the first positions are restrained mainly to interchanges between neighbors at each time moment. The storage scheme allows the computation of the exact factorization, implicitly working on indefinitely long vectors. The sparsity level of the solution, i.e. the number of nonzero elements, is estimated using information theoretic criteria, in particular Bayesian Information Criterion and Predictive Least Squares. We present simulations showing that, for identifying sparse time-varying FIR channels, our algorithm is consistently better than previous sparse RLS methods based on the ℓ_1-norm regularization of the RLS criterion. We also use our sparse greedy RLS algorithm for computing linear predictions in a lossless audio coding scheme and obtain better compression than MPEG4 ALS using an RLS-LMS cascade.**

***Index Terms*—adaptive algorithms, sparse filters, orthogonal least squares, channel identification, audio coding**

I. Introduction

Sparse solutions to systems of linear equations can be found, in various ways, by many algorithms developed in the latest 15 years, with applications to typical signal and image processing problems [1]. Most of these algorithms are in batch form, i.e. need the whole data for computing the solution. However, in applications like channel identification or echo cancellation, where sparse filters are natural models, adaptive processing is the adequate approach, able to cope with the time-varying environment. Adaptive algorithms for sparse problems have received less attention until recent years. They have the extra challenge that not only the values of the filter coefficients may change in time, but also its support. Here is a short review of existing literature.

We discern two main directions of research. The first draws its tools from the traditional arsenal of adaptive filtering and is illustrated by papers like [2], [3], [4], [5], dealing mostly with enhancements of the LMS algorithm. The second starts from typical batch sparse approximation algorithms and tries to find efficient adaptive counterparts. Convex relaxation techniques (Basis Pursuit), in which the ℓ_1 norm of the filter is minimized together with an error criterion, have received most of the attention. They have been used for deriving a specific LMS in [6] and sparse RLS algorithms in [7] and [8]; the latter two works (practically simultaneous and listed here in alphabetical order) use low-complexity algorithms, namely coordinate descent and approximate expectation maximization, for adapting the filter coefficients at each time step; both methods employ an exponential window. An algorithm based on projections onto convex sets, using a sliding window and ℓ_1 errors, was proposed in [9]. Expectation maximization, used in [8], is extended to other adaptive algorithms in [10].

Greedy algorithms for sparse approximation have inspired less adaptive methods. Matching Pursuit (MP) can be converted to a low complexity adaptive form, as done in [11], but its performance is usually much worse than that of other sparse approximation algorithms. There is a mention to Orthogonal MP (OMP) used in time-varying context [12]; however, the paper gives no efficient implementation. In [13], an extended support is greedily estimated, normalized LMS is used for coefficient adaptation, then only the most significant coefficients are retained.

We propose here a greedy RLS (GRLS) algorithm with exponential window, based on the Orthogonal Least Squares (OLS) algorithm [14], known also as optimized OMP [15]. To our knowledge, this is the first such algorithm. Essentially, at each time moment, the algorithm updates a partial orthogonal triangularization with pivoting of the data matrix. As such, it is able to compute a true sparse least-squares solution. Since OLS or OMP could imply a complete reshuffle of the matrix columns, we put a cap on complexity by allowing almost only neighbor permutations, i.e. permutations between adjacent columns of the data matrix. Moreover, we are able to limit also the necessary memory, by efficiently storing information that allows the use of Householder reflectors on indefinitely long vectors; we have found no similar trick in the literature.

While Basis Pursuit gives itself an estimate of the number of nonzero coefficients, greedy algorithms are somewhat rudimentary in this respect. To overcome this difficulty, we take advantage of the inherent ordering of the columns selected by a greedy algorithm and employ information theoretic criteria (ITC) for selecting the number of relevant columns of the data matrix. This is a novelty in adaptive context, but ITC were used with greedy algorithms in [14], [16]; there is also a theoretical discussion in [17].

The contents of the paper is as follows. Section II presents the basic problem of minimizing an RLS criterion and Section III its solution with the OLS algorithm. Section IV is dedicated to our greedy sparse RLS; it presents in detail the update of the triangular factorization, using the assumption that a bound for the number of nonzero coefficients is known. Section V shows how ITC can be integrated in the algorithm such

The authors are with Department of Signal Processing, Tampere University of Technology, Finland. E-mails: firstname.lastname@tut.fi. B. Dumitrescu is also with Department of Automatic Control and Computers, "Politehnica" University of Bucharest, Romania. This work has been supported by a Tekes FiDiPro grant.

that it can work without additional information regarding the sparsity level. Section VI contains simulations for time-varying FIR channel identification, including comparisons with the algorithms from [7] and [8]. Section VII presents lossless coding results using a coder based on our GRLS algorithm.

A first form of our algorithm appeared in [18]. An extension to sliding window RLS (not treated at all here) can be found in [19].

II. The problem

The adaptive filtering problem consists of estimating, at each time moment $t \in \mathbb{N}$, the value of a (possibly time-varying) vector $\boldsymbol{x}_t \in \mathbb{R}^N$, given input vectors $\boldsymbol{a}_\tau \in \mathbb{R}^N$ and output d_τ, with $\tau \leq t$, based on the errors

$$e_\tau = d_\tau - \boldsymbol{a}_\tau^T \boldsymbol{x}_t. \tag{1}$$

A standard example is that of FIR channel identification, in which $\boldsymbol{x}_t$ is the vector—to be estimated—of filter coefficients and the filter input and output samples at time τ are u_τ and d_τ, respectively; for a filter with length N, the input vector contains consecutive (in time) inputs

$$\boldsymbol{a}_\tau = [u_\tau \ u_{\tau-1} \ \ldots \ u_{\tau-N+1}]^T. \tag{2}$$

Most currently, a least-squares criterion is employed and the errors are exponentially windowed, giving the RLS criterion

$$J_t(\boldsymbol{x}_t) = \sum_{\tau=0}^{t} \lambda^{t-\tau} e_\tau^2, \tag{3}$$

where λ is the forgetting factor. The same criterion can be written as the squared norm of the residual of the linear system $\boldsymbol{A}_t \boldsymbol{x}_t \approx \boldsymbol{d}_t$, in the form

$$J_t(\boldsymbol{x}_t) = \|\boldsymbol{d}_t - \boldsymbol{A}_t \boldsymbol{x}_t\|^2, \tag{4}$$

where $\boldsymbol{A}_t \in \mathbb{R}^{(t+1)\times N}$, $\boldsymbol{d}_t \in \mathbb{R}^{t+1}$; the i-th row of matrix $\boldsymbol{A}_t$, $i = 1 : t+1$, is $\sqrt{\lambda^{t-i+1}}\boldsymbol{a}_{i-1}^T$ and the i-th element of vector $\boldsymbol{d}_t$ is $\sqrt{\lambda^{t-i+1}} d_t$ (all indices of matrices and vectors start from 1), hence allowing the recursive description

$$\boldsymbol{A}_t = \begin{bmatrix} \sqrt{\lambda} \cdot \boldsymbol{A}_{t-1} \\ \boldsymbol{a}_t^T \end{bmatrix}, \quad \boldsymbol{d}_t = \begin{bmatrix} \sqrt{\lambda} \cdot \boldsymbol{d}_{t-1} \\ d_t \end{bmatrix}. \tag{5}$$

We investigate the problem of minimizing the RLS criterion (3) when it is known that the solution $\boldsymbol{x}_t$ is a sparse vector, i.e. the number of its nonzero elements, denoted $\|\boldsymbol{x}_t\|_0$, is small compared to its length N. For most of the presentation we will assume that a bound M is known such that $\|\boldsymbol{x}_t\|_0 \leq M$, but in Section V we will treat algorithmically the case where such a bound is not available. We name M-sparse a solution with at most M nonzeros.

III. Orthogonal least squares

The base of our adaptive approach is the orthogonal least square (OLS) algorithm [14], which belongs to the group of greedy algorithms for solving LS problems like minimizing (4), along with orthogonal matching pursuit (OMP) and its many variations. The algorithm computes the partial orthogonal triangularization with pivoting

$$\boldsymbol{A}_t \boldsymbol{P}_t = \boldsymbol{Q}_t \boldsymbol{R}_t, \tag{6}$$

where $\boldsymbol{Q}_t \in \mathbb{R}^{(t+1)\times(t+1)}$ is an orthogonal matrix, $\boldsymbol{R}_t \in \mathbb{R}^{(t+1)\times N}$ is upper triangular in its first M columns and $\boldsymbol{P}_t$ is a permutation matrix that brings the columns corresponding to the nonzero elements of $\boldsymbol{x}_t$ into the first M positions; these columns are named *active*, while the remaining $N - M$ are called *inactive*. The active columns are selected one by one, the k-th column being chosen such that, when appended to the already selected $k-1$ ones, produces the k-sparse solution with smallest LS residual. The algorithm is shown in Table I, with all computations performed in place.

We denote $\mathcal{A}$ and $\mathcal{I}$ the sets of indices of the active and inactive columns, respectively; for notation simplicity, we often drop the index t; we denote $\boldsymbol{x}_\mathcal{A}$ the vector of nonzero elements of $\boldsymbol{x}_t$; for a matrix, a similar notation is used for its restriction to the respective set of columns. The output of the algorithm is

$$\boldsymbol{Q}_t^T \boldsymbol{A}_t \boldsymbol{P}_t = \boldsymbol{R}_t = \begin{bmatrix} \boldsymbol{R}_\mathcal{A} & \boldsymbol{R}_\mathcal{I} \\ 0 & \boldsymbol{F} \end{bmatrix}, \quad \boldsymbol{Q}_t^T \boldsymbol{d}_t = \begin{bmatrix} \boldsymbol{b} \\ \boldsymbol{g} \end{bmatrix}. \tag{8}$$

The solution is obtained by solving an $M \times M$ triangular system:

$$\boldsymbol{x}_\mathcal{A} = \boldsymbol{R}_\mathcal{A}^{-1} \boldsymbol{b}, \quad \boldsymbol{x}_\mathcal{I} = 0. \tag{9}$$

The orthogonal matrix $\boldsymbol{Q}_t$ is not needed explicitly, it only has to be applied on $\boldsymbol{A}_t$ and $\boldsymbol{d}_t$; the permutation matrix $\boldsymbol{P}_t$ can be stored in the vector $\boldsymbol{p}$, which contains the order of the variables in $\boldsymbol{x}_t$. The distinctive feature of the algorithm is the greedy column selection in step 1.1 [14]: the chosen column has the largest relative projection on the residual, but the projection is computed using only the components of these vectors (columns and residual) that are orthogonal on the subspace of already selected $k-1$ columns (in contrast, OMP uses the full vectors). This choice minimizes the LS residual, given the previous $k-1$ columns; hence, OLS is also named the "greedy LS" algorithm. For the sake of completion, we give in the Appendix 1 a full justification of the choice (7).

Obviously, applying the OLS algorithm at each t has huge complexity, both in terms of computation and memory; the next section will present an approximate on-line version.

IV. Greedy sparse RLS algorithm

We present here an adaptive algorithm that is able to maintain information which, at each time t, is essentially equivalent to the OLS output from (8), but with a different permutation, which results from low-complexity considerations. Despite this difference, the simulations described in Section VI show that the adaptive algorithm gives results that are comparable or even better than those of OLS.

A. Principles of the algorithm

The algorithm, named Greedy RLS (GRLS), updates at each t only the first M rows of the data from (8), precisely the matrix $\boldsymbol{R} = [\boldsymbol{R}_\mathcal{A} \ \boldsymbol{R}_\mathcal{I}] \in \mathbb{R}^{M\times N}$ and the vector $\boldsymbol{b} \in \mathbb{R}^M$. This information is associated with the *present*, since its active part, $\boldsymbol{R}_\mathcal{A}$ and $\boldsymbol{b}$, is used for computing the current solution. The *past* information, represented by $\boldsymbol{F}$ and $\boldsymbol{g}$ in (8), cannot be dismissed, since the active columns may change, and it cannot be stored explicitly, since it grows indefinitely with t.

TABLE I
ORTHOGONAL LEAST-SQUARES ALGORITHM.

Algorithm OLS (Orthogonal least-squares)
Input. $\boldsymbol{A} \in \mathbb{R}^{(t+1)\times N}$, $\boldsymbol{d} \in \mathbb{R}^{t+1}$, M
0. $\boldsymbol{p} = [1\ 2\ \ldots\ N]$
1. for $k = 1 : M$
1.1. Choose "best" column:

$$j = \arg\max_{\ell=k:N} |\boldsymbol{A}(k:t+1,\ell)^T \boldsymbol{d}(k:t+1)|/\|\boldsymbol{A}(k:t+1,\ell)\| \tag{7}$$

1.2. Swap columns k and j of matrix $\boldsymbol{A}$. Swap $\boldsymbol{p}(k) \leftrightarrow \boldsymbol{p}(j)$.
1.3. Find Householder reflector $\boldsymbol{U}_k$ that zeros the k-th column of $\boldsymbol{A}$ below the diagonal.
1.4. Put $\boldsymbol{A} \leftarrow \boldsymbol{U}_k\boldsymbol{A}$, $\boldsymbol{d} \leftarrow \boldsymbol{U}_k\boldsymbol{d}$.
Output: active columns indices $\mathcal{A} = \boldsymbol{p}(1:M)$; the nonzero elements of the solution are obtained by solving the upper triangular system $\boldsymbol{A}(1:M, 1:M)\cdot \boldsymbol{x}_{\mathcal{A}} = \boldsymbol{d}(1:M)$.

We replace it with the fixed-size scalar products among the past vectors:

$$\boldsymbol{\Psi} = \boldsymbol{F}^T\boldsymbol{F} \in \mathbb{R}^{(N-M)\times(N-M)}, \qquad \boldsymbol{s} = \boldsymbol{F}^T\boldsymbol{g} \in \mathbb{R}^{N-M}. \tag{10}$$

Note that $\boldsymbol{\Psi}$ is a symmetric matrix, hence only its upper triangle should be stored; for brevity reasons, we will update the full matrix $\boldsymbol{\Psi}$ in the description of the algorithms but count only the relevant operations. Also, the permutation vector $\boldsymbol{p} \in \mathbb{N}^N$ is used for storing column order. This information, namely $\boldsymbol{R}$, $\boldsymbol{b}$, $\boldsymbol{p}$, $\boldsymbol{\Psi}$, $\boldsymbol{s}$, will be called *GRLS data*.

The GRLS data can be maintained with relatively low complexity, as we will show later, if some constraints on the possible permutations are imposed. Allowing arbitrary permutations, as dictated by the full OLS algorithm, is not possible due to the potential destruction of the triangular form of $\boldsymbol{R}$; imagine that the first column chosen by the OLS algorithm at time t is one of the inactive columns at time $t-1$; if this happens, permuting this column in the first position completely destroys the triangularity, in the sense that the triangularization of the permuted $\boldsymbol{R}$ has to be made as for a full matrix. To avoid this situation, we rely on the slow time-variability of the support of the solution and we replace the full search from (7) with the following strategy:

- for each position $k = 1 : M-1$ we let compete only columns k and $k+1$; so, the permutations that may occur are only between neighbors in the active set;
- for the last active position (M), we allow all columns (positions M to N) to compete, like in (7); so, at most a single inactive column may become active at time t.

Moreover, the complexity of the algorithm can be further decreased by performing the permutations only at time moments that are multiple of some small integer τ_0.

We describe next the most important parts of the algorithm.

B. Basic update

When the active set $\mathcal{A}$ is not changed and no permutations are performed (i.e. when $t \bmod \tau_0 \neq 0$), the necessary operations are those described in Table II and have the typical form of an orthogonal triangularization update. Actually, the same operations are employed at all times t, as explained later. In step 1, the current data vector is appended to the matrix $\boldsymbol{R}$, producing an arrow matrix; the triangular form is restored in the first M columns by zeroing, in step 2, the elements of the last row via Givens rotations. The remaining (transformed) elements of the new row are moved into the past in step 3 and then the row can be formally deleted in step 4.

The complexity of step 1 is about MN operations; an operation is either a multiplication or an addition. The retriangularization also needs $O(MN)$ operations, since at each k only rows k and $M+1$ are involved and the number of operations is proportional to $N-k$. The past update (12) is the most time consuming task, requiring about $3/2\cdot(N-M)^2$ operations.

C. Neighbor permutation

We describe now the permutation operations that involve two neighbor columns. Let us first notice that the test (7) can be applied as well before or after the *Basic_update* retriangularization; this is due to the fact that Givens rotations are orthogonal matrices and, when they multiply several vectors, they do not change the scalar products among these vectors. Hence, we work on the GRLS data as given by the *Basic_update* algorithm.

The permutation algorithm is shown in Table III. At iteration k, instead of choosing the best next column among all columns $k : N$, we restrain our search to columns k and $k+1$. The test (13) is the restrained version of (7). It is so simple because columns k and $k+1$ have only one and two, respectively, relevant nonzero elements (on and below row k). A permutation introduces only a subdiagonal element, that can be eliminated with a single Givens rotation; the situation is illustrated in Figure 1 for $k = 1$; the nonzero elements are represented by $\times$ and those modified by an operation by $*$; the figures show $\boldsymbol{R}$: (a) initially, (b) after the permutation and (c) after the application of the Givens rotation.

The number of operations is at most $O(MN)$, if all permutations are performed, and is small compared to the cost of the *Basic_update* algorithm.

D. Selection of last active column

After the neighbor permutations have been performed, the M-th and last active column is selected like in the OLS

TABLE II
BASIC UPDATE BY RE-TRIANGULARIZATION.

Algorithm Basic_update
Input. Current data: $\boldsymbol{a}_t \in \mathbb{R}^N$, $d_t \in \mathbb{R}$. GRLS data: $\boldsymbol{R} \in \mathbb{R}^{M\times N}$ (upper triangular), $\boldsymbol{b} \in \mathbb{R}^M$, $\boldsymbol{p} \in \mathbb{N}^N$, $\boldsymbol{\Psi}$, $\boldsymbol{s}$ as in (10).
1. Append current data to present information:

$$\boldsymbol{R} \leftarrow \begin{bmatrix} \sqrt{\lambda}\cdot\boldsymbol{R} \\ \boldsymbol{a}_t^T(\boldsymbol{p}) \end{bmatrix}, \qquad \boldsymbol{b} \leftarrow \begin{bmatrix} \sqrt{\lambda}\cdot\boldsymbol{b} \\ d_t \end{bmatrix}. \tag{11}$$

2. for $k = 1 : M$ (re-triangularization of arrow matrix)
2.1. Compute Givens rotation $\boldsymbol{G}_k$ that zeros $\boldsymbol{R}(M+1,k)$ using $\boldsymbol{R}(k,k)$
2.2. Put $\boldsymbol{R} \leftarrow \boldsymbol{G}_k\boldsymbol{R}$, $\boldsymbol{b} \leftarrow \boldsymbol{G}_k\boldsymbol{b}$
3. Denoting $\boldsymbol{\varphi}^T = \boldsymbol{R}(M+1, M+1:N)$, $\gamma = \boldsymbol{b}(M+1)$, update past:

$$\boldsymbol{\Psi} \leftarrow \lambda\boldsymbol{\Psi} + \boldsymbol{\varphi}\boldsymbol{\varphi}^T, \quad \boldsymbol{s} \leftarrow \lambda\boldsymbol{s} + \gamma\boldsymbol{\varphi} \tag{12}$$

4. Delete row $M+1$ of $\boldsymbol{R}$ and $\boldsymbol{b}$
Output: updated GRLS data.

TABLE III
NEIGHBOR PERMUTATIONS ALGORITHM AND RE-TRIANGULARIZATION.

