# Flexible Multi-layer Sparse Approximations of Matrices and Applications

Luc Le Magoarou, Rémi Gribonval

# Flexible Multi-layer Sparse Approximations of Matrices and Applications

Luc Le Magoarou, Rémi Gribonval, *Fellow, IEEE*

***Abstract*—The computational cost of many signal processing and machine learning techniques is often dominated by the cost of applying certain linear operators to high-dimensional vectors. This paper introduces an algorithm aimed at reducing the complexity of applying linear operators in high dimension by approximately factorizing the corresponding matrix into few sparse factors. The approach relies on recent advances in non-convex optimization. It is first explained and analyzed in details and then demonstrated experimentally on various problems including dictionary learning for image denoising, and the approximation of large matrices arising in inverse problems.**

***Index Terms*—Sparse representations, fast algorithms, dictionary learning, low complexity, image denoising, inverse problems.**

## I. Introduction

SPARSITY has been at the heart of a plethora of signal processing and data analysis techniques over the last two decades. These techniques usually impose that the objects of interest be sparse in a certain domain. They owe their success to the fact that sparse objects are easier to manipulate and more prone to interpretation than dense ones especially in high dimension. However, to efficiently manipulate high-dimensional data, it is not sufficient to rely on sparse objects: efficient operators are also needed to manipulate these objects.

The $n$-dimensional Discrete Fourier Transform (DFT) is certainly the most well known linear operator with an efficient implementation: the Fast Fourier Transform (FFT) [3], allows to apply the operator in $\mathcal{O}(n \log n)$ arithmetic operations instead of $\mathcal{O}(n^2)$ in its dense form. Similar complexity savings have been achieved for other widely used operators such as the Hadamard transform [4], the Discrete Cosine Transform (DCT) [5] or the Discrete Wavelet Transform (DWT) [6]. For all these fast linear transforms, the matrix $\mathbf{A}$ corresponding to the dense form of the operator admits a *multi-layer sparse* expression,

$$\mathbf{A} = \prod_{j=1}^{J} \mathbf{S}_j, \tag{1}$$

corresponding to a multi-layer factorization[1] into a small number $J$ of sparse factors $\mathbf{S}_j$. Following the definition of a linear algorithm given in [7], this multi-layer sparse factorization is actually the natural representation of any fast linear transform.

Luc Le Magoarou (luc.le-magoarou@inria.fr) and Rémi Gribonval (remi.gribonval@inria.fr) are both with Inria, Rennes, France, PANAMA team. This work was supported in part by the European Research Council, PLEASE project (ERC-StG- 2011-277906). Parts of this work have been presented at the conferences ICASSP 2015 [1] and EUSIPCO 2015 [2].

[1]The product being taken from right to left: $\prod_{j=1}^{J} \mathbf{S}_j = \mathbf{S}_J \cdots \mathbf{S}_1$

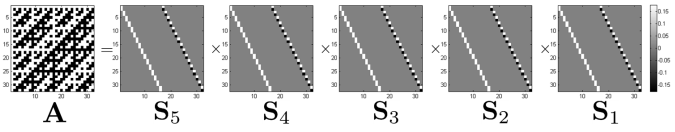


Fig. 1. The Hadamard matrix of size $n \times n$ with $n = 32$ (left) and its factorization. The matrix is totally dense so that the naive storage and multiplication cost $\mathcal{O}(n^2 = 1024)$. On the other hand, we show the factorization of the matrix into $\log_2(n) = 5$ factors, each having $2n = 64$ non-zero entries, so that the storage and multiplication in the factorized form cost $\mathcal{O}(2n \log_2(n) = 320)$.

For example each step of the butterfly radix-2 FFT can be seen as the multiplication by a sparse matrix having only two non-zero entries per row and per column. This fact is further illustrated in the case of the Hadamard transform on Figure 1. For other examples, see e.g. [1, Appendix A].

Inspired by these widely used transforms, our objective is to find approximations of operators of interest encountered in concrete applications, as products of sparse matrices as in (1). Such approximations will be called *Flexible Approximate MUlti-layer Sparse Transforms (FA$\mu$ST)*.

As a primary example of potential application of such approximations, consider *linear inverse problems*, where data and model parameters are linked through a linear operator. State of the art algorithms addressing such problems with sparse regularization [8]–[12] are known to heavily rely on matrix-vector products involving both this operator and its adjoint. As illustrated in Section V on a biomedical inverse problem, replacing the operator by an accurate FA$\mu$ST has the potential to substantially accelerate these methods.

To choose a regularizer for inverse problems, *dictionary learning* is a common method used to learn the domain in which some training data admits a sparse representation [13]. Its applicability is however also somewhat limited by the need to compute many matrix-vector products involving the learned dictionary and its adjoint, which are in general dense matrices. We will see that recent approaches to learn fast dictionaries [14], [15] can be seen as special cases of the *FA$\mu$ST dictionary learning* approach developed in Section VI, where the learned dictionaries are constrained to be FA$\mu$ST.

Beyond the above considered examples, any task where it is required to apply a linear operator in high dimension would obviously benefit from a FA$\mu$ST corresponding to the considered operator. For example, in the emerging area of *signal processing on graphs* [16], novel definitions of usual operators such as the Fourier or wavelet transforms have been introduced. They have no known general sparse forms, and consequently no associated fast algorithms. Finding multi-

layer sparse approximations of these usual operators on graphs would certainly boost the dissemination and impact of graph signal processing techniques.

**Objective.** The quest for multi-layer sparse approximations of large linear operators, which is the core objective of this paper, actually amounts to a matrix factorization problem, where the matrix $\mathbf{A}$ corresponding to the dense form of the operator is to be decomposed into the product of sparse factors $\mathbf{S}_j$, so as to satisfy an approximate form of (1).

**Contributions.** This paper substantially extends the preliminary work started in [1], [2] and [17], both on the theoretical and experimental sides, with the following contributions:

- A general framework for multi-layer sparse approximation (MSA) is introduced, that allows to incorporate various constraints on the sought sparse form;
- Recent advances in non-convex optimization [18] are exploited to tackle the resulting non-convex optimization problem with local convergence guarantees;
- A heuristic hierarchical factorization algorithm leveraging these optimization techniques is proposed, that achieves factorizations empirically stable to initialization;
- The versatility of the framework is illustrated with extensive experiments on two showcase applications, linear inverse problems and dictionary learning, demonstrating its practical benefits.

The remaining of the paper is organized as follows. The problem is formulated, related to prior art and the expected benefits of FA$\mu$STs are systematically explained in Section II. A general optimization framework for the induced matrix factorization problem is introduced in Section III and Section IV, and as a first illustration we demonstrate that it is possible to *reverse-engineer* the Hadamard transform. Several applications and experiments on various tasks, illustrating the versatility of the proposed approach are performed in sections V and VI.

## II. Problem formulation

**Notation.** Throughout this paper, matrices are denoted by bold upper-case letters: $\mathbf{A}$; vectors by bold lower-case letters: $\mathbf{a}$; the $i$th column of a matrix $\mathbf{A}$ by: $\mathbf{a}_i$; and sets by calligraphic symbols: $\mathcal{A}$. The standard vectorization operator is denoted by vec$(\cdot)$. The $\ell_0$-norm is denoted by $\|\cdot\|_0$ (it counts the number of non-zero entries), $\|\cdot\|_F$ denotes the Frobenius norm, and $\|\cdot\|_2$ the spectral norm. By abuse of notations, $\|\mathbf{A}\|_0 = \|\text{vec}(\mathbf{A})\|_0