Algorithm Neighbor_permutation
for $k = 1 : M-1$
1. if column $k+1$ is better than k, i.e. if

$$|\boldsymbol{b}(k)| < \frac{|\boldsymbol{R}(k,k+1)\boldsymbol{b}(k) + \boldsymbol{R}(k+1,k+1)\boldsymbol{b}(k+1)|}{\sqrt{\boldsymbol{R}(k,k+1)^2 + \boldsymbol{R}(k+1,k+1)^2}} \tag{13}$$

1.1. Swap columns k and $k+1$ of $\boldsymbol{R}$. Swap $\boldsymbol{p}(k) \leftrightarrow \boldsymbol{p}(k+1)$.
1.2. Compute Givens rotation $\hat{\boldsymbol{G}}_k$ that zeros $\boldsymbol{R}(k+1,k)$ using $\boldsymbol{R}(k,k)$
1.3. Put $\boldsymbol{R} \leftarrow \hat{\boldsymbol{G}}_k\boldsymbol{R}$, $\boldsymbol{b} \leftarrow \hat{\boldsymbol{G}}_k\boldsymbol{b}$

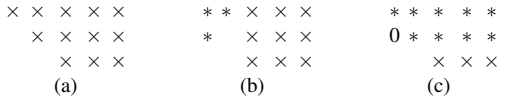


Fig. 1. Matrix $\boldsymbol{R}$ during the first iteration of algorithm *Neighbor_permutation* ($M = 3$, $N = 5$).

algorithm, using the criterion (7). However, for the inactive columns, found in positions $\ell = M+1 : N$, past information is needed; recalling the definition (10) of past scalar products, the quantity whose maximum is sought is

$$\alpha(\ell) = \frac{|\boldsymbol{R}(M,\ell)\boldsymbol{b}(M) + \boldsymbol{s}(\ell - M)|}{\sqrt{\boldsymbol{R}(M,\ell)^2 + \boldsymbol{\Psi}(\ell - M, \ell - M)}}. \tag{14}$$

(Remind that $\boldsymbol{\Psi}$ and $\boldsymbol{s}$ are defined only for the inactive columns, hence index $\ell - M$ corresponds to column ℓ.) Accordingly, the test from step 2 of the algorithm in Table IV decides the last active column; the left term of the inequality is similar to that from (13), since the M-th column has no past (is zero below the diagonal). If the M-th column is still the best, there is nothing more to be done. If the inactive column $j > M$ is better, it has to be permuted into the M-th position and then zeroed below the diagonal, using a Householder reflector. However, since the past of the column is not stored explicitly, this operation and the remainder of the algorithm require explanations.

At this point, the first $M-1$ rows of the matrix and vector from (8) are no longer relevant. We work only with vectors (columns) of unknown length m, having a known value on their first position, which belongs to the M-th row of (8), and other values stored indirectly through scalar products. If $\boldsymbol{c}$ is such a vector, we split it as

$$\boldsymbol{c} = \begin{bmatrix} c_1 \\ \tilde{\boldsymbol{c}} \end{bmatrix}, \tag{15}$$

where c_1 is a scalar. Let $\boldsymbol{c}$ be the new M-th column of $\boldsymbol{R}_t$ with the first $M-1$ elements removed; the Householder reflector that zeroes all its elements but the first is

$$\boldsymbol{U} = \boldsymbol{I} - (\boldsymbol{u}\boldsymbol{u}^T)/\beta, \tag{16}$$

where the vector $\boldsymbol{u} \in \mathbb{R}^m$ (split like in (15)) and the scalar β are defined by

$$\sigma = \operatorname{sgn}(\boldsymbol{c}(1))\sqrt{\sum_{i=1}^{m}\boldsymbol{c}(i)^2} = \operatorname{sgn}(c_1)\sqrt{c_1^2 + \tilde{\boldsymbol{c}}^T\tilde{\boldsymbol{c}}}, \tag{17a}$$

$$u_1 = c_1 + \sigma, \tag{17b}$$

$$\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{c}}, \tag{17c}$$

$$\beta = u_1\sigma. \tag{17d}$$

The first element of the transformed vector $\boldsymbol{U}\boldsymbol{c}$ is equal to $-\sigma$. The values σ, u_1 and β from (17a), (17b), (17d) can be computed since the scalar product $\tilde{\boldsymbol{c}}^T\tilde{\boldsymbol{c}}$ is stored. This is reflected in lines 2.4–2.7 of the *Last_active_column* algorithm. The vector $\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{c}}$ is not explicitly available, but it turns out that this is not necessary.

TABLE IV
ALGORITHM FOR SELECTION OF THE M-TH ACTIVE COLUMN AND RE-TRIANGULARIZATION.

Algorithm Last_active_column
1. Compute $j = \arg\max_{\ell=M+1:N} \alpha(\ell)$, see (14)
2. If $|\boldsymbol{b}(M)| < \alpha(j)$ (column j is the best and must enter the active set)
  2.1. Swap columns M and j of $\boldsymbol{R}$. Swap $\boldsymbol{p}(M) \leftrightarrow \boldsymbol{p}(j)$
  2.2. Save $\boldsymbol{z} = \boldsymbol{\Psi}(:, j-M)$. Set column and row $j-M$ of $\boldsymbol{\Psi}$ to zero.
  2.3. Save $\eta = \boldsymbol{s}(j-M)$, put $\boldsymbol{s}(j-M) \leftarrow 0$.
  2.4. $\sigma = \text{sgn}(\boldsymbol{R}(M,M))\sqrt{\boldsymbol{R}(M,M)^2 + \boldsymbol{z}(j-M)}$. Put $\boldsymbol{z}(j-M) \leftarrow 0$
  2.5. $\boldsymbol{u}(1) = \boldsymbol{R}(M,M) + \sigma$, $\beta = \boldsymbol{u}(1)\sigma$
  2.6. Save $\boldsymbol{\varphi}^T = \boldsymbol{R}(M, M+1:N)$, $\gamma = \boldsymbol{b}(M)$
  2.7. $\boldsymbol{R}(M,M) \leftarrow -\sigma$
  2.8. for $k = M+1 : N$
    2.8.1. $\theta = [\boldsymbol{u}(1)\boldsymbol{R}(M,k) + \boldsymbol{z}(k-M)]/\beta$
    2.8.2. $\boldsymbol{R}(M,k) \leftarrow \boldsymbol{R}(M,k) - \theta\boldsymbol{u}(1)$
  2.9. $\boldsymbol{b}(M) \leftarrow \boldsymbol{b}(M) - [\boldsymbol{u}(1)\boldsymbol{b}(M) + \eta]\boldsymbol{u}(1)/\beta$
  2.10. $\boldsymbol{s} \leftarrow \boldsymbol{s} + \gamma\boldsymbol{\varphi} - \boldsymbol{R}(M, M+1:N)\boldsymbol{b}(M)$
  2.11. $\boldsymbol{\Psi} \leftarrow \boldsymbol{\Psi} + \boldsymbol{\varphi}\boldsymbol{\varphi}^T - \boldsymbol{R}(M, M+1:N)^T\boldsymbol{R}(M, M+1:N)$

Let $\boldsymbol{v}$ be another column of $\boldsymbol{R}_t$ or $\boldsymbol{Q}_t^T\boldsymbol{d}_t$ from (8), with the first $M-1$ elements removed; to update the ortogonal triangularization, we have to multiply this column with the Householder reflector, i.e. to compute

$$\boldsymbol{w} = \boldsymbol{U}\boldsymbol{v} = \boldsymbol{v} - \frac{\boldsymbol{u}^T\boldsymbol{v}}{\beta}\boldsymbol{u}.$$

Splitting again like in (15) and taking into account that $\tilde{\boldsymbol{u}} = \tilde{\boldsymbol{c}}$, this amounts to

$$\theta = \boldsymbol{u}^T\boldsymbol{v}/\beta = (u_1 v_1 + \tilde{\boldsymbol{c}}^T\tilde{\boldsymbol{v}})/\beta, \tag{18a}$$

$$w_1 = v_1 - \theta u_1, \tag{18b}$$

$$\tilde{\boldsymbol{w}} = \tilde{\boldsymbol{v}} - \theta\tilde{\boldsymbol{c}}. \tag{18c}$$

Relations (18a), (18b) and (17b) show that w_1 can be computed, since the past scalar product $\tilde{\boldsymbol{c}}^T\tilde{\boldsymbol{v}}$ is available. Hence, the M-th row of the representation (8) can be computed, which is explicited in lines 2.8 and 2.9 of the *Last_active_column* algorithm.

It remains to update the past scalar products (10). Let now $\boldsymbol{y}$ and $\boldsymbol{v}$ be two distinct arbitrary columns of $\boldsymbol{R}_t$ or $\boldsymbol{Q}_t^T\boldsymbol{d}_t$ from (8), with the first $M-1$ elements removed; their multiplication with the reflector (16) produces the vectors $\boldsymbol{z} = \boldsymbol{U}\boldsymbol{y}$ and $\boldsymbol{w} = \boldsymbol{U}\boldsymbol{v}$. Since $\boldsymbol{U}$ is orthogonal, the scalar products $\boldsymbol{z}^T\boldsymbol{w}$ and $\boldsymbol{y}^T\boldsymbol{v}$ are equal. Using again the splitting (15), it results that

$$\tilde{\boldsymbol{z}}^T\tilde{\boldsymbol{w}} = \tilde{\boldsymbol{y}}^T\tilde{\boldsymbol{v}} + y_1 v_1 - z_1 w_1. \tag{19}$$

So, the scalar products can be updated using only the values on the M-th row of the representation (8), before and after the multiplication with the Householder reflector (16). This is implemented in lines 2.10 and 2.11 of the *Last_active_column* algorithm.

The complexity of the *Last_active_column* algorithm is dominated by the scalar product updates and is about $2(N-M)^2$ operations if an inactive column becomes active and $O(N-M)$ otherwise.

E. Algorithm overview

The structure of the overall GRLS algorithm is shown in Table V. If a regularization term $\delta\lambda^t\|\boldsymbol{x}_t\|_2^2$ is added to the criterion (3), the initialization $\boldsymbol{A}_{-1} = \sqrt{\delta}\boldsymbol{I}_N$, $\boldsymbol{b}_{-1} = \boldsymbol{0}$ is needed for the construction (5). This amounts to the initialization of $\boldsymbol{R}$ and $\boldsymbol{\Psi}$ as in Table V. The processing required at each time t is in accordance with the developments from the previous sections; it is understood that all the operations are performed on the GRLS data described in algorithm *Basic_update*. We note that the estimation $\boldsymbol{x}_t$ is typically used at time $t+1$, e.g. for prediction. The overall average number of operations is

$$\left(\frac{3}{2} + \frac{2}{\tau_0}\right)(N-M)^2 + O(MN), \tag{20}$$

which is typically less than the standard RLS complexity of $4N^2$ operations for a full filter of length N. The worst case complexity is about $7/2 \cdot (N-M)^2$, for $\tau_0 = 1$ and assuming that all column permutations are necessary; in this case, the constant multiplying the MN term in (20) is about 12; so, for reasonable sparse systems with $M = N/10$, the worst case complexity of GRLS is about the same as that of standard RLS.

F. Modifying the number of active columns

The GRLS algorithm from Table V has the drawback of assuming that a bound M for the number of nonzeros is known. If this bound is good, in particular equal to the true value, the behavior of GRLS is very good, as the experiments will show. However, a large M can degrade the performance, and this was suggested by some simulations shown in [18]. Since a good guess for M is usually not available, it may be desirable to vary M in the GRLS algorithm. Postponing for the next section the criteria that decide how M should vary, let us note here that decreasing or increasing M by 1 can be easily incorporated in the GRLS algorithm.

Decreasing M means simply deleting the M-th row of $\boldsymbol{R}$ and $\boldsymbol{b}$, but not before using it for updating the past information, similarly to steps 3 and 4 of *Basic_update*; this can be done after step 2.2.1 of the GRLS algorithm; when M is decreased, step 2.2.2 is no longer necessary. Increasing M can be done by running an algorithm similar with *Last_active_column* for the selection of the $(M+1)$-th active column; for keeping worst case complexity under control, this can be done at times t that follow a multiple of τ_0. Since the formal description of these

TABLE V
OVERVIEW OF THE GRLS ALGORITHM.

Algorithm GRLS
Parameters: M—sparsity level, τ_0—permutation update lag, λ—forgetting factor, δ—regularization factor
1. Initialization: $\boldsymbol{R}(:, 1:M) = \sqrt{\delta}\boldsymbol{I}$, $\boldsymbol{b} = 0$, $\boldsymbol{\Psi} = \delta\boldsymbol{I}$, $\boldsymbol{s} = 0$, $\boldsymbol{p} = [1\ 2\ \ldots\ N]$.
2. At each time t
  2.1. Get current data $\boldsymbol{a}_t \in \mathbb{R}^N$, $d_t \in \mathbb{R}$ and run *Basic_update*
  2.2. If $t \bmod \tau_0 = 0$
    2.2.1. Run *Algorithm Neighbor_permutation*
    2.2.2. Run *Last_active_column*
  2.3. Estimation: the nonzero elements of $\boldsymbol{x}_t$ are obtained by solving the upper triangular system $\boldsymbol{R}(:, 1:M) \cdot \boldsymbol{x}_{\mathcal{A}} = \boldsymbol{b}$, with $\mathcal{A} = \boldsymbol{p}(1:M)$ the set of active column indices

operations takes too much space and is rather straightforward, the reader interested in details can consult our programs at http://www.cs.tut.fi/~bogdand/Software/grls.zip.

V. SPARSITY LEVEL SELECTION USING INFORMATION THEORETIC CRITERIA

Greedy algorithms like GRLS order the elements of the solution $\boldsymbol{x}_t$ according to their (approximated) relevance. If $L_t = \|\boldsymbol{x}_t\|_0$ is the true number of nonzeros of the solution and if the parameter M is large enough, i.e. $M \geq L_t$, it is reasonable to assume that for slowly time-varying processes and after a sufficiently long transitory regime, the first L_t elements of the active set given by the GRLS algorithm are the true nonzero positions. Assuming that an estimated value $\hat{L}_t \leq M$ is available for L_t, there is no problem in computing an $\hat{L}_t$-sparse solution in step 2.1 of the GRLS algorithm, instead of an M-sparse one. The only question is how to estimate L_t reliably.

Information theoretic criteria (ITC), as traditionally used for order selection (like e.g. for AR models), are a perfect tool in this context; assuming that the model coefficients are ordered, the only decision is on how many coefficients to consider. A recent discussion on ITC for forgetting factor least-squares algorithms can be found in [20]; basic work on this topic was done in [21], [22], [23]. Among the several ITC, with possible parameterized extensions, we have found two criteria giving good results in the GRLS context. (We do not exclude the possibility that other criteria give better results.)

A. Bayesian Information Criterion (BIC)

The Bayesian Information Criterion [24] uses the squared norm (4) of the residual in the form

$$J_t(\hat{\boldsymbol{x}}_{t,k}) = \|\boldsymbol{d}_t - \boldsymbol{A}_t\hat{\boldsymbol{x}}_{t,k}\|^2, \tag{21}$$

where $\hat{\boldsymbol{x}}_{t,k}$ is the k-sparse solution at time t, as computed by the GRLS algorithm, with k taking values between 1 and M. BIC combines the squared norm (21) of the residual with the number of samples, i.e. the size of $\boldsymbol{d}_t$, taking into account the forgetting factor; the *effective* number of samples is defined as

$$n_{ef,t} = \sum_{\tau=0}^{t} \lambda^{\tau} = 1 + \lambda n_{ef,t-1}. \tag{22}$$

Using these ingredients, the BIC criterion has the expression

$$\mathrm{B}IC_t(k) = n_{ef,t} \ln J_t(\hat{\boldsymbol{x}}_{t,k}) + (k+1) \ln n_{ef,t}. \tag{23}$$

The BIC criterion can be easily computed in the GRLS context, taking advantage of the properties of the orthogonal triangularization (8). The residual of a k-sparse solution has the same norm as the vector $\boldsymbol{Q}_t^T\boldsymbol{d}_t$ from (8) with the first k elements removed, e.g. the residual of a M-sparse solution has the squared norm equal to $\|\boldsymbol{g}\|^2$; see (33) in Appendix 1. The squared norm of the full vector, $\|\boldsymbol{Q}_t^T\boldsymbol{d}_t\|^2 = \|\boldsymbol{d}_t\|^2$, can be computed with the recursion in time

$$\nu_t = d_t^2 + \lambda\nu_{t-1}, \quad \nu_0 = d_0. \tag{24}$$

At time t, the squared norm (21) of the residual can be computed with the recursion (in "order")

$$J_t(\hat{\boldsymbol{x}}_{t,k}) = J_t(\hat{\boldsymbol{x}}_{t,k-1}) - \boldsymbol{b}(k)^2, \quad J_t(\hat{\boldsymbol{x}}_{t,0}) = \nu_t, \tag{25}$$

where $\boldsymbol{b}$ is the vector from (8), as updated by the GRLS algorithm at time t. We note that the number of operations required by the recursions (22), (24) and (25) is insignificant in the GRLS algorithm, being only $O(M)$.

B. Predictive Least Squares (PLS)

The PLS criterion [22] is

$$\mathrm{P}LS_t(k) = \sum_{\tau=0}^{t} \lambda^{t-\tau} e_\tau(k)^2, \tag{26}$$

where $e_\tau(k)$ is the a priori estimation error at time τ produced by the k-sparse solution $\hat{\boldsymbol{x}}_{\tau-1,k}$, i.e. $e_\tau(k) = d_\tau - \boldsymbol{a}_\tau^T\hat{\boldsymbol{x}}_{\tau-1,k}$ (compare with (1)). The criterion (26) can be updated easily by

$$\mathrm{P}LS_t(k) = \lambda \cdot \mathrm{P}LS_{t-1}(k) + e_t(k)^2. \tag{27}$$

The computation of the a priori errors $e_t(k)$, $k = 1:M$, can be implemented through a recursion shown in Appendix 2 and requires $O(M^2)$ operations, hence it is still relatively cheap in the context of the GRLS algorithm.

C. ITC in the GRLS algorithm

We describe here two variants of GRLS in conjuction with ITC use. In the first, the maximum sparsity level M is *fixed*. Its value should be large enough to cover all practical cases of sparse filters occurring in the application at hand, but as small as possible for complexity reasons. The ITC criteria (23) or (26) are computed before step 2.3 of GRLS. Their minimal value is attained for

$$\hat{L}_t = \arg\min_{k=1:M} \mathrm{B}IC_t(k) \text{ or } \mathrm{P}LS_t(k). \tag{28}$$

Then, in step 2.3 of GRLS, the nonzero elements of the solution are computed by solving the triangular system $\boldsymbol{R}(1:\hat{L}_t, 1:\hat{L}_t)\cdot\boldsymbol{x}_{\mathcal{A}} = \boldsymbol{b}(1:\hat{L}_t)$, with $\mathcal{A} = \boldsymbol{p}(1:\hat{L}_t)$.

The second variant of GRLS employs a *variable* M, modified using a parameter Δ. This parameter has the significance of a safety cushion which ensures that there are enough candidates for finding the best number of nonzeros (28). More precisely, the value of M is modified with the objective of making it equal to $\hat{L}_t + \Delta$. Since, as explained in section IV-F, it is more natural to modify the value of M only by 1, the following simple approach is used. After computing (28), the value of M is increased by 1 if $M < \hat{L}_t + \Delta$, decreased by 1 if $M > \hat{L}_t + \Delta$ and preserved if $M = \hat{L}_t + \Delta$. The BIC criterion can be computed as for fixed M, since the only required past information is (24). However, PLS needs the past a priori errors for all $k = 1 : M$; after M increases, the past values of $e_\tau(M)$ are no longer available for some $\tau < t$. In this situation, lacking other information, we put $\mathrm{P}LS_t(M) = \mathrm{P}LS_t(M-1)$; if M does not decrease, the PLS will tend to its true value, due to the effect of the forgetting factor. Also, an implicit approximation is made at neighbor permutations, where the past PLS values are preserved, although the new models are actually different. Despite these approximations, PLS works well in practice, as shown in the next sections.

VI. Simulations

The performance of the algorithms was tested for a sparse FIR channel identification problem (1), (2) with the true number of nonzero coefficients $L_t = 5$ and $L_t = 10$ and a filter length $N = 200$. The positions of the nonzero coefficients are randomly chosen and the coefficient variation is described by three different models. Most of the simulations were performed for a coefficient variation according to the sinusoidal model

$$\boldsymbol{x}_t(i) = a_i \cos(2\pi f T_s t + \alpha_i), \tag{29}$$

where i is the position of a nonzero coefficient in the vector $\boldsymbol{x}_t$ and the amplitude a_i and phase α_i are uniformly distributed in $[0.05, 1]$ and $[0, 2\pi]$, respectively. The product fT_s, where f is the frequency and T_s is the sampling interval, determines the variation speed of the channel. We have also used the Gauss-Markov model

$$\boldsymbol{x}_t(i) = \beta\boldsymbol{x}_{t-1}(i) + \boldsymbol{w}_t(i), \tag{30}$$

where $\boldsymbol{x}_0(i)$ is generated independently with the distribution $\mathcal{N}(0,1)$ and $\boldsymbol{w}_t(i)$ is generated independently with $\mathcal{N}(0, 1-\beta^2)$. The variation speed is controlled by the parameter $\beta \in (0,1)$. The third model is a sparse multipath channel generated using Jakes' sum of sinusoids model. Each coefficient $\boldsymbol{x}_t(i)$ is a sum of 10 sinusoids with frequencies $f_n = f\cos\vartheta_n$, where ϑ_n is uniformly distributed in $[0, 2\pi]$. Similarly to (29), the variation speed is determined by the product fT_s. In all cases the coefficients vector is normed such that its average squared norm over all t is 1.

The input samples, associated with the input vectors (2), are normally distributed according to $\mathcal{N}(0,1)$, while the outputs are corrupted by an additive Gaussian noise with $\sigma^2 = 0.01$. The performance of the algorithms is measured in terms of the coefficient mean squared error

$$\mathrm{MSE}_t = E\{||\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t||_2^2\}, \tag{31}$$

where $\boldsymbol{x}_t$ contains the actual values of the coefficients and $\hat{\boldsymbol{x}}_t$ their estimates. MSE_t is evaluated by averaging over 1000 runs at a given t, for each model and each algorithm, and the average MSE is defined as the average of MSE_t (31) over the last 100 values of t.

The tested algorithms are named as follows: RLS, the standard algorithm that computes the full filter of length N; RLS-SP, a sparsity informed RLS algorithm, i.e. having prior knowledge of the nonzero coefficients number and positions; GRLS, the algorithm presented in Table V, running with $M = L_t$, i.e. knowing only the true number of nonzeros; OLS, the batch algorithm run at each time t on all accumulated data, knowing L_t; GRLS-BIC-F, the algorithm that uses the BIC criterion for estimating the number of nonzeros and employs a fixed upper bound M; GRLS-BIC-V, the algorithm that uses the BIC criterion for estimating the number of nonzeros and employs a variable upper bound using the parameter Δ, as described in Section V-C; GRLS-PLS-F and GRLS-PLS-V, similar with the above two algorithms, but using the PLS criterion; SPARLS, the algorithm presented in [8]; OCCD-TNWL, the best (but also the most complex) among the algorithms presented in [7].

All the algorithms based on GRLS use $\tau_0 = 2$, i.e. the permutations are performed only at every other time sample. The configuration parameter γ for SPARLS is optimized by grid search with the step 5 for each of the simulation conditions, if not otherwise specified; the other parameter is $\alpha = \frac{\sigma}{2}\sqrt{N}$, chosen as recommended in [8]. For OCCD-TNWL, the penalty term of the filter ℓ_1-norm regulatization term in the RLS criterion was taken as $\lambda_t = \sqrt{2\sigma^2\log(N)}\sqrt{\sum_{i=1}^{t}\lambda^{2(t-i)}}$; other parameters are $\mu_t = \lambda_t/\sum_{i=1}^{t}\lambda^{t-i}$ and $a = 3.7$; these choices are the same as in [7]. The true value of σ was provided to both SPARLS and OCCD-TNWL.

The OCCD-TWNL algorithm uses an inner RLS to provide weights for its l_1-norm regularization solver that generates the sparse estimator, thus the performance of RLS can influence the resulting solution. Especially for fast varying channels, the RLS algorithm behaves poorly if we use a small forgetting factor. In order to asses the impact the RLS performance has, we performed tests (as suggested by a reviewer) in which the inner RLS used in OCCD-TNWL had a different forgetting factor $\lambda_{\mathrm{RLS}_{\mathrm{opt}}}$, chosen such that the RLS algorithm would produce the smallest MSE. This forgetting factor was found using a grid search with a step size 0.0025. We name OCCD-TNWL-OPT the resulting algorithm (and RLS-OPT the underlying RLS).

In Table VI we present the average MSE for the algorithms presented above, together with the parameters used for the configuration of the algorithms, in the case where $L_t = 5$. The forgetting factor λ was chosen upon the principle that it should decrease as the variation speed increases, but otherwise not optimized; we will present later a discussion on the effect

of λ. The best MSE value of an algorithm, excluding RLS-SP, GRLS and OLS (which have the benefit of prior information), for a set of simulation conditions, is marked with bold digits. For the values marked with * only the stable tests were considered because the algorithm OCCD-TNWL was unstable for a small number of tests.

Table VII presents information similar to that from Table VI, but for $L_t = 10$. Due to the relatively similar performance, we give in Table VIII only a shorter set of results for the Gauss-Markov and Jakes models.

We conclude from the simulations that, excepting the constant channel case, where OCCD-TNWL gives the best MSE, our algorithms are superior to those from [7] and [8]; for varying channels, OCCD-TNWL-OPT is best in a single case; in all the others, a GRLS algorithm is the best. The gap widens as the variation speed increases. For smoothly varying coefficients, like in the sinusoidal and Jakes models, the GRLS family is clearly better than the competitors. For the Gauss-Markov model, where the variation has an impulsive component, the gap between GRLS and OCCD-TNWL-OPT is relatively narrow.

Remarkably, the use of ITC puts the results on par with those given by the GRLS algorithm that knows the true number of nonzeros; for varying channels, a possible explanations for the better results given by BIC or PLS is that they are able to adapt to the situations when some coefficients become nearly zero, while GRLS is not. It is hard to point out a clear winner, but BIC seems to give slightly better results than PLS for variable M, while the situation is reversed for fixed M. Higher values of M and Δ appear to be better for high variation speeds. From these and other experiments, we estimate that the GRLS family of algorithms seems robust to the values of these parameters, as long as they are not too small. Taking $\Delta = 5$ appears as a safe choice for the variable M algorithms. If M is fixed, then a good choice, as long as only MSE is concerned, would be to take $M \geq \max_t L_t + 5$; of course, an estimate for $\max_t L_t$ should be available.

The poor behavior of the batch OLS algorithm for quickly varying channels (associated with a relatively low forgetting factor) is somewhat surprising. One would expect OLS and GRLS have similar performance. The explanation is that, in case of adverse noise, OLS has occasionally large errors due to the selection in the first positions of a wrong column, which completely spoils the support. The wrong support is quickly corrected in the next moments, since OLS can change the whole support at each time. On the contrary, GRLS reacts slower due to the neighbor exchange mechanism; the wrong columns enter in the last position of the active set and do not have time to advance further, hence the errors they cause are small. So, by tempering the support changes, GRLS is inherently more robust than the (sometimes) excessively greedy OLS.

To have an idea of the convergence speed of the algorithms, we present in Figure 2 the evolution in time of MSE_t. The channel is variable according to a sinusoidal model with $fT_s = 0.001$. At time $t = 500$, three of the $L_t = 5$ coefficients suddenly and randomly change positions. The forgetting factor is $\lambda = 0.92$ for all algorithms; the optimal forgetting factor $\lambda_{\text{RLS}_{\text{opt}}} = 0.9825$ was used for both RLS-OPT and the internal RLS of OCCD-TNWL-OPT. The left figure presents the evolution of only one BIC algorithm and the right figure allows a comparison of the effects of M and Δ (GRLS and RLS-SP are shown for reference). The algorithms using the PLS criterion behave similarly. We remark that a large value of M for the fixed case causes a slower convergence speed, but the fixed M algorithm with relatively low M (still much larger than L_t) and the variable M algorithm have convergence speeds comparable to OCCD-TNWL and better than SPARLS.

A series of tests were also performed for different filter lengths, but keeping $L_t = 5$, to assess the performance as a function of N. Figure 3 shows the average MSE for $\lambda = 0.92$ and $fT_s = 0.001$; the PLS algorithms (not shown) behave similarly with BIC ones. We note that the GRLS algorithms perform robustly, the MSE increasing moderately with N; the BIC and PLS versions give results that are similar and often better than those of GRLS for all tested values of N. In contrast, the performance of OCCD-TNWL, very good for low N, deteriorates more abruptly.

Similarly, in Figure 4, we present average MSE values for varying number L_t of nonzero coefficients and fixed filter length N, again for $\lambda = 0.92$ and $fT_s = 0.001$. As L_t increases, the performance of all sparse algorithms deteriorates, SPARLS behaving the most robustly. However, GRLS is still better in this L_t range, which is high enough for barely qualifying the model as sparse at the upper end. The BIC and PLS algorithms have now different behaviors, with PLS being clearly better for large L_t. Only the PLS algorithm with fixed $M = L_t + 10$ is now constantly better than the basic GRLS.

We have also varied the forgetting factor λ, keeping $L_t = 5$, $N = 200$, $fT_s = 0.001$. The obtained average MSE are shown in Figure 5; again, the PLS algorithms give results similar with BIC. The optimal λ is different for the different algorithms (in particular, we note that the value $\lambda = 0.92$ chosen for the previous simulations is not optimal for any algorithm); GRLS algorithms attain their best performance for smaller forgetting factors than SPARLS and OCCD-TNWL. However, unless the forgetting factor is near 1 and all algorithms behave similarly (but rather poorly), our algorithms are superior, for each λ and also in the best case.

The above simulation results show that our GRLS algorithms are almost always better than SPARLS and better than OCCD-TNWL for time-varying channels. The complexity of SPARLS is estimated at 20-30% of that of RLS in [8], by effectively counting the operations performed by the running algorithm. For $\tau_0 = 2$, we estimate based on (20) that our algorithms have about 60-80% of RLS complexity, hence they are about three times slower than SPARLS. OCCD-TNWL has a clearly higher complexity, since it is built on top of the full RLS; we estimate its complexity at 150% of that of RLS, so GRLS is about twice faster. However, a definitive comparison between the algorithms should be based on execution times of professional implementations. The robustness of the GLRS family to parameter variations and different channel conditions also recommends it as a first choice algorithm in FIR channel identification.

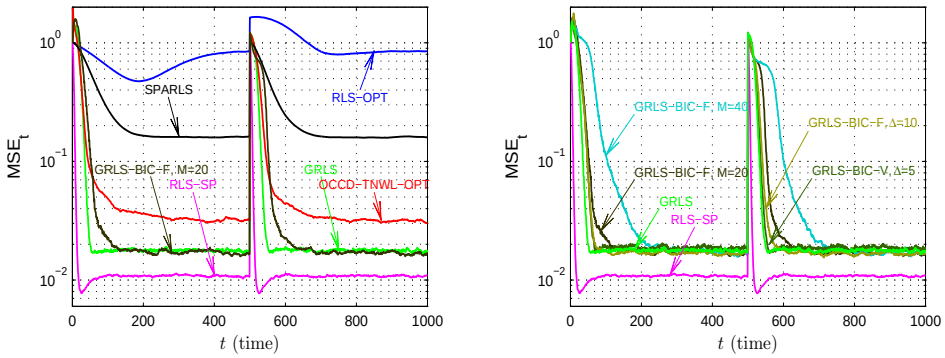


Fig. 2. MSE_t variation as a function of time t, for $L_t = 5$.

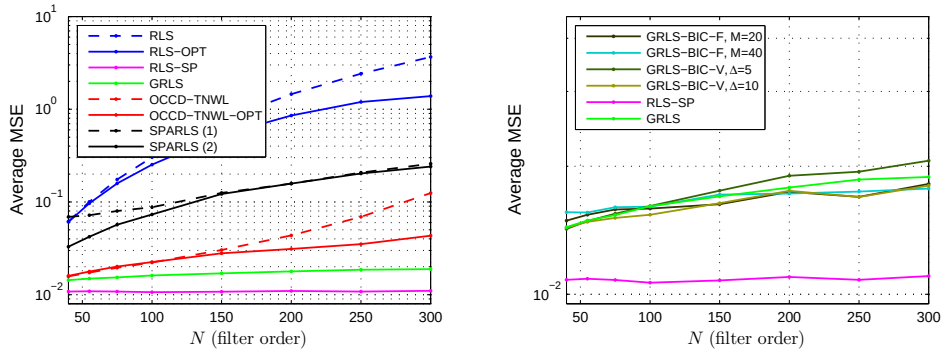


Fig. 3. Average MSE as a function of N, for $L_t = 5$, $\lambda = 0.92$ and $fT_s = 0.001$. For the SPARLS algorithm, the curve marked SPARLS (1) uses the configuration parameters chosen for $N = 200$ in all other tests, while for SPARLS (2) the configuration parameters were tuned for each test case.

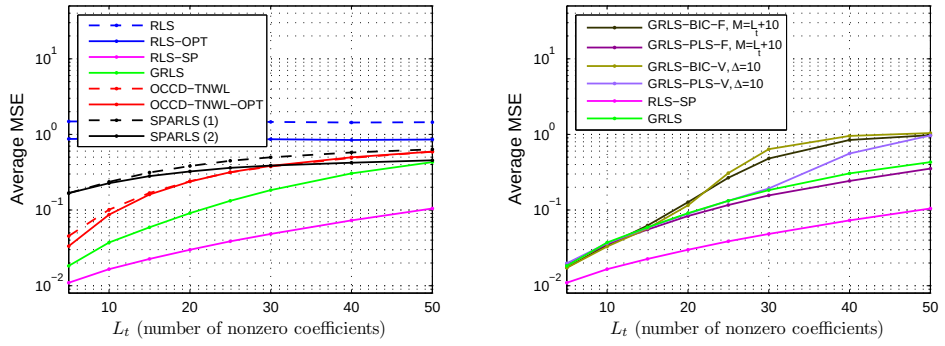


Fig. 4. Average MSE as a function of L_t, for $N = 200$, $\lambda = 0.92$ and $fT_s = 0.001$.

TABLE VI
AVERAGE MSE VALUES FOR FILTER ORDER $N = 200$, WITH TRUE NUMBER OF NON-ZEROS $L_t = 5$ AND SINUSOIDAL COEFFICIENT VARIATION.

	Variable						Constant	
fT_s	0.005	0.002	0.001	0.0005	0.0002	0.0001	0	0
λ	0.86	0.90	0.92	0.94	0.96	0.98	0.99	0.995
RLS	13.06	5.3012	1.4565	0.36427	0.09774	0.04734	0.012360	0.005814
RLS-SP	0.089	0.0267	0.0110	0.00518	0.00246	0.00306	0.000260	0.000124
GRLS	0.277	0.0501	0.0178	0.00785	0.00343	0.00343	0.000260	0.000124
OLS	0.475	0.1130	0.0310	0.00923	0.00350	0.00344	0.000260	0.000124
GRLS-BIC-F								
$M = 20$	0.211	0.0507	0.0174	0.00783	0.00334	0.00360	0.000450	0.000188
$M = 40$	**0.194**	0.0507	**0.0173**	0.00775	0.00353	0.00358	0.000391	0.000182
GRLS-BIC-V								
$\Delta = 5$	0.280	0.0540	0.0190	0.00881	0.00427	0.00384	0.000506	0.000190
$\Delta = 10$	0.224	**0.0485**	0.0175	0.00763	0.00352	0.00364	0.000475	0.000189
GRLS-PLS-F								
$M = 20$	0.213	0.0511	0.0189	0.00853	0.00356	0.00357	0.000324	0.000148
$M = 40$	**0.194**	0.0500	0.0180	0.00824	0.00374	0.00373	0.000336	0.000150
GRLS-PLS-V								
$\Delta = 5$	0.285	0.0569	0.0187	**0.00762**	**0.00330**	**0.00340**	0.000317	0.000147
$\Delta = 10$	0.242	0.0570	0.0196	0.00845	0.00341	0.00346	0.000322	0.000149
SPARLS	0.824	0.4417	0.1578	0.04225	0.01120	0.00767	0.001267	0.000624
γ	215	170	110	75	50	75	95	150
OCCD-TNWL	2.72*	0.4802	0.0436	0.01231	0.00447	0.00372	0.000304	0.000127
OCCD-TNWL-OPT	0.498	0.1491	0.0311	0.01248	0.00471	0.00386	**0.000298**	**0.000126**
RLS-OPT	1.325	1.2808	0.8544	0.25247	0.06968	0.04406	0.002971	0.002974
$\lambda_{\mathrm{RLS_{opt}}}$	0.9975	0.9975	0.9825	0.9800	0.9850	0.9900	0.9975	0.9975

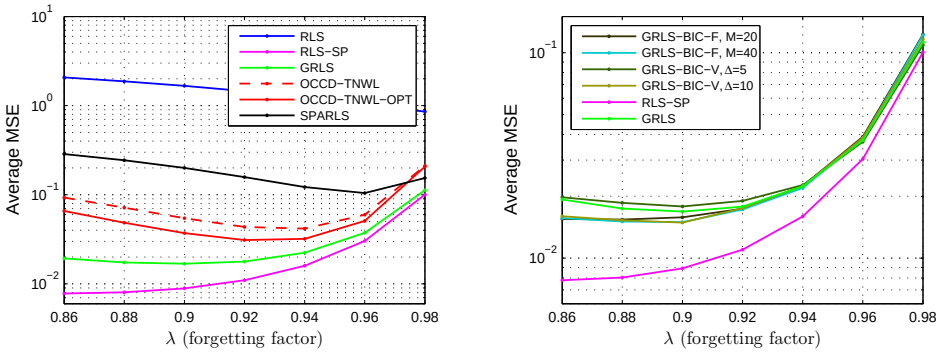


Fig. 5. Average MSE as a function of the forgetting factor λ, for $L_t = 5$, $N = 200$ and $fT_s = 0.001$.

VII. APPLICATION TO LOSSLESS AUDIO CODING

The GRLS family of algorithms has been applied to lossless audio coding by computing linear predictors that use the previous N samples for estimating the current sample. The same prediction algorithm is run at the encoder and the decoder. At the encoder, the prediction residuals are entropy coded using the Golomb-Rice method and sent to the decoder. The decoder restores the residuals and adds them to the predicted values, thus retrieving the original samples.

To test the method, we used one minute extracts from 13 pieces of music. They were all 16-bit audio tracks sampled at 44100 Hz. The material included pop music with human voice, clean guitar and drums ('conga', 'pinkmoon', 'pocket', 'ordeal'), loud rock music ('forty', 'mummified'), jazz ('sowhat') and classical music with full orchestra ('finlandia', 'morgen'), a few instruments ('prelude', 'sunflower', 'weep') or choir ('de_ore_leonis').

Each extract was divided into 60 frames (each holding 44100 samples and encoded independently), to enable a more arbitrary beginning point for decoding. The GRLS algorithms were rerun with no prior information at the beginning of each frame. Golomb-Rice was utilized to code the residual values in blocks of 900 samples. For each block, the optimal number of bits used for the remainder was sought by trying out each value between 0 and 15. The forgetting factor is $\lambda = 0.9965$ and the permutation update lag is $\tau_0 = 3$. The BIC criterion was modified by taking the effective number of samples as $n_{ef} = 4/(1-\lambda)$, which is 4 times higher than the asymptotic value given by (22); this is justified by the compression results; we mention that a factor of 2 was advocated in [21]. BIC and PLS criteria have been used as in the algorithms GRLS-BIC-F and GRLS-PLS-F from the previous section.

TABLE VII
AVERAGE MSE VALUES FOR FILTER ORDER $N = 200$, WITH TRUE NUMBER OF NON-ZEROS $L_t = 10$ AND SINUSOIDAL COEFFICIENT VARIATION.

	Variable						Constant	
fT_s	0.005	0.002	0.001	0.0005	0.0002	0.0001	0	0
λ	0.86	0.90	0.92	0.94	0.96	0.98	0.99	0.995
RLS	12.68	5.2811	1.4596	0.3666	0.09333	0.04312	0.012303	0.005795
RLS-SP	0.144	0.0393	0.0159	0.0077	0.00370	0.00327	0.000521	0.000256
GRLS	0.760	0.1128	0.0357	0.0153	0.00672	0.00428	0.000521	0.000256
OLS	1.014	0.3887	0.1425	0.03237	0.00743	0.00431	0.000521	0.000256
GRLS-BIC-F								
$M = 20$	0.489	0.1082	0.0336	0.0132	0.00556	0.00412	0.000749	0.000324
$M = 40$	0.411	0.1029	0.0349	0.0140	0.00559	0.00409	0.000671	0.000317
GRLS-BIC-V								
$\Delta = 5$	0.699	0.1100	0.0328	0.0140	0.00643	0.00430	0.000777	0.000326
$\Delta = 10$	0.571	0.1036	**0.0322**	**0.0130**	0.00557	0.00412	0.000747	0.000324
GRLS-PLS-F								
$M = 20$	0.471	0.1035	0.0351	0.0141	0.00561	0.00399	0.000588	0.000289
$M = 40$	**0.383**	**0.0929**	0.0340	0.0144	0.00595	0.00423	0.000598	0.000290
GRLS-PLS-V								
$\Delta = 5$	0.701	0.1147	0.0328	0.0131	**0.00552**	**0.00395**	**0.000584**	0.000287
$\Delta = 10$	0.569	0.1124	0.0351	0.0139	0.00553	0.00399	0.000591	0.000288
SPARLS	0.858	0.5223	0.2257	0.0657	0.01545	0.00894	0.002097	0.001052
γ	165	130	85	55	30	50	75	125
OCCD-TNWL	2.90*	0.7613	0.0984	0.0266	0.00937	0.00488	0.000631	0.000265
OCCD-TNWL-OPT	0.763	0.3866	0.0840	0.02788	0.00984	0.00511	0.000614	**0.000263**
RLS-OPT	1.325	1.2805	0.8565	0.25141	0.06496	0.03797	0.002969	0.002966
$\lambda_{\mathrm{RLS_{opt}}}$	0.9975	0.9975	0.9825	0.9800	0.9850	0.9900	0.9975	0.9975

TABLE VIII
AVERAGE MSE VALUES FOR FILTER ORDER $N = 200$, WITH TRUE NUMBER OF NON-ZEROS $L_t = 5$, RESPECTIVELY $L_t = 10$, AND COEFFICIENT VARIATION ACCORDING TO THE GAUSS-MARKOV AND JAKES MODELS.

	Gauss-Markov Model					Jakes Model			
	$L_t = 5$		$L_t = 10$			$L_t = 5$		$L_t = 10$	
β	0.99	0.999	0.99	0.999	fT_s	0.001	0.0005	0.001	0.0005
λ	0.9	0.95	0.9	0.95	λ	0.92	0.94	0.92	0.94
RLS	4.941	0.6355	4.899	0.5857		0.8880	0.24613	0.8564	0.23998
RLS-SP	0.144	0.0303	0.188	0.0346		0.0071	0.00368	0.0112	0.00595
GRLS	0.289	0.0386	0.535	0.0552		0.0114	0.00556	0.0237	0.01161
OLS	0.383	0.0395	0.763	0.0650		0.0283	0.00832	0.1560	0.03225
GRLS-BIC-F									
$M = 20$	0.269	0.0391	0.511	0.0513		0.0115	0.00540	0.0215	0.00982
$M = 40$	0.267	0.0400	0.509	0.0555		0.0113	0.00548	0.0239	0.01054
GRLS-BIC-V									
$\Delta = 5$	0.269	0.0388	0.494	0.0501		0.0128	0.00671	0.0219	0.01107
$\Delta = 10$	**0.260**	**0.0383**	0.484	**0.0498**		**0.0112**	0.00542	**0.0208**	0.00971
GRLS-PLS-F									
$M = 20$	0.282	0.0421	0.495	0.0534		0.0128	0.00601	0.0227	0.01030
$M = 40$	0.279	0.0419	0.482	0.0559		0.0124	0.00613	0.0240	0.01128
GRLS-PLS-V									
$\Delta = 5$	0.282	0.0390	0.493	0.0504		0.0117	**0.00535**	0.0212	**0.00968**
$\Delta = 10$	0.295	0.0412	0.497	0.0533		0.0125	0.00574	0.0228	0.01025
SPARLS	0.554	0.0997	0.611	0.1175		0.1126	0.03173	0.1766	0.05071
γ	145	105	100	60		90	60	65	40
OCCD-TNWL	0.723	0.0468	0.863	0.0579		0.0257	0.00876	0.0610*	0.02060
OCCD-TNWL-OPT	0.290	0.0407	**0.456**	0.0544		0.0221	0.00908	0.0602	0.02192
RLS-OPT	1.048	0.3300	1.058	0.3025		0.5154	0.16287	0.5074	0.15757
$\lambda_{\mathrm{RLS_{opt}}}$	0.9975	0.9925	0.9975	0.9925		0.9825	0.9800	0.9800	0.9825

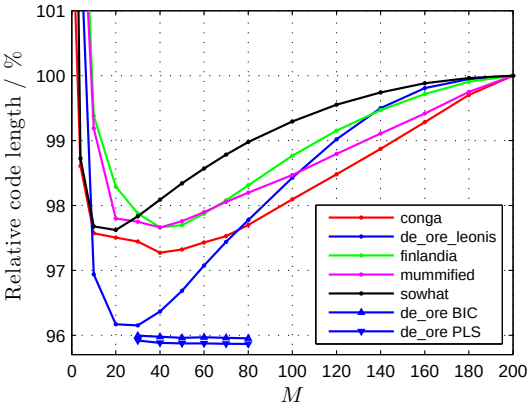


Fig. 6. Relative code lengths with respect to a full predictor of length $N = 200$.

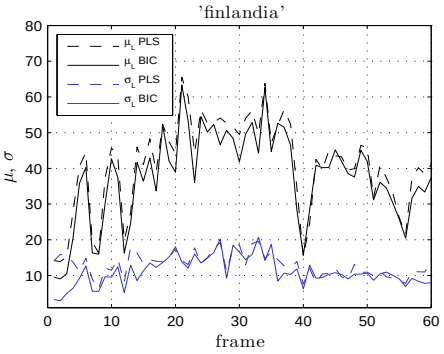


Fig. 7. Average selected $\hat{L}_t$ based on ITC for 'finlandia'. The blue curves show the standard deviation.

A. Performance of sparse predictors

We assess first the preformance of sparse predictors in comparison with full ones computed with RLS. We take as reference the results obtained with a full predictor of length $N = 200$; for other predictors we report the relative code length $\text{filesize}_{\text{predictor}}/\text{filesize}_{\text{full},N=200} \cdot 100\%$, where filesize is the length of the compressed file (so, smaller values are better). Figure 6 shows typical relative code lengths, other test pieces being quite similar to 'finlandia' and 'mummified'. When GRLS with fixed number M of nonzeros (and order $N = 200$) is used, it is clear that values of M in the range 20–50 give better compression than the full predictor. However, the optimal M is different for different pieces. When the BIC or PLS criteria are used for order selection, the situation is similar to that shown in the figure for 'de_ore_leonis': the compression is better than that with the best fixed M. For BIC and PLS, M represents the allowed maximum number of nonzeros; compression becomes slightly better as M increases; a value of around 60 gives a good compromise between quality and complexity.

To illustrate the ability of ITC to adapt the predictor order to the audio contents, we show in Figure 7 the average (over frames of 44100 samples) selected sparsity level, μ_L, for PLS and BIC run on 'finlandia'. The structure of the song is visible in the sense that at frames 5, 9 and 13 there are loud horn sections while there is silence in between. Then comes a more uniform part where there are multiple instruments, until about frame 40 at which a quiet flute passage begins.

Increasing prediction order improves the best sparse predictor results for almost all test pieces. Figure 8 shows relative code lengths for several predictor orders, again with respect to a full predictor of length $N = 200$. While the optimal M increases relatively slowly, the compression is better as N grows.

B. Comparison with MPEG-4 ALS

To evaluate the compression quality, we used for comparison the reference software of MPEG-4 Audio Lossless Coding, version `RM23` (see http://www.nue.tu-berlin.de/menue/research/projects/projects/mpeg-4_audio_lossless_coding_als/parameter/en/#230252), selecting its compression mode based on a prediction scheme similar to ours, where MPEG-4 uses a cascade made of an RLS and several LMS full predictors [25]. In Table IX we report the obtained compression ratios, defined as the ratio (original file size)/(compressed file size).

MPEG-4 ALS was run with the parameters `-z1 -r10 -n1764`, which means that the orders were 8 for RLS and 128, 32 and 4 for the cascaded LMS predictors. The overall predictor order is then 173. The option `-r10` means that a "random access" scheme was utilized so that decoding can start at each $10 \cdot 0.1 = 1$ s. This is similar to our scheme and relates to what we name frame length. The option `-n1764` sets the frame length for LPC. Within this backward adaptive context, it merely sets the block size for Golomb-Rice at $1764/4 = 441$.

GRLS was run with the same order, $N = 173$, and the residuals were entropy coded with the MPEG-4 method, such that only predictors performance is compared. (We note that our previously used simple Golomb-Rice residual coding gives only slightly worse results.) The columns BIC 8 and PLS 8 correspond to predictors that always have nonzeros on their first 8 positions; this is justified by the presence of these positions in practically all observed sparse predictors computed by GRLS; this choice slightly improves the compression ratios. It can be seen from Table IX that our sparse predictors are better than the cascaded MPEG-4 predictors. However, at this moment, without having a fully optimized implementation that would compete not only in compression ratio but also in speed, we do not claim that our coder is better than MPEG-4 ALS. We claim only that sparse adaptive predictors are better than full ones for audio signals.

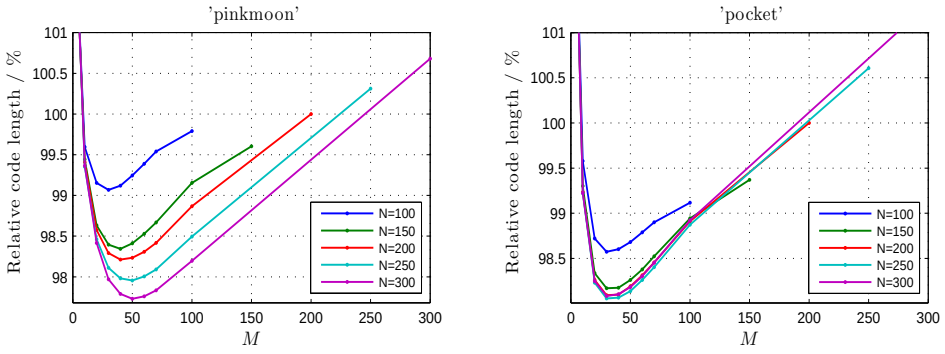


Fig. 8. Relative code lengths for different predictor orders.

TABLE IX
COMPRESSION RATIOS COMPARED TO MPEG-4 ALS, $N = 173$, $M = 100$

	BIC	PLS	BIC 8	PLS 8	MPEG-4
conga	1.349	1.357	1.349	**1.358**	1.357
de_ore_leonis	3.099	**3.105**	3.099	**3.105**	3.075
finlandia	2.273	2.273	2.273	**2.274**	2.242
forty	1.905	**1.910**	1.905	**1.910**	1.909
morgen	2.358	**2.359**	2.359	**2.359**	2.330
mummified	1.380	1.381	1.380	1.381	**1.383**
ordeal	1.965	1.967	1.966	**1.968**	1.961
pinkmoon	1.972	1.983	1.973	**1.984**	1.971
pocket	1.675	1.678	1.676	**1.679**	1.671
prelude	2.721	2.725	2.722	**2.727**	2.725
sowhat	2.012	2.013	2.012	**2.014**	**2.014**
sunflower	4.002	4.042	4.003	**4.044**	3.927
weep	2.403	2.405	2.403	**2.406**	2.386
average	2.240	2.246	2.240	**2.247**	2.227

VIII. CONCLUSIONS

We have presented a family of adaptive greedy RLS (GRLS) algorithms derived from the Orthogonal Least Squares batch algorithm, appropriate for the online computation of sparse solutions to linear systems, with applications to FIR channel identification and linear prediction. The complexity of GRLS is lower than that of the full RLS, while the performance is significantly better when the solution is indeed sparse. For time-varying channels, GRLS produces estimates with lower MSE than the algorithms from [7] and [8]. It appears more robust to variations of the channel conditions, which can be explained in part by the fact that, using information theoretic criteria for estimating the number of nonzero variables, the GRLS algorithms need a single parameter, namely a not necessarily tight upper bound of this number; otherwise, GRLS does not have tunable parameters and does not need the value of the noise variance. Further work will aim to find low-complexity variations of the algorithm that only approximately minimize the RLS criterion, but still have good behavior.

APPENDIX 1: EXPLANATION OF OLS GREEDY CHOICE (7)

After $k-1$ steps of the OLS algorithm, the input data are transformed, similarly to (8), into (we drop all indices)

$$\boldsymbol{Q}^T\boldsymbol{A}\boldsymbol{P} = \begin{bmatrix} \boldsymbol{R} & \dots & \boldsymbol{v} & \dots \\ 0 & \dots & \boldsymbol{w} & \dots \end{bmatrix}, \quad \boldsymbol{Q}^T\boldsymbol{d} = \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{h} \end{bmatrix}, \quad (32)$$

where $\boldsymbol{R} \in \mathbb{R}^{(k-1)\times(k-1)}$, $\boldsymbol{c} \in \mathbb{R}^{k-1}$ and we highlight a single (yet) inactive column of the transformed data matrix. Denote $\hat{\boldsymbol{x}}_{t,k-1}$ the $(k-1)$-sparse solution thus obtained, whose nonzero elements are $\boldsymbol{R}^{-1}\boldsymbol{c}$. Since $\boldsymbol{Q}$ is orthogonal, the criterion (4) is

$$J_t(\hat{\boldsymbol{x}}_{t,k-1}) = \|\boldsymbol{h}\|^2 = \|\boldsymbol{d}\|^2 - \|\boldsymbol{c}\|^2. \quad (33)$$

Lemma. If the highlighted column is included in the active set, the criterion (4) becomes

$$J_t(\hat{\boldsymbol{x}}_{t,k}) = \|\boldsymbol{d}\|^2 - \|\boldsymbol{c}\|^2 - \frac{(\boldsymbol{w}^T\boldsymbol{h})^2}{\|\boldsymbol{w}\|^2}. \quad (34)$$

Remark. At step k, the OLS greedy selection (7) chooses the column for which $|\boldsymbol{w}^T\boldsymbol{h}|/\|\boldsymbol{w}\|$ is maximum, hence indeed minimizes the criterion (4), for the given first $k-1$ columns.

Proof. Let $\boldsymbol{U}$ be an orthogonal matrix, e.g. a Householder reflector, such that $\boldsymbol{U}\boldsymbol{w} = \rho\boldsymbol{e}_1$, where $\rho = \|\boldsymbol{w}\|$ and $\boldsymbol{e}_1$ is the unit vector with 1 in the first position and zeros elsewhere. Denoting $\boldsymbol{u}^T$ the first row of $\boldsymbol{U}$, since $\boldsymbol{w} = \rho\boldsymbol{U}^T\boldsymbol{e}_1 = \rho\boldsymbol{u}$, it follows that $\boldsymbol{u} = \boldsymbol{w}/\rho$. To continue the QR factorization, we multiply (32) by $\begin{bmatrix} \boldsymbol{I} & 0 \\ 0 & \boldsymbol{U} \end{bmatrix}$ at the left, obtaining

$$\begin{bmatrix} \boldsymbol{I} & 0 \\ 0 & \boldsymbol{U} \end{bmatrix}\begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{h} \end{bmatrix} = \begin{bmatrix} \boldsymbol{c} \\ \boldsymbol{U}\boldsymbol{h} \end{bmatrix} = \begin{bmatrix} \boldsymbol{c} \\ \gamma \\ \tilde{\boldsymbol{h}} \end{bmatrix},$$

where

$$\gamma = \boldsymbol{e}_1^T\boldsymbol{U}\boldsymbol{h} = \boldsymbol{u}^T\boldsymbol{h} = \frac{1}{\rho}\boldsymbol{w}^T\boldsymbol{h} = \frac{\boldsymbol{w}^T\boldsymbol{h}}{\|\boldsymbol{w}\|}.$$

The criterion is now

$$J_t(\hat{\boldsymbol{x}}_{t,k}) = \|\tilde{\boldsymbol{h}}\|^2 = \|\boldsymbol{d}\|^2 - \|\boldsymbol{c}\|^2 - \gamma^2,$$

which is (34).

APPENDIX 2: EFFICIENT COMPUTATION OF A PRIORI ESTIMATION ERRORS

We show here how the a priori estimation errors $e_t(k) = d_t - \boldsymbol{a}_t^T \hat{\boldsymbol{x}}_{t-1,k}$, $k = 1 : M$, used in the PLS criterion, can be computed efficiently. After the operations at time $t-1$ are completed, denoting $\boldsymbol{R}_k = \boldsymbol{R}(1:k, 1:k)$, $\boldsymbol{b}_k = \boldsymbol{b}(1:k)$ for some $k \leq M$, the GRLS data are split as

$$\boldsymbol{R}_k = \begin{bmatrix} \boldsymbol{R}_{k-1} & \boldsymbol{v} \\ 0 & \rho \end{bmatrix}, \quad \boldsymbol{b}_k = \begin{bmatrix} \boldsymbol{b}_{k-1} \\ \beta \end{bmatrix}.$$

The (permuted) input vector at time t is split accordingly into

$$\boldsymbol{a}_t^T(\boldsymbol{p}(1:k)) = [\boldsymbol{a}^T \ \alpha].$$

The nonzero elements of $\hat{\boldsymbol{x}}_{t-1,k-1}$ are equal to $\boldsymbol{R}_{k-1}^{-1}\boldsymbol{b}_{k-1}$ and so

$$e_t(k-1) = d_t - \boldsymbol{a}^T \boldsymbol{R}_{k-1}^{-1}\boldsymbol{b}_{k-1} = d_t - \boldsymbol{z}_{k-1}^T \boldsymbol{b}_{k-1},$$

where we define $\boldsymbol{z}_{k-1}^T = \boldsymbol{a}^T \boldsymbol{R}_{k-1}^{-1}$. Similarly, the a priori estimation error of the k-sparse solution is

$$e_t(k) = d_t - \boldsymbol{z}_k^T \boldsymbol{b}_k, \tag{35}$$

where

$$\begin{aligned} \boldsymbol{z}_k^T &= [\boldsymbol{a}^T \ \alpha]\boldsymbol{R}_k^{-1} = [\boldsymbol{a}^T \ \alpha] \begin{bmatrix} \boldsymbol{R}_{k-1} & \boldsymbol{v} \\ 0 & \rho \end{bmatrix}^{-1} \\ &= [\boldsymbol{a}^T \ \alpha] \begin{bmatrix} \boldsymbol{R}_{k-1}^{-1} & -\frac{1}{\rho}\boldsymbol{R}_{k-1}^{-1}\boldsymbol{v} \\ 0 & \frac{1}{\rho} \end{bmatrix} \\ &= \left[\boldsymbol{z}_{k-1}^T \ \ \frac{1}{\rho}\left(\alpha - \boldsymbol{z}_{k-1}^T \boldsymbol{v}\right)\right] \stackrel{\text{def}}{=} [\boldsymbol{z}_{k-1}^T \ \zeta]. \end{aligned}$$

This recursion, initialized with $\boldsymbol{z}_1 = \boldsymbol{a}_t(\boldsymbol{p}(1))/\boldsymbol{R}(1,1)$, shows that $\boldsymbol{z}_k$ can be computed from $\boldsymbol{z}_{k-1}$ with $O(k)$ operations. Moreover, the scalar product appearing in (35) can be updated as

$$\boldsymbol{z}_k^T \boldsymbol{b}_k = \boldsymbol{z}_{k-1}^T \boldsymbol{b}_{k-1} + \zeta\beta,$$

hence needs only $O(1)$ operations. Overall, it results that $O(M^2)$ operations are needed for all a priori errors $e_t(k)$, $k = 1 : M$.

REFERENCES

[1] A.M. Bruckstein, D.L. Donoho, and M. Elad, "From Sparse Solutions of Systems of Equations to Sparse Modeling of Signals and Images," *SIAM Rev.*, vol. 51, no. 1, pp. 34–81, 2011.

[2] D.L. Duttweiler, "Proportionate Normalized Least-Mean-Squares Adaptation in Echo Cancelers," *IEEE Trans. Speech Audio Proc.*, vol. 8, no. 5, pp. 508–518, Sept. 2000.

[3] R.K. Martin, W.A. Sethares, R.C. Williamson, and C.R. Johnson, Jr., "Exploiting Sparsity in Adaptive Filters," *IEEE Trans. Signal Proc.*, vol. 50, no. 8, pp. 1883–1894, Aug. 2002.

[4] P.A. Naylor, J. Cui, and M. Brookes, "Adaptive Algorithms for Sparse Echo Cancellation," *Signal Proc.*, vol. 86, no. 6, pp. 1182–1192, 2006.

[5] L.R. Vega, H. Rey, J. Benesty, and S. Tressens, "A Family of Robust Algorithms Exploiting Sparsity in Adaptive Filters," *IEEE Trans. Audio Speech Lang. Proc.*, vol. 17, no. 4, pp. 572–581, May 2009.

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

[7] D. Angelosante, J.A. Bazerque, and G.B. Giannakis, "Online Adaptive Estimation of Sparse Signals: Where RLS Meets the ℓ_1-Norm," *IEEE Trans. Signal Proc.*, vol. 58, no. 7, pp. 3436–3447, July 2010.

[8] B. Babadi, N. Kalouptsidis, and V. Tarokh, "SPARLS: The Sparse RLS Algorithm," *IEEE Trans. Signal Proc.*, vol. 58, no. 8, pp. 4013–4025, Aug. 2010.

[9] Y. Kopsinis, K. Slavakis, and S. Theodoridis, "Online Sparse System Identification and Signal Reconstruction Using Projections Onto Weighted ℓ_1 Balls," *IEEE Trans. Signal Proc.*, vol. 59, no. 3, pp. 936–952, Mar. 2011.

[10] N. Kalouptsidis, G. Mileounis, B. Babadi, and V. Tarokh, "Adaptive Algorithms for Sparse System Identification," *Signal Proc.*, vol. 91, pp. 1910–1919, 2011.

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

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

[13] G. Mileounis, B. Babadi, N. Kalouptsidis, and V. Tarokh, "An Adaptive Greedy Algorithm With Application to Nonlinear Communications," *IEEE Trans, Signal Proc.*, vol. 58, no. 6, pp. 2998–3007, June 2010.

[14] S. Chen, S.A. Billings, and W. Luo, "Orthogonal Least Squares Methods and Their Application to Non-Linear System Identification," *Int. J. Control*, vol. 50, no. 5, pp. 1873–1896, 1989.

[15] L. Rebollo-Neira and D. Lowe, "Optimized Orthogonal Matching Pursuit Approach," *IEEE Signal Proc. Letters*, vol. 9, no. 4, pp. 137–140, April 2002.

[16] G.V. Rocha and B. Yu, "Greedy and Relaxed Approximations to Model Selection: a Simulation Study," in *Festschrift in honor of Jorma Rissannen on the occasion of his 75th birthday*, pp. 63–80. TICSP Series no.38, 2007.

[17] J. Rissanen, *Information and Complexity in Statistical Modeling*, Springer, 2007.

[18] B. Dumitrescu and I. Tăbuş, "Greedy RLS for Sparse Filters," in *European Sign. Proc. Conf. EUSIPCO*, Aalborg, Denmark, 2010, pp. 1484–1488.

[19] A. Onose, B. Dumitrescu, and I. Tăbuş, "Sliding Window Greedy RLS for Sparse Filters," in *ICASSP*, Prague, Czech Republic, 2011.

[20] C.D. Giurcăneanu and S.A. Razavi, "AR Order Selection in the Case When the Model Parameters Are Estimated by Forgetting Factor Least-Squares Algorithms," *Signal Proc.*, vol. 90, pp. 451–466, 2010.

[21] M. Niedzwiecki, "Bayesian-like Autoregressive Spectrum Estimation in the Case of Unknown Process Order," *IEEE Trans. Auto. Control*, vol. 30, no. 10, pp. 950–961, Oct. 1985.

[22] J. Rissanen, "Order Estimation by Accumulated Prediction Errors," *J. Appl. Prob.*, vol. 23, pp. 55–61, 1986.

[23] E.J. Hannan, A.J. McDougall, and D.S. Poskitt, "Recursive Estimation of Autoregressions," *J. Royal Stat. Soc.. Ser. B*, vol. 51, no. 2, pp. 217–233, 1989.

[24] G. Schwarz, "Estimating the Dimension of a Model," *Ann. Stat.*, vol. 6, no. 2, pp. 461–464, 1978.

[25] H. Huang, P. Franti, D. Huang, and S. Rahardja, "Cascaded RLS-LMS Prediction in MPEG-4 Lossless Audio Coding," *IEEE Trans. Audio Speech Lang. Proc.*, vol. 16, no. 3, pp. 554–562, Mar. 2008.

Bogdan Dumitrescu (M'01) was born in Bucharest, Romania, in 1962. He received the M.S. and Ph.D. degrees in 1987 and 1993, respectively, from the "Politehnica" University of Bucharest, Romania.

He is now a Professor with the Department of Automatic Control and Computers, "Politehnica" University of Bucharest and a FiDiPro Fellow with the Department of Signal Processing, Tampere University of Technology, Finland. He is an Associate Editor for IEEE Transactions on Signal Processing since 2008. His scientific interests are in numerical methods, optimization, and their applications to signal processing.

Alexandru Onose (S'11) was born in Bârlad, Romania. He received the Diploma Engineer degree in automatic control and applied informatics from "Politehnica" University of Bucharest, Bucharest, Romania, in 2009.

He is currently working towards the Ph.D. degree in signal processing at the Department of Signal Processing, Tampere University of Technology, Tampere, Finland. His research interests include adaptive signal processing, sparse representation and compressed sensing.

Petri Helin was born in Turku, Finland, in 1987. He received the B.Sc. degree in signal processing from Tampere University of Technology in 2011. He has been working in the field of lossless audio compression as a research assistant with the Department of Signal Processing since 2009.

Ioan Tăbuş (SM'99) Ioan Tabus received the M.S. degree in electrical engineering in 1982, the Ph.D. degree from the "Politehnica" University of Bucharest, Romania, in 1993, and the Ph.D. degree (with honors) from Tampere University of Technology (TUT), Finland, in 1995. He was a Teaching Assistant, Lecturer, and Associate Professor with the Department of Control and Computers, "Politehnica" University of Bucharest between 1984 and 1995. From 1996 to 1999 he was a Senior Researcher and since January 2000 he has been a Professor with the Signal Processing Department at TUT. He is coauthor of more than 170 publications in the fields of signal processing, signal compression, and genomic signal processing. He is a Senior Member of IEEE and was an Associate Editor for IEEE Transactions on Signal Processing between 2002 and 2005. He was a member of the TC Bio Image and Signal Processing of IEEE Signal Processing Society. He currently is the Editor-in-Chief of EURASIP Journal on Bioinformatics and Systems Biology. Dr. Tabus is co-recipient of 1991 "Traian Vuia" Award of the Romanian Academy and co-recipient of the NSIP 2001 Best Paper Award and Norsig 2004 Best Paper Award.

Publication 6

[P6] A. Onose, B. Dumitrescu, and I. Tăbuş. Sliding window greedy RLS for sparse filters. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3916–3919, Prague, Czech Republic, May 2011.

SLIDING WINDOW GREEDY RLS FOR SPARSE FILTERS

Alexandru Onose, Bogdan Dumitrescu, Ioan Tăbuş

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

ABSTRACT

We present a sliding window RLS for sparse filters, based on the greedy least squares algorithm. The algorithm adapts a partial QR factorization with pivoting, using a simplified search of the filter support that relies on a neighbor permutation technique. For relatively small window size, the proposed algorithm has a lower complexity than recent exponential window RLS algorithms. Time-varying FIR channel identification simulations show that the proposed algorithm can also give better mean squared coefficient errors.

***Index Terms*—** adaptive algorithm, recursive least squares, sliding window, sparse filters

1. INTRODUCTION

The recursive least squares (RLS) algorithm is most often implemented using an exponential window (EW). Although sliding window (SW) algorithms exist [1, 2], they offer relatively few benefits, while having a higher complexity; hence, they are much less used than EW RLS. The recent interest in RLS algorithms for sparse filters [3, 4, 5] was focused on EW RLS. Our aim in this paper is to show that, opposite to the full filters case, a sliding window RLS can have lower complexity than EW RLS.

At each time $t \in \mathbb{N}$, the RLS algorithm provides a least-squares solution $\boldsymbol{x}_t \in \mathbb{R}^N$ to the overdetermined linear system $\boldsymbol{A}_t\boldsymbol{x}_t \approx \boldsymbol{b}_t$, taking into account the new equation $\boldsymbol{a}_t^T\boldsymbol{x}_t \approx b_t$ together with previous equations of the same form. In a time-varying context, more weight is given to recent equations, while the past needs to be forgotten. The SW RLS criterion for a window of length L is

$$J(t) = \sum_{\tau=0}^{L-1} \lambda^\tau |e(t-\tau)|^2, \tag{1}$$

where λ is the forgetting factor and

$$e(t) = b_t - \boldsymbol{a}_t^T\boldsymbol{x}_t \tag{2}$$

is the approximation error for the equation appeared at time t. So, only the most recent L equations are considered for computing $\boldsymbol{x}_t$. For example, in an FIR channel identification problem, the error (2) has the form

$$e(t) = y(t) - \sum_{i=0}^{N-1} h_i u(t-i), \tag{3}$$

where $u(t)$ is the input, $y(t)$ is the output and $\boldsymbol{h} \in \mathbb{R}^N$ is the vector of filter coefficients; the correspondence with (2) is immediate.

We consider the SW RLS problem for sparse filters: we assume that the solution $\boldsymbol{x}_t$ has at most M nonzero elements, M being given; the typical case is $M \ll N$. For full filters, it is necessary to take $L > N$ in order to obtain a least-squares solution. However, for sparse filters, the required condition is $L > M$, which leaves open the possibility to take $L < N$. This is the key to low complexity and good performance is obtained for sufficiently fast time-varying channels, as we will show later.

For solving the sparse SW RLS problem, we adapt the greedy least-squares (GLS) algorithm [6] (known also, in a modified form, as optimized orthogonal matching pursuit [7]) to the QR factorization update algorithm from [1]. In particular, for the greedy selection of the nonzero elements of the solution, we employ the neighbor-restricted search technique introduced in [5].

2. SLIDING WINDOW GREEDY RLS

Using a sliding window, the least-squares approximation problem corresponding to the criterion (1) is to minimize $\|\boldsymbol{b}_t - \boldsymbol{A}_t\boldsymbol{x}_t\|_2$, with $\boldsymbol{A}_t \in \mathbb{R}^{L\times N}$, under the constraint that $\boldsymbol{x}_t$ has M nonzero elements. (For simplicity, we assume that the forgetting factors are included in $\boldsymbol{A}_t$ and $\boldsymbol{b}_t$.) The algorithm we propose maintains the partial QR factorization with pivoting

$$\boldsymbol{A}_t\boldsymbol{P}_t = \boldsymbol{Q}_t\boldsymbol{R}_t, \tag{4}$$

where $\boldsymbol{Q}_t \in \mathbb{R}^{L\times L}$ is an orthogonal matrix, $\boldsymbol{R}_t \in \mathbb{R}^{L\times N}$ is upper triangular in its first M columns and $\boldsymbol{P}_t$ is a permu-

This work was supported by Tekes FiDiPro – Finland Distinguished Professor Programme. B.Dumitrescu is also with Department of Automatic Control and Computers, "Politehnica" University of Bucharest, Romania.

tation matrix responsible for bringing into the first M positions the columns corresponding to the nonzero elements of $\boldsymbol{x}_t$ (called active columns). These elements are computed by solving the triangular system with the matrix $\boldsymbol{R}_t(1:M, 1:M)$ and the right hand side made by the first M elements of $\boldsymbol{Q}_t^T \boldsymbol{b}_t$.

At time t, the algorithm has two main steps: downdating and updating [1]. Downdating has the purpose of eliminating the oldest equation from the previous window, i.e., starting from (4) at time $t-1$, to compute

$$\boldsymbol{A}_{t-1}\boldsymbol{P}_{t-1} = \begin{bmatrix} 1 & 0 \\ 0 & \tilde{\boldsymbol{Q}}_{t-1} \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{a}}_{t-L}^T \\ \tilde{\boldsymbol{R}}_{t-1} \end{bmatrix}, \tag{5}$$

where $\tilde{\boldsymbol{a}}_{t-L}^T = \boldsymbol{a}_{t-L}^T \boldsymbol{P}_{t-1}$; hence, the first row of (5) reads $\tilde{\boldsymbol{a}}_{t-L}^T = \tilde{\boldsymbol{a}}_{t-L}^T$ and can be eliminated. The matrix $\tilde{\boldsymbol{R}}_{t-1} \in \mathbb{R}^{(L-1)\times N}$ is upper triangular in its first M columns, hence the last $L-1$ rows of $\boldsymbol{A}_{t-1}\boldsymbol{P}_{t-1}$ have the QR factorization $\tilde{\boldsymbol{Q}}_{t-1}\tilde{\boldsymbol{R}}_{t-1}$.

Updating starts from adding the (permuted) current equation to the above factorization

$$\boldsymbol{A}_t\boldsymbol{P}_{t-1} = \begin{bmatrix} \tilde{\boldsymbol{Q}}_{t-1} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{R}}_{t-1} \\ \tilde{\boldsymbol{a}}_t^T \end{bmatrix}, \tag{6}$$

where $\tilde{\boldsymbol{a}}_t^T = \boldsymbol{a}_t\boldsymbol{P}_{t-1}$, and obtains (4). It is in the updating stage that active columns are (re)selected. Following the strategy introduced in [5], we allow at most one inactive column to enter (and hence another column to leave) the active set at time t.

All downdating and updating operations use orthogonal transformations and hence are numerically reliable. We give below details on the operations performed in the two steps. The matrix variables in the algorithm are $\boldsymbol{R} \in \mathbb{R}^{L\times N}$ for storing $\boldsymbol{R}_t$ and $\boldsymbol{U} \in \mathbb{R}^{L\times L}$ for storing $\boldsymbol{Q}_t^T$ (we work on the transposed for applying all transformations from the left); the column permutations will be described informally. The variable $\boldsymbol{b}$ will store $\boldsymbol{Q}_t^T\boldsymbol{b}_t$.

2.1. Downdating

Downdating consists of computing the elementary orthogonal transformations that bring $\boldsymbol{Q}_{t-1}$ to the form from (5), i.e. force zeros in its first row (and column), in an order chosen to damage the least the triangular form of $\boldsymbol{R}_{t-1}$. The initial and final form of the matrices $\boldsymbol{U}$ and $\boldsymbol{R}$ are shown in Figure 1, where $M=3$, $L=6$ and only the first 5 of the N columns of $\boldsymbol{R}$ are depicted. The downdating process is similar with that from [1], excepting that here a Householder reflector can be used for zeroing the last $L-M+1$ elements of the first column of $\boldsymbol{U}$, instead of several Givens rotations. The main operations are the following.

1. Compute reflector $\boldsymbol{H}$ that zeroes $\boldsymbol{U}(M+2:L, 1)$ and apply it: $\boldsymbol{U} \leftarrow \boldsymbol{H}\boldsymbol{U}$, $\boldsymbol{R} \leftarrow \boldsymbol{H}\boldsymbol{R}$, $\boldsymbol{b} \leftarrow \boldsymbol{H}\boldsymbol{b}$.

2. For $k = M:-1:1$

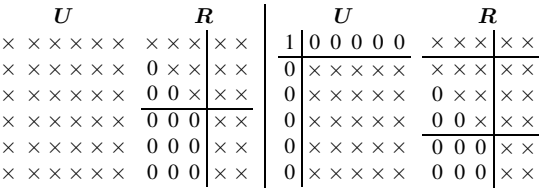


Fig. 1. $\boldsymbol{U}$ and $\boldsymbol{R}$ before and after downdating.

2.1. Compute rotation $\boldsymbol{G}$ that zeroes $\boldsymbol{U}(k+1, 1)$, based on $\boldsymbol{U}(k, 1)$, and apply it: $\boldsymbol{U} \leftarrow \boldsymbol{G}\boldsymbol{U}$, $\boldsymbol{R} \leftarrow \boldsymbol{G}\boldsymbol{R}$, $\boldsymbol{b} \leftarrow \boldsymbol{G}\boldsymbol{b}$.

Of course, the above operations are performed efficiently: in step 1, only rows $M+1:L$ of the matrices are modified; in step 2.1, only rows k and $k+1$ are affected. This operation order ensures that at the end $\boldsymbol{R}$ is upper Hessenberg in the first M columns and, after eliminating its first row, it becomes upper triangular in the first M columns.

2.2. Updating

Greedy LS belongs to the family of matching pursuit algorithms and finds the nonzero (active) positions of $\boldsymbol{x}_t$ (or the corresponding columns of $\boldsymbol{A}_t$) one by one. Each position is chosen such that, when adding it to the active set, the LS residual corresponding to the solution obtained using the current active positions is minimized. In the original batch algorithm, all positions are allowed to compete for the active set. In a recursive context, it is unlikely that radical changes of the active set are possible to detect immediately. So, it is enough to allow gradual changes. We have shown in [5] that, with an exponential window, the following strategy for changing the active set is successful in tracking a time-varying channel:

- For position $k \le M-1$, choose only between columns k and $k+1$ of the current active set, process called neighbor permutation.
- For position M, choose between all remaining columns.

So, at most one new column enters the active set at time t. We adopt the same strategy for the SW algorithm and give next a formal description of the update step, then explain it in more detail. Remind that the update starts with $\boldsymbol{R}$ as in (6), i.e. upper triangular in the first M columns, with the exception of the last row, as in the leftmost diagram from Figure 2.

A. *Neighbor permutation*

1. For $k = 1:M-1$

1.1. Find the index of the best column in $\mathcal{S} = \{k, k+1\}$:

$$\tilde{k} = \arg\max_{i\in\mathcal{S}} \frac{\boldsymbol{R}(k:L, i)^T \cdot \boldsymbol{b}(k:L)}{\|\boldsymbol{R}(k:L, i)\|_2^2} \tag{7}$$

1.2. If $\tilde{k} = k+1$ (permutation is needed)

1.2.1. Swap columns k and $k+1$ of $\boldsymbol{R}$

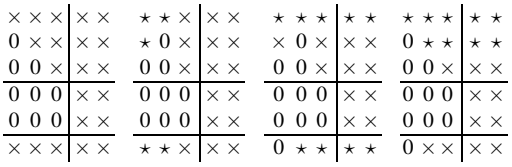


Fig. 2. Evolution of $\boldsymbol{R}$ in first neighbor permutation.

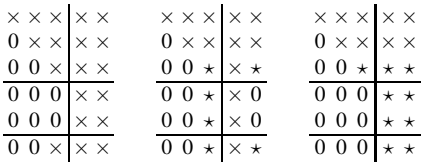


Fig. 3. Evolution of $\boldsymbol{R}$ after selection of last active column.

1.2.2. Compute rotation $\boldsymbol{G}$ that zeroes $\boldsymbol{R}(L,k)$, based on $\boldsymbol{R}(k,k)$, and apply it: $\boldsymbol{U} \leftarrow \boldsymbol{GU}$, $\boldsymbol{R} \leftarrow \boldsymbol{GR}$, $\boldsymbol{b} \leftarrow \boldsymbol{Gb}$.

1.3. Compute rotation $\boldsymbol{\Gamma}$ that zeroes $\boldsymbol{R}(k+1,k)$, based on $\boldsymbol{R}(k,k)$, and apply it: $\boldsymbol{U} \leftarrow \boldsymbol{\Gamma U}$, $\boldsymbol{R} \leftarrow \boldsymbol{\Gamma R}$, $\boldsymbol{b} \leftarrow \boldsymbol{\Gamma b}$.

B. *Selection of last active column*

2. Find the best column in $\mathcal{S} = M : N$, by computing (7)

3. Swap columns k and $\tilde{k}$ of $\boldsymbol{R}$

4. Compute the reflector $\boldsymbol{H}$ that zeroes $\boldsymbol{U}(M+2:L,1)$ and apply it: $\boldsymbol{U} \leftarrow \boldsymbol{HU}$, $\boldsymbol{R} \leftarrow \boldsymbol{HR}$, $\boldsymbol{b} \leftarrow \boldsymbol{Hb}$.

Figure 2 illustrates the operations made on $\boldsymbol{R}$ in steps 1.2 and 1.3, for $k = 1$. (The operations on $\boldsymbol{U}$ are trivial, since $\boldsymbol{U}$ is practically a full matrix.) Nonzero elements are marked by $\times$, modified elements are marked by $\star$. The second diagram shows $\boldsymbol{R}$ after swapping the first two columns. The next two diagrams depict the successive elimination of the two nonzero subdiagonal elements on the first column.

Figure 3 shows the transformations of $\boldsymbol{R}$ after the selection of the last active column. The second diagram shows the matrix after the permutation of column $\tilde{k}$ (here $\tilde{k} = 5$) in position M and the third after the application of the Householder reflector that completes the upper triangular structure in the first M columns and hence the updating process. The M nonzero elements of the solution are computed by solving the triangular system $\boldsymbol{R}(1:M,1:M) \cdot \boldsymbol{x} = \boldsymbol{b}(1:M)$.

The criterion (7) has the general form for GLS: it selects the column that is best aligned with the residual, after orthogonalization of the active columns. In step 1.1, due to the structure of $\boldsymbol{R}$, the computation of (7) involves only a few operations. In step 2, we have not implemented (7) directly, but via scalar products $\boldsymbol{R}(M+1:L,i)^T \cdot \boldsymbol{b}(M+1:L)$, $i = M+1:N$, that are cheaply updated at each time t.

We also note that updating or downdating can be performed alone at time t, hence increasing or decreasing L. Also, the above algorithms can be easily modified for allowing M to increase or decrease. Further work will be directed towards finding a criterion for adapting M dynamically towards the true number of nonzero elements of the LS solution.

2.3. Complexity issues

The active set needs not be changed at each moment, but only when t is a multiple of a small integer τ_0. When the active set is not changed, the updating algorithm becomes much simpler; column permutations are not needed and the triangular structure is restored plainly with Givens rotations.

We assume that $M \ll N$, i.e. the solution is indeed sparse. With a careful implementation, the average number of operations (multiplications or additions) required by downdating and updating is

$$\nu \approx \left(4 + \frac{4}{\tau_0}\right) L(N+L). \tag{8}$$

Taking $\tau_0 = 2$, then for $L = N/2$ it results that $\nu \approx 4.5N^2$, which is already less than the cost of a full length RLS. For $L = N/4$, it results that $\nu < 2N^2$; in this case the algorithm becomes cheaper than the EW greedy LS from [5] or other EW algorithms [3, 4].

So, if the window size is small enough, the sliding window algorithm has a complexity advantage over algorithms based on exponential window. In the next section we will show that for some time-varying systems one can take L small enough, with quality comparable to that of EW algorithms.

3. SIMULATION RESULTS

We have tested the SW RLS algorithm for the FIR channel identification problem (3), where the channel filters have $\tilde{M} = 5$ nonzero coefficients and order $N = 200$. The nonzero coefficients are in random positions and are time-varying upon the multipath fading model

$$h_i(t) = c_i \cos(2\pi f_d T_s t + \alpha_i), \tag{9}$$

with amplitude c_i and phase α_i uniformly distributed in $[0.05, 1]$ and $[0, 2\pi]$, respectively; f_d is the Doppler frequency shift and T_s is the channel sampling interval; the product $f_d T_s$ determines the variation speed of the channel. The input $u(t)$ is Gaussian from $\mathcal{N}(0,1)$ and the output $y(t)$ is affected by additive white noise with variance $\sigma^2 = 0.01$.

Denoting $\boldsymbol{h}(t) \in \mathbb{R}^{N+1}$ the true vector of coefficients and $\hat{\boldsymbol{h}}(t)$ the estimated one at time t we have measured the mean squared coefficient error

$$\text{MSE}(t) = \frac{E\{\|\hat{\boldsymbol{h}}(t) - \boldsymbol{h}(t)\|_2^2\}}{E\{\|\boldsymbol{h}(t)\|_2^2\}} \tag{10}$$

by averaging 1000 runs with different channel filters at each run, all with the same variation speed $f_d T_s$. In Table 1, we

	SW-GRLS	EW-GRLS	RLS	SI-SW-RLS
f_dT_s	MSE (L,λ)	MSE (λ)	MSE (λ)	MSE (L,λ)
0.0002	0.00352 (55,0.98)	0.00333 (0.96)	0.051 (0.98)	0.00182 (55,0.98)
0.0005	0.00606 (45,0.96)	0.00621 (0.94)	0.176 (0.96)	0.00328 (45,0.96)
0.001	0.01202 (40,0.92)	0.01233 (0.92)	0.568 (0.96)	0.00599 (40,0.92)
0.002	0.02684 (40,0.90)	0.02883 (0.90)	1.333 (0.98)	0.01363 (40,0.90)
0.005	0.11973 (25,0.90)	0.13187 (0.86)	1.391 (0.995)	0.05128 (25,0.90)

Table 1. MSE for the studied algorithms.

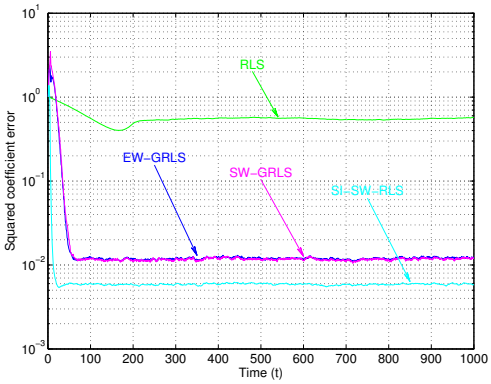


Fig. 4. Squared coefficient error for $M = 5$.

report the time-average of MSE(t) (computed after the end of the initial transient regime) for four algorithms: SW-GRLS, the algorithm presented in this paper; EW-GRLS, from [5], RLS, the standard algorithm that assumes a full filter of order N; SI-SW-RLS, a sparsity informed SW RLS, knowing the positions of the nonzero coefficients. Both SW and EW-GRLS assume that $M = \tilde{M}$ and use $\tau_0 = 2$. Besides MSE, the table contains the optimized values of (L, λ) for SW-GRLS (used also by SI-SW-GRLS) and of λ for EW-GRLS; the optimization was done for each algorithm separately. Figure 4 shows the evolution of the squared coefficient error (10) averaged over the 1000 runs, for $f_dT_s = 0.001$.

It can be noticed from Table 1 that, as the channel variation speed increases, the best window length L decreases and also SW-GRLS becomes increasingly better compared to its EW counterpart. Since L is about or less than $N/4$, the complexity of SW-GRLS is lower than that of EW-GRLS. While SW-GRLS largely (and expectedly) outperformes the full RLS algorithm (which is unable to give meaningful results for higher variation speeds), it gives a MSE only about twice larger than the sparsity informed RLS, which is the best attainable by an RLS algorithm. If $M > \tilde{M}$, the performance degrades, but is still acceptable if M is only slightly larger than $\tilde{M}$. For example, for $M = 10$, MSE is about twice larger than the values from Table 1.

Comparisons with SPARLS [4] appear to be favorable, but not included here since SPARLS adapts also to the true number of coefficients $\tilde{M}$, while our algorithm does not yet adapt.

4. CONCLUSIONS

We have presented a sliding window RLS that uses a greedy approach for finding the nonzero coefficients of the sparse solution. The algorithm is based on orthogonal operations and hence is numerically stable. Theoretical and simulation study show that, for small window length ($L < N/4$) and for sufficiently fast time-varying channels, the algorithm is less complex and gives better coefficient errors than exponential window greedy RLS. So, for sparse filters, the sliding window can give benefits in terms of both complexity and estimation quality, a property that does not hold for nonsparse filters.

5. REFERENCES

[1] M.P. Mahon, L.H. Sibul, and H.M. Valenzuela, "A Sliding Window Update for the Basis Matrix of the QR Decomposition," *IEEE Trans. Signal Proc.*, vol. 41, no. 5, pp. 1951–1953, May 1993.

[2] C. Papaodysseus, "A Robust, Parallelizable, $O(m)$, A Posteriori Recursive Least Squares Algorithm for Efficient Adaptive Filtering," *IEEE Trans. Signal Proc.*, vol. 47, no. 9, pp. 2252–2257, Sept. 1999.

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

[4] B. Babadi, N. Kalouptsidis, and V. Tarokh, "SPARLS: The Sparse RLS Algorithm," *IEEE Trans. Signal Proc.*, vol. 58, no. 8, pp. 4013–4025, Aug. 2010.

[5] B. Dumitrescu and I. Tăbuş, "Greedy RLS for Sparse Filters," in *European Sign. Proc. Conf. EUSIPCO*, Aalborg, Denmark, 2010, pp. 1484–1488.

[6] S. Chen, S.A. Billings, and W. Luo, "Orthogonal Least Squares Methods and Their Application to Non-Linear System Identification," *Int. J. Control*, vol. 50, no. 5, pp. 1873–1896, 1989.

[7] L. Rebollo-Neira and D. Lowe, "Optimized Orthogonal Matching Pursuit Approach," *IEEE Signal Proc. Letters*, vol. 9, no. 4, pp. 137–140, April 2002.

Publication 7

[P7] A. Onose and B. Dumitrescu. Group greedy RLS sparsity estimation via information theoretic criteria. In *Proceedings of the International Conference on Control Systems and Computer Science*, volume 2, pages 359–364, Bucharest, Romania, May 2013.

Group Greedy RLS Sparsity Estimation via Information Theoretic Criteria

Alexandru Onose*, Bogdan Dumitrescu*,†
*Tampere University of Technology,
Department of Signal Processing,
FI-33101 Tampere, Finland
†'Politehnica' University of Bucharest,
Department of Automatic Control and Computers,
RO-060042 Bucharest, Romania
emails: alexandru.onose@tut.fi, bogdan.dumitrescu@tut.fi

***Abstract*—This work introduces a group sparse adaptive greedy algorithm that uses information theoretic criteria (ITC) to estimate online the sparsity level. The algorithm selects a set of candidate groups using group neighbor permutations and maintains a partial QR decomposition to compute the solution. It contains a mechanism that allows group joining which, complementing the splitting of groups, produces a robust algorithm. We focus here on a study of the ITC use, namely the predictive least squares (PLS) and Bayesian information criterion (BIC), in conjunction with the group sparse algorithm. We propose several forms of group oriented ITC and evaluate them with extensive simulations for a time-varying channel identification problem. Compared to the non group aware counterparts, the performance is improved at the cost of higher complexity. The best results are given by a group PLS criterion directly generalizing the standard PLS.**

***Keywords*—adaptive greedy algorithm, group sparse filters, model selection, channel identification**

I. Introduction

In the quickly expanding field of sparse approximations, the development of adaptive algorithms is an important topic receiving much interest in recent years. Most of the latest work addresses the modifications needed by classic algorithms, like LMS or RLS, to obtain good performance on finding on-line sparse linear models and thus solving typical signal processing problems like channel identification. Sparse LMS was pioneered in [1], while for sparse RLS there are several approaches, using techniques inspired by convex relaxation [2], [3] or having greedy nature [4]. Among other notable works, the first sparse adaptive algorithm was a direct modification of matching pursuit [5]; more recently, projections onto hyperslabs were used in [6] and techniques resulting from traditional adaptive filtering in [7].

One of the significant difficulties in sparse adaptive filtering is to decide on-line, at each time moment, the sparsity level of the filter, i.e. the number of nonzero coefficients. Algorithms based on ℓ_1 relaxation do this implicitly, by the choice of a trade-off parameter between sparsity and representation error. A different approach was promoted in [4], by using Information Theoretic Criteria (ITC), allowing to select the sparsity level without using any extra parameters, by proposing low complexity adaptive implementations of ITC like Predictive Least Squares (PLS) [8] and Bayesian Information Criterion (BIC) [9].

The adaptive filtering problem at hand is to find a recursive least-squares solution to the linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{A} \in \mathbb{R}^{t \times N}$ is a matrix whose number of rows is t, the current time. The solution $\boldsymbol{x} \in \mathbb{R}^N$ is assumed to be group sparse [10]: its number of nonzero elements is much smaller than N and the nonzero elements are grouped in few clusters with variable sizes. The environment is time varying, hence the values *and* the positions of the nonzero elements of $\boldsymbol{x}$ may change in time.

The solution is found at each time moment t, after a new equation is appended to the system and the matrix $\boldsymbol{A}$ is exponentially weighted with a forgetting factor λ, by minimizing the typical least-squares criterion

$$J(\boldsymbol{x}) = \|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|^2. \tag{1}$$

The aim of this paper is to introduce the *group* greedy RLS and explore the use of ITC for estimating the sparsity online. Other existing algorithms able to produce group sparse solutions, [11], [12] and [13] (the latter being an adaptive algorithm), are based on convex relaxation techniques and use the $\ell_{1,\infty}$ norm. The difference from the standard sparse problem is that now the nonzero coefficients appear in clusters, not independently. Since their positions are consecutive, it is more efficient to work with groups of nonzeros instead of individual nonzeros. Working on a group level, one has to decide the number of such groups and the size of each group.

The content of the paper is organized as follows. We introduce the greedy group RLS algorithm in Section II. In Section III we show how the ITC can be tailored for the task of estimating online the group sparsity while in Section IV we present a series of simulations giving insight into the performance of the group sparse algorithm and the proposed sparsity selection criteria. Section V contains a brief analysis of the performance for the proposed ITC.

II. Group sparse greedy RLS

For the group sparsity, the non zero coefficients locations are supposed to be found in M blocks of sizes P_i, $i = 1{:}M$.

This work was supported by Tekes FiDiPro—Finland Distinguished Professor Programme and by GETA Finland.

Similarly to the GRLS [4] and OLS [14] algorithms, we employ a greedy strategy to select M_c active columns $\boldsymbol{a}_i$ from $\boldsymbol{A}$ (forming the M active groups) for computing the sparse solution. The column selection is performed on a group level; we assume for now that all groups have the same size P, afterwards we provide a mechanism for splitting the groups up to a minimum size $p \leq P/2$ and for joining them. Also, the group number is considered known. We postpone the description of the online group sparsity estimation via the ITC.

The set $\mathcal{G}_i$ contains the P (consecutive) indices of the nonzero coefficients of the solution belonging to the i-th group (the indices of the columns of $\boldsymbol{A}$ which are used to compute the solution and correspond to nonzero coefficients from $\boldsymbol{x}$). The algorithm maintains a QR factorization with pivoting of the matrix $\boldsymbol{A}$, such that the first columns are those of the active columns (all other remaining columns are called inactive).

At time t, when a new equation arrives, the algorithm performs the following operations

- restores the triangular factor;
- reorders the active groups upon their significance in reducing the criterion (1);
- recomputes the solution.

The choice to include a group $\mathcal{G}_i$ into the active set is made based on (1); the group becomes active if, by adding it to the previously selected groups, decreases with the largest amount the error sum from the criterion $J(\boldsymbol{x})$. This selection strategy also ensures a group order based on the group's contribution in decreasing $J(\boldsymbol{x})$. The algorithm is recursive and, once new input samples are received, reuses the previous group order to minimize the computational cost.

A. Partial QR factorization

To compute the solution we maintain, similarly to [4], a partial QR factorization with pivoting,

$$\begin{array}{rcl} \mathbf{R} & \leftarrow & \prod_i \mathbf{Q}_i{}^T \mathbf{A} \prod_i \mathbf{P}_i \\ \boldsymbol{b} & \leftarrow & \prod_i \mathbf{Q}_i{}^T \mathbf{y}. \end{array} \tag{2}$$

The orthogonal matrices $\mathbf{Q}_i$ produce a superior triangular form for the active columns in $\mathbf{R}$. They are applied successively allowing in place computations of the data matrices $\mathbf{R}$ and $\boldsymbol{b}$ and thus eliminating the need for explicit storage. The matrices $\mathbf{P}_i$ permute the non zero elements of $\boldsymbol{x}$ (and the corresponding columns in $\mathbf{A}$ and $\mathbf{R}$) to the first M_c positions. We always consider the groups to be formed by adjacent columns from the original matrix $\mathbf{A}$ before any permutations are applied. The notation $\mathbf{R}_{\mathcal{G}_i}$ implies the column subset form $\mathbf{R}$ associated with the group $\mathcal{G}_i$ (the permutations can change its actual position in $\mathbf{R}$); $\mathbf{R}_{i:j}$ contains the columns from $\mathbf{R}$ on positions $i{:}j$; $\mathbf{R}_{\mathcal{G}_i,\mathcal{G}_j}$ selects a partition defined by the rows $\mathcal{G}_i$ and the columns $\mathcal{G}_j$.

Any k-sparse solution, $k \leq M_c$, is computed by solving the upper triangular system defined by $\mathbf{R}_{1:k,1:k}$ and $\boldsymbol{b}_{1:k}$. Since only the first M_c rows are used, the information contained in the remaining rows is stored in the form of the scalar products

$$\begin{array}{rcl} \boldsymbol{\Psi} & = & \mathbf{R}^T_{M_c+1:t,M_c+1:N} \mathbf{R}_{M_c+1:t,M_c+1:N} \\ \boldsymbol{\phi} & = & \mathbf{R}^T_{M_c+1:t,M_c+1:N} \boldsymbol{b}_{M_c+1:t}, \end{array} \tag{3}$$

and it is deleted from $\mathbf{R}$ and $\boldsymbol{b}$; the matrices have predetermined sizes, $\mathbf{R} \in \mathbb{R}^{M_c \times N}$ and $\boldsymbol{b} \in \mathbb{R}^{M_c}$ eliminating the need to store the indefinitely long $\mathbf{A}$ and $\mathbf{y}$.

Due to the orthogonal triangularization the criterion becomes the norm of the residual

$$J(\boldsymbol{x}) = \boldsymbol{b}^T_{M_c+1:t} \boldsymbol{b}_{M_c+1:t} \tag{4}$$

and thus the decrease of $J(\boldsymbol{x})$ due to each group of coefficient $\boldsymbol{x}_{\mathcal{G}_i}$ is $\delta_{\mathcal{G}_i} = \|\boldsymbol{b}_{\mathcal{G}_i}\|^2$.Note, this is valid for a given group order, any change in the order can modify the elements of $\boldsymbol{b}_{\mathcal{G}_i}$ and consequently $\delta_{\mathcal{G}_i}$.

B. Basic update

When new input and output data, $\boldsymbol{\alpha}^T$ and β, are available they are included in $\mathbf{A}$ and in $\mathbf{y}$ or correspondingly in $\mathbf{R}$ and $\boldsymbol{b}$,

$$\mathbf{R} \leftarrow \begin{bmatrix} \sqrt{\lambda} \cdot \mathbf{R} \\ \boldsymbol{\alpha}^T \end{bmatrix}, \quad \boldsymbol{b} \leftarrow \begin{bmatrix} \sqrt{\lambda} \cdot \boldsymbol{b} \\ \beta \end{bmatrix}. \tag{5}$$

This however changes the upper triangular structure of the matrix $\mathbf{R}$. Using Givens rotations we zero the first M_c newly introduced elements with the modification of the diagonal elements in $\mathbf{R}$. The remaining $N - M_c$ non-zero elements are included in the scalar products

$$\begin{array}{rcl} \boldsymbol{\Psi} & \leftarrow & \boldsymbol{\Psi} + \mathbf{R}^T_{M_c+1,1:N} \mathbf{R}_{M_c+1,1:N} \\ \boldsymbol{\phi} & \leftarrow & \boldsymbol{\phi} + \mathbf{R}^T_{M_c+1,1:N} \boldsymbol{b}_{M_c+1}, \end{array} \tag{6}$$

and the last row in $\mathbf{R}$ and $\boldsymbol{b}$ is deleted thus restoring the upper triangular structure. More details can be found in [4].

C. Neighbor group permutations

After receiving new data, the group order is prone to change; we assume that the change is slow, allowing the use, once every θ_p samples, of a neighbor permutation strategy that restores the group order (in [4] similar permutations are used element wise, not group wise).

Given two neighbor groups $\mathcal{G}_i$ and $\mathcal{G}_j$ the decision to permute them is based on the decrease of the residual their permutation produces. If we select from $\mathbf{R}$ (and $\boldsymbol{b}$) only the triangular part associated with the two groups we have

$$\begin{bmatrix} \mathbf{R}_{\mathcal{G}_i,\mathcal{G}_i} & \mathbf{R}_{\mathcal{G}_i,\mathcal{G}_j} \\ \mathbf{0} & \mathbf{R}_{\mathcal{G}_j,\mathcal{G}_j} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{\mathcal{G}_i} \\ \boldsymbol{x}_{\mathcal{G}_j} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_{\mathcal{G}_i} \\ \boldsymbol{b}_{\mathcal{G}_j} \end{bmatrix}, \tag{7}$$

where $\mathbf{R}_{\mathcal{G}_i,\mathcal{G}_i}$ and $\mathbf{R}_{\mathcal{G}_j,\mathcal{G}_j}$ are upper triangular matrices. The residual decrease due to selecting first $\mathcal{G}_i$ is $\delta_{\mathcal{G}_i} = \|\boldsymbol{b}_{\mathcal{G}_i}\|^2$.

Any change in the group order affects the output vector $\boldsymbol{b}$ thus we need to compute $\delta_{\mathcal{G}_j}$ after we perform the permutation. By permuting $\mathcal{G}_i$ and $\mathcal{G}_j$ the upper triangular form is destroyed and thus computing $\delta_{\mathcal{G}_j}$ is not immediate. We apply $|\mathcal{G}_j|$ Householder reflectors ($|\mathcal{G}_j|$ is the number of columns in group $\mathcal{G}_j$) to restore the triangular form. Each reflector is computed such that it zeros $|\mathcal{G}_i|$ sub diagonal elements on each column, starting from the left, and modifies the diagonal element. After the triangular form is restored

$$\begin{bmatrix} \mathbf{R}'_{\mathcal{G}_j,\mathcal{G}_j} & \mathbf{R}'_{\mathcal{G}_j,\mathcal{G}_i} \\ \mathbf{0} & \mathbf{R}_{\mathcal{G}_i,\mathcal{G}_i} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}'_{\mathcal{G}_j} \\ \boldsymbol{x}'_{\mathcal{G}_i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}'_{\mathcal{G}_j} \\ \boldsymbol{b}'_{\mathcal{G}_i} \end{bmatrix}, \tag{8}$$

$\delta_{\mathcal{G}_j}$ can be computed, $\delta_{\mathcal{G}_j} = \|\boldsymbol{b}'_{\mathcal{G}_i}\|^2$. Note that, the permuted columns associated with $\mathcal{G}_i$ may not updated now since we only apply the triangularization to compute $\delta_{\mathcal{G}_j}$ and the permutation is temporary. We make it permanent only if

$$\frac{\delta_{\mathcal{G}_j}}{|\mathcal{G}_j|} > \frac{\delta_{\mathcal{G}_i}}{|\mathcal{G}_i|}, \tag{9}$$

and so the reflectors used to obtain it are applied to the whole $\mathbf{R}$ to the right of each zeroed column, not only for the group $\mathcal{G}_j$. Applying each reflector changes only $|\mathcal{G}_i| + 1$ rows.

D. Last group

To allow changes in the active group set, every θ_p samples, any possible inactive group can compete for the last active position. An exhaustive search for all possible inactive groups of size P is performed; smaller groups are allowed only if the previously selected active groups limit the number of adjacent inactive columns.

Let us first examine an alternate way of determining the criterion $J(\boldsymbol{x})$. Consider the partition $\mathcal{G}_i$ of $\mathbf{A}$; we can write

$$\begin{aligned} J(\boldsymbol{x}_{\mathcal{G}_i}) &= (\mathbf{y} - \mathbf{A}_{\mathcal{G}_i}\mathbf{A}^+_{\mathcal{G}_i}\mathbf{y})^T(\mathbf{y} - \mathbf{A}_{\mathcal{G}_i}\mathbf{A}^+_{\mathcal{G}_i}\mathbf{y}) \\ &= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{A}_{\mathcal{G}_i}\mathbf{A}^+_{\mathcal{G}_i}\mathbf{y}, \end{aligned} \tag{10}$$

with $\mathbf{A}^+_{\mathcal{G}_i} = (\mathbf{A}^T_{\mathcal{G}_i}\mathbf{A}_{\mathcal{G}_i})^{-1}\mathbf{A}^T_{\mathcal{G}_i}$. Including the group $\mathcal{G}_i$ in the solution is therefore decreasing the criterion with $\delta_{\mathcal{G}_i} = \mathbf{y}^T\mathbf{A}_{\mathcal{G}_i}\mathbf{A}^+_{\mathcal{G}_i}\mathbf{y}$. This holds for any orthogonal transformation of $\mathbf{A}$ and $\mathbf{y}$, particularly $\mathbf{R}$ and $\boldsymbol{b}$.

To replace the last group $\mathcal{G}_M$ with a new group $\mathcal{G}_i$ ($\mathcal{G}_i$ can overlap with $\mathcal{G}_M$) we require that

$$\frac{\delta_{\mathcal{G}_M}}{|\mathcal{G}_M|} < \max_{\mathcal{G}_i} \frac{\delta_{\mathcal{G}_i}}{|\mathcal{G}_i|}. \tag{11}$$

We compute $\delta_{\mathcal{G}_i} = (\mathbf{R}^T_{\mathcal{G}_M,\mathcal{G}_i}\boldsymbol{b}_{\mathcal{G}_i} + \boldsymbol{\phi}_{\mathcal{G}_i})^T\mathbf{s}$, with $\mathbf{s}$ given by

$$(\mathbf{R}^T_{\mathcal{G}_M,\mathcal{G}_i}\mathbf{R}_{\mathcal{G}_M,\mathcal{G}_i} + \boldsymbol{\Psi}_{\mathcal{G}_i,\mathcal{G}_i})\mathbf{s} = \mathbf{R}^T_{\mathcal{G}_M,\mathcal{G}_i}\boldsymbol{b}_{\mathcal{G}_M} + \boldsymbol{\phi}_{\mathcal{G}_i}, \tag{12}$$

to avoid the inversion from $\mathbf{R}^+_{\mathcal{G}_i}$. Note that we only need the scalar products $\boldsymbol{\Psi}$ and $\boldsymbol{\phi}$ not the whole matrices $\mathbf{R}$ and $\boldsymbol{b}$.

Consider the matrix $\mathbf{R}^T_{\mathcal{G}_M,1:N}\mathbf{R}_{\mathcal{G}_M,1:N} + \boldsymbol{\Psi}$ (and the associated $\mathbf{R}^T_{\mathcal{G}_M,1:N}\boldsymbol{b}_{\mathcal{G}_M} + \boldsymbol{\phi}$) permuted back to the original column order from $\mathbf{A}$; each $P \times P$-diagonal block defines a systems from (12). To find the solution we transform the system to be upper triangular and use a downdate-update procedure [15], [16] to morph between every possible diagonal block.

Let $\mathcal{S}_i = \{i : i + 2P - 1\}$ for any possible i, $\mathbf{U} = \mathbf{R}^T_{\mathcal{G}_M,\mathcal{S}_i}\mathbf{R}_{\mathcal{G}_M,\mathcal{S}_i} + \boldsymbol{\Psi}_{\mathcal{S}_i,\mathcal{S}_i}$ and $\mathbf{d} = \mathbf{R}^T_{\mathcal{G}_M,\mathcal{S}_i}\boldsymbol{b}_{\mathcal{G}_M} + \boldsymbol{\phi}_{\mathcal{S}_i}$. We construct the orthogonal matrix $\mathbf{H}$ and zero, with P reflectors, the subdiagonal elements from $\mathbf{U}$. A solution $\mathbf{s}$ of (12) is then found by solving the upper triangular system defined by $\mathbf{U}_{1:P,1:P}$ and $\mathbf{d}_{1:P}$. All transformations we consider are applied implicitly on all matrices including $\mathbf{H}$.

To find the solution for the next system associated with $\mathcal{S}_{i+1}$, we use a downdate-update procedure to remove the first row and column from the matrix $\mathbf{U}$ and to add a new end row. The downdate requires the use of $P - 1$ rotations to zero the first column of $\mathbf{H}$, starting from the last row and continuing upwards (we zero the first element of the current row and modify the current row and the one immediately above). The downdate and the removal of the first row maintains the upper triangular structure in $\mathbf{U}$ but the removal of the first column does not (subdiagonal elements are introduced). Therefore, we require $P - 1$ additional rotations (applied diagonally from left to right) to zero the subdiagonal elements and to produce a triangular system containing $P - 1$ equations.

Appending a new row changes the group for which the system (12) is defined. After restoring the triangular form with $P - 1$ Givens rotations applied on the matrices like in Section II-B, the solution can be computed easily from the triangular part of the matrices. The same recursion is used to compute all values of $\delta_{\mathcal{G}_i}$ but care should be taken when we encounter the already selected groups and the procedure is restarted. The chosen group is then permuted to the last position and, using P reflectors, the upper triangular form is restored. Note that for the permuted columns the scalar products are non zero and they also need to be updated when the reflectors are applied; more details can be obtained from [4].

E. Split groups

Imposing a strict group size P can degrade the performance if groups of smaller size exist. In what follows we propose an heuristic approach, applied once every θ_s samples, for splitting each group. We decide to split each group creating a new group $\mathcal{G}_{i,-k}$ or $\mathcal{G}_{i,k-}$ (formed from $\mathcal{G}_i$ by taking the first/last k columns) such that the new group may exit the active set via permutations at a later time. The last group is never split because it is used to determine the split point in all the others. Also we limit the smallest group size to p.

We use a criterion similar to (9) applied for the $\boldsymbol{b}_{\mathcal{G}_{i,-k}}$ (or $\boldsymbol{b}_{\mathcal{G}_{i,k-}}$) subgroup and the last active group. Computing the decrease $\delta_{\mathcal{G}_{i,-k}}$ (or $\delta_{\mathcal{G}_{i,k-}}$) is immediate as presented in Section II-C. The computations involving the last active group permutation to the position occupied by the subgroup and the computation of $\delta_{\mathcal{G}_M}$ are more demanding; in (7) the triangular matrix $\mathbf{R}_{\mathcal{G}_j,\mathcal{G}_j}$ is replaced by the tall matrix $\mathbf{R}_{\mathcal{G}_{j:M},\mathcal{G}_M}$ because the groups are not neighbors. We apply a procedure similar to the basic update from Section II-B and compute the $\delta_{\mathcal{G}_M}$ for all permutation positions. Sequentially adding to $\mathbf{R}_{\mathcal{G}_M,\mathcal{G}_M}$ all rows $\mathbf{R}_{k,\mathcal{G}_M}$ from $\mathbf{R}_{\mathcal{G}_{1:M-1},\mathcal{G}_M}$ and restoring the triangular form generates the decrease $\delta_{\mathcal{G}_M}$ for all needed permutation places. Depending on the number of rows added, we use $|\mathcal{G}_M|$ Givens rotations or $|\mathcal{G}_M|$ Householder transforms.

F. Joining groups

Since continuous group splitting can potentially fragment all active groups to the minimum size p, we allow every θ_j samples the joining of the active groups $\mathcal{G}_i$ and $\mathcal{G}_j$, with $i > L$ and $j \leq L$, if they contain adjacent columns. This involves permuting $\mathcal{G}_i$ to be adjacent to $\mathcal{G}_j$ and restoring the triangular form. For efficient computation the restoration of the triangular form is done only after all possible groups are joined (and the corresponding permutations are performed). The joining of groups is only necessary in case when new support positions arise to allow for low group fragmentation and should not introduce any performance gains if the support does not chance.

G. Complexity

The algorithm is more complex than GRLS due to the block level operations. The most demanding operation is the restoration of the triangular form. This is especially computational intensive when the groups are joined since the whole triangular form may be damaged; we suppose that the group joining is performed infrequently. Taking all into consideration the worst case complexity is $O(P(N-M_c)^2)+O(P^2N)$ or about P times that of the base GRLS algorithm. Due to the greedy nature the average complexity is usually lower because the group order and number may not change frequently.

III. Information theoretic criteria

The group order introduced by the permutation strategy can be exploited to allow the online estimation of the sparsity level with the use of the PLS and BIC criteria. Since the solution has a double granularity given by the columns and the groups, it can be exploited to produce column or group level criteria. While the computation of the column aware ITC is similar to [4], it is used to infer a group level sparsity estimation instead. For the group aware ITC, the estimate is computed with the groups as the atomic entities.

A. Column aware ITC

The column level PLS criterion at time t is given by

$$\mathbf{PLS}_t^{(c)}(k)=\sum_{\tau=0}^{t}\gamma^{t-\tau}\epsilon_\tau(k)^2 \tag{13}$$

where the a priori error $\epsilon_\tau(k)$ from time τ is computed using the k sparse solution $\boldsymbol{x}$ (the solution computed using the first k columns) from time $\tau-1$. The exponentially decaying factor $\gamma^{t-\tau}$ windows the past error allowing for variations in the sparsity level estimate.

The BIC criterion uses the criterion $J(\boldsymbol{x})$ for each k sparse solution $\boldsymbol{x}$ and combines it with the effective number of samples n_t used to compute $\boldsymbol{x}$, $n_t=\sum_{\tau=0}^{t}\lambda^\tau$. It produces the expression

$$\mathbf{BIC}_t^{(c)}(k)=n_t\ln J(\boldsymbol{x})+(k+1)\ln n_t. \tag{14}$$

The sparsity estimate is the minimizer of the criterion chosen, $L^{(c)}=\min_k \mathbf{PLS}_t^{(c)}(k)$ or $L^{(c)}=\min_k \mathbf{BIC}_t^{(c)}(k)$. To adapt the sparsity estimate for our group problem, we compute the group number estimate L by finding the group containing the column $L^{(c)}$, $L=k$, such that $L^{(c)}\in\mathcal{G}_k$.

B. Group aware ITC

The ordering performed with the permutations is done at the group level without any inherent in-group column order. Because of this, the column sparsity estimation strategy can potentially provide inaccurate estimates. To alleviate this we propose the use of a group aware form for the ITC. In the BIC case the changes are straightforward, the criterion is computed for solutions $\boldsymbol{x}$ produced by the whole groups $\mathcal{G}_{1:k}$,

$$\begin{aligned}\mathbf{BIC}_t^{(g)}(k)&=n_t\ln J(\boldsymbol{x})+\left(1+\textstyle\sum_{i=1}^{k}|\mathcal{G}_i|\right)\ln n_t=\\&=\mathbf{BIC}_t^{(c)}(\textstyle\sum_{i=1}^{k}|\mathcal{G}_i|).\end{aligned} \tag{15}$$

Note that we consider the penalty for the model complexity $(1+\sum_{i=1}^{k}|\mathcal{G}_i|)\ln n_t$ to depend on the number of columns in the groups.

The PLS has a temporal dependance on the previous solutions and the direct generalization for the group aware case does not maintain the group atomicity. Due to this, we propose several versions. The simplest uses the current group structure $\mathcal{G}_{1:M}$ from time t and computes the solution at group level,

$$\begin{aligned}\mathbf{PLS}_t^{(g)}(k)&=\textstyle\sum_{\tau=0}^{t}\gamma^{t-\tau}\epsilon_\tau\left(\sum_{i=1}^{k}|\mathcal{G}_i|\right)^2=\\&=\mathbf{PLS}_t^{(c)}\left(\textstyle\sum_{i=1}^{k}|\mathcal{G}_i|\right).\end{aligned} \tag{16}$$

If we consider the groups to be atomic, then we use the group structure $\mathcal{G}_{1:M}^{\tau}$ obtained at each time instant τ. In this case the group level PLS is given by

$$\mathbf{PLS}_t^{(g_a)}(k)=\sum_{\tau=0}^{t}\gamma^{t-\tau}\epsilon_\tau\left(\sum_{i=1}^{k}|\mathcal{G}_i^\tau|\right)^2. \tag{17}$$

The splitting of groups and the different size of the groups potentially adversely influence the ability of $\mathbf{PLS}_t^{(g_a)}$ to produce a good sparsity estimate. To overcome these problems we propose a weighted PLS criterion,

$$\mathbf{PLS}_t^{(g_w)}(k)=\sum_{\tau=0}^{t}\frac{\gamma^{t-\tau}}{|\mathcal{G}_k^\tau|}\epsilon_\tau\left(\sum_{i=1}^{k}|\mathcal{G}_i^\tau|\right)^2. \tag{18}$$

Special care should be taken in the case when a group $\mathcal{G}_i$ is split; the PLS criterion for the first new formed group is approximated as the mean of $\mathbf{PLS}_t^{(g_w)}(i-1)$ and $\mathbf{PLS}_t^{(g_w)}(i)$ where $\mathbf{PLS}_t^{(g_w)}(0)=\boldsymbol{b}^T\boldsymbol{b}$ and the criterion for the second group remains the same as for group $\mathcal{G}_i$.

If the weighting is not performed directly like in (18), the influence of different group sizes is mitigated when the permutations are performed by using a weighted average $\mathbf{PLS}^{(g_{wa})}$ of the PLS for the affected groups. When we split a group a similar weighted average is used.

IV. Simulations

The behavior of the group GRLS algorithm using the different ITC proposed in the previous section is studied for an FIR channel identification problem. The variation of the true coefficients is sinusoidal given by

$$\tilde{\boldsymbol{x}}_i=\alpha_i cos(2\pi f T_s+\beta_i), \tag{19}$$

with the amplitude α_i and the phase β_i distribute uniformly in $[0.05,1]$ and $[0,2\pi]$. We consider the filter length $N=180$ and we generate $M_t=2$ groups, containing P_t nonzero elements, randomly on a grid with step P_t. The inputs are generated random according to $\mathcal{N}(0,1)$ and the outputs are corrupted by white noise with $\sigma^2=0.01$. The filter is normed such that $E\{||\tilde{x}||^2\}=1$. We define the performance measure as

$$\text{MSE}=E\{||\tilde{\boldsymbol{x}}-\boldsymbol{x}||^2\} \tag{20}$$

and we estimate it by averaging 300 runs.

We use two versions of the algorithms employing the online groups sparsity estimation criteria. The first version (named GRLS-G$_g$) knows the grid distribution of the groups (in the

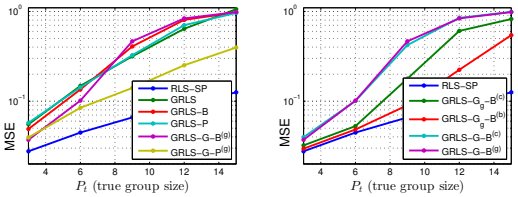


Fig. 1. Average MSE as a function of P_t; $M_t = 2$, $fT_s = 0.002$.

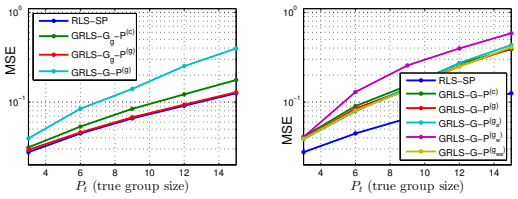


Fig. 2. Average MSE as a function of P_t; $M_t = 2$, $fT_s = 0.002$.

search for the last group M) and uses the group size fixed to $P = P_t$. The second version (named GRLS-G) uses a exhaustive sweep over all possible groups and the split and join strategies with $\theta_s = 20$ and $\theta_j = 100$, respectively. It also uses the group size $P = P_t$ but can accommodate any other size. The algorithms use either the PLS or the BIC criteria presented in Section III (the names end with -P for PLS and -B for BIC). For all algorithms $p = 2$, $\theta_p = 2$ and $\Delta_g = 2$.

Additionally, we use the column GRLS algorithms from [4] to illustrate the performance in the group unaware case; the algorithm GRLS is the algorithms that knows the true number of columns but not their positions while the fully adaptive algorithms named GRLS-B and GRLS-P use BIC and PLS to estimate the sparsity level. The algorithms use the parameter $\Delta = 10$ and $\tau_0 = 2$. As e reference we also include the sparsity aware RLS algorithm (RLS-SP) which knows both the column positions and the number of non zero coefficients.

Fig. 1 and 2 contain the MSE for the studied algorithm for a high variation speed, $fT_s = 0.002$, as a function of the true group size P_t. Similarly, Fig. 3 and 4 show the MSE for a low variation speed, $fT_s = 0.0002$. In both cases $M_t = 2$ and the forgetting factors are $\lambda = \gamma = 0.9$ and $\lambda = \gamma = 0.96$, respectively.

Since the parameter γ from the PLS criterion is not necessary linked with the global forgetting factor λ, we study the MSE as a function of γ for the best of the PLS criteria in Fig. 5.

The convergence speed can be evaluated from Fig. 6. The test is performed for $fT_s = 0.0002$, $\lambda = 0.96$ and a different group sparsity setup. The filter length is now $N = 240$, the number of groups is $M_t = 3$ and the group size is $P_t = 8$. We only include one selection criterion; the others that achieve comparable MSE are performing similarly. The group algorithm has better convergence speed than the group unaware GRLS and GRLS-P.

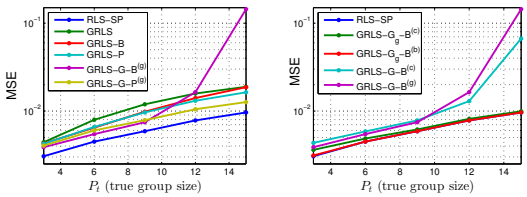


Fig. 3. Average MSE as a function of P_t; $M_t = 2$, $fT_s = 0.0002$.

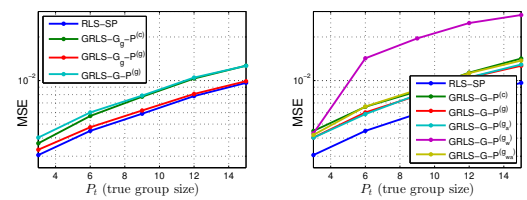


Fig. 4. Average MSE as a function of P_t; $M_t = 2$, $fT_s = 0.0002$.

V. Performance Assessment

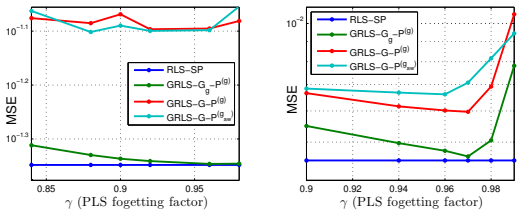
The simulations suggest that the performance of the group algorithms is generally better than that of the groups unaware counterparts. The ability of the ITC to select the right group sparsity varies however among the two BIC and the five PLS criteria. On average the PLS criteria are superior if compared to the BIC ones. In what follows we present a detailed performance analysis and identify the possible causes for performance degradation.

A. BIC performance

In general the BIC criteria provide similar performance to that of the PLS criteria for low group sizes but show fast performance degradation as the size increases. Since BIC depends solely on the penalized residual from current time, its performance is greatly influenced by the column and the group order. Thus if a group contains columns which do not correspond to nonzeros, BIC can produce spurious estimates. Furthermore it has the tendency to underestimate the sparsity level, especially for higher levels.

As can be seen from Fig. 1, for a fast variation speed and small groups BIC provides good performance approaching that of the sparsity aware RLS. As the group size increases the performance degrades fast. The grid unaware algorithms are still able to find parts of the support but they also include many false columns resulting in the BIC being unable to estimate the correct sparsity. The grid aware algorithms, since they can find the true support easier, produce better results but the BIC estimate is still poor for large groups underestimating the sparsity level.

For the lower variation speed presented in Fig. 3, it is notable that the grid aware algorithms provide near oracle performance. The other BIC algorithms still show performance degradation for large groups but, due to the lower variation speed of the coefficient, they maintain their ability to estimate the support for group sizes up to 10.



(a) $fT_s = 0.002$, $\lambda = 0.90$ (b) $fT_s = 0.0002$, $\lambda = 0.96$

Fig. 5. Average MSE as a function of γ for the PLS criterion; $M_t = 2$.

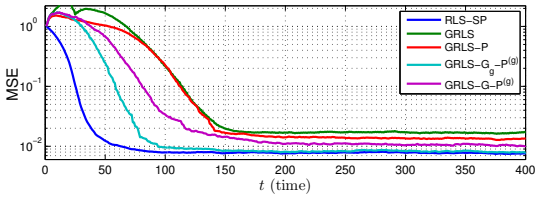


Fig. 6. MSE for $N = 240$, $M_t = 3$ and $P_t = 8$.

B. PLS performance

The PLS criteria achieve better results than the BIC and are consistent for all group sizes. Since the PLS depends on past estimation errors, having different group sizes generates difficulties for the group level criteria. Among the ones proposed, the best performers are the naive group level $\mathbf{PLS}^{(g)}$ and the weighted average $\mathbf{PLS}^{(g_{wa})}$. The group weighted criterion $\mathbf{PLS}^{(g_w)}$ has the worst performance but is still robust to changes in the group sizes. For the grid aware algorithm all the group level criteria are equivalent and are treated as such.

The performance of the grid aware algorithm using $\mathbf{PLS}^{(g)}$ is close to that of the sparsity informed RLS suggesting almost perfect support and sparsity level estimation. The column based criterion $\mathbf{PLS}^{(c)}$ provides slightly inferior estimates.

While most of the group level PLS criteria (besides $\mathbf{PLS}^{(g_w)}$) produce similar results, for the grid unaware algorithms, $\mathbf{PLS}^{(g)}$ is the best by a small margin. $\mathbf{PLS}^{(g_a)}$ is very close but sometimes exhibits large error spikes, due to permutations between small and large groups, although its average performance is good.

The algorithms can benefit to some degree from a different exponential window applied to the estimation error involved in the computation of the PLS. Since the support size is not necessary linked to the variation speed of the coefficients, using a $\gamma \neq \lambda$ can improve the sparsity level estimation. Ideally γ needs to be related to the support size variation rate. In Fig. 5, the support remains the same during the simulation and a higher γ produces improved performance. A too large γ however lowers the convergence speed and may produce worse estimates because it favors too much the past. A smaller γ favors an estimate based on more recent data but can be easier influenced by noise; as γ grows more data is included in the decision and any perturbations are averaged out. Thus γ can be considered a tradeoff between consistency and adaptation speed. In any case, the conservative choice $\gamma = \lambda$ appears to be robust; in all all simulations, this choice provides a performance that is near the best attainable by a different γ.

VI. Conclusions

We have proposed a group aware sparse algorithm based on the OLS and GRLS algorithms. The method uses orthogonal transformations to maintain a partial QR factorization. A neighbor permutation strategy keeps the group order and allows the use of the ITC to estimate online the number of solution groups. Additionally a group slitting and joining strategy was introduced. We studied several group level selection strategies, based on ITC, for the task of online group sparsity estimation. The simulations confirm the ability of the ITC to estimate online the group sparsity level and suggest that the PLS is more robust than BIC. On average, due to group operations, it is P times more computationally intensive than GRLS but has improved performance; the added complexity due to ITC is very low.

References

[1] Y. Chen, Y. Gu, and A. Hero III, "Sparse LMS for System Identification," in *Int. Conf. Acoustics, Speech, Signal Proc.*, 2009, pp. 3125–3128.

[2] D. Angelosante, J. Bazerque, and G. Giannakis, "Online Adaptive Estimation of Sparse Signals: Where RLS Meets the ℓ_1-Norm," *IEEE Trans. Signal Proc.*, vol. 58, no. 7, pp. 3436–3447, July 2010.

[3] B. Babadi, N. Kalouptsidis, and V. Tarokh, "SPARLS: The Sparse RLS Algorithm," *IEEE Trans. Signal Proc.*, vol. 58, no. 8, pp. 4013–4025, 2010.

[4] B. Dumitrescu, A. Onose, P. Helin, and I. Tăbuş, "Greedy Sparse RLS," *IEEE Trans. Signal Proc.*, vol. 60, no. 5, pp. 2194–2207, 2012.

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

[6] Y. Kopsinis, K. Slavakis, and S. Theodoridis, "Online Sparse System Identification and Signal Reconstruction Using Projections Onto Weighted ℓ_1 Balls," *IEEE Trans. Signal Proc.*, vol. 59, no. 3, pp. 936–952, Mar. 2011.

[7] L. Vega, H. Rey, J. Benesty, and S. Tressens, "A Family of Robust Algorithms Exploiting Sparsity in Adaptive Filters," *IEEE Trans. Audio Speech Lang. Proc.*, vol. 17, no. 4, pp. 572–581, May 2009.

[8] J. Rissanen, "Order Estimation by Accumulated Prediction Errors," *J. Appl. Probab.*, vol. 23, pp. 55–61, 1986.

[9] G. Schwarz, "Estimating the Dimension of a Model," *Ann. Stat.*, vol. 6, no. 2, pp. 461–464, 1978.

[10] Y. Eldar, P. Kuppinger, and H. Bolcskei, "Block-Sparse Signals: Uncertainty Relations and Efficient Recovery," *IEEE Trans. Sign. Proc.*, vol. 58, no. 6, pp. 3042–3054, June 2010.

[11] S. Negahban and M. Wainwright, "Simultaneous support recovery in high dimensions: Benefits and perils of block ℓ_1/ℓ_∞-regularization," *IEEE Trans. Info. Theory*, vol. 57, no. 6, pp. 3841 –3863, June 2011.

[12] S. Wright, R. Nowak, and M. Figueiredo, "Sparse Reconstruction by Separable Approximation," *IEEE Trans. Sign. Proc.*, vol. 57, no. 7, pp. 2479 –2493, July 2009.

[13] Y. Chen and A. Hero III, "Recursive $\ell_{1,\infty}$ Group Lasso," *IEEE Trans. Signal Proc.*, vol. 60, no. 8, pp. 3978–3987, Aug. 2012.

[14] S. Chen, S. Billings, and W. Luo, "Orthogonal Least Squares Methods and Their Application to Non-Linear System Identification," *Int. J. Control*, vol. 50, pp. 1873–1896, 1989.

[15] M. Mahon, L. Sibul, and H. Valenzuela, "A Sliding Window Update for the Basis Matrix of the QR Decomposition," *IEEE Trans. Signal Proc.*, vol. 41, no. 5, pp. 1951–1953, May 1993.

[16] A. Onose, B. Dumitrescu, and I. Tăbuş, "Sliding Window Greedy RLS for Sparse Filters," in *ICASSP*, May 2011, pp. 3916–3919.

Afterword

One kind word can warm three winter months.
`Japanese proverb`

You are never truly prepared when you move to a new place. There are always a lot of small things that have the potential to amaze or shock you. Among all the good and the bad, there will always be kind people, friends whose sympathy and support will motivate you to carry on.

I wish to thank everybody who 'was there' during the *journey* that led me, from my early days of school in Bârlad and at the university in Bucharest, to Tampere and to writing this thesis[1]: my good friends and colleagues that I met during all my time spent in school; my family; all my professors, especially to those that had a great influence on me and even those that I didn't really like back then; the mountain-loving friends with whom I traveled much of the Carpathian Mountains every summer; the friends with whom I spent some wonderful times at the seaside or in the Danube Delta; a particular friend for offering me a place to stay during some tough times in my second university year; all my rock-loving friends with whom I went to plenty of concerts and had such a good time; my roommates and friends from in and around 'Regie'; the 'amiq-s' for a very flexible but motivating work environment that allowed me to follow my dreams[2]; all those with whom I shared a beer over the years; all the 'ciorba-s' for a great summer trip in the Carpathians and to the Black Sea; the friend, without whom, this thesis would have had 100 commas less; the 'dai' and 'bai' for an intriguing trip in the Himalayas and in a particularly rainy jungle; all my friends for all the sauna and 'avanto' that made the long, dark and cold Finnish winter more enjoyable; the 'Monday-then-Thuesday' 'beer-then-pepsi' evening people; my friends for all the football, floorball, badminton, volleyball we played together and for trying to teach me how to skate and play hokey; those who trained for and suffered with me during the marathon and two half-marathons I managed to run; all the friends for the nice camping/forest trips we made; my flatmates from Mikontalo, well, most of them; the board game and computer game geeks; a typically troubled policeman who shall not be named; all those who endured the awful Juvenes food and lived to have a coffee with me in Café Rom afterward; one friend who has sworn never

[1] In no particular order.
[2] I had to resign twice to do so but that's a different story.

to return to Finland; the people I met during the conference trips I made in Japan and the Czech Republic; my friends and colleagues from GETA and all the GETA staff for the wonderful boat seminars and meetings; the friends, all of them with their peculiar ways ranging from potato maniacs to chilly or crêpe fanatics, who cooked such wonderful food during the years; all the 'bastards' who enjoyed our cooking but never cooked anything in return; two particular Chinese friends that thought me to see things differently; my friends who kept in touch with me despite the long distance between us; the friends who found some time to come and visit me in Tampere; everybody else who I, deliberately or not, forgot to mention.

Thank you all!
Alex

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-3282-5
ISSN 1459-2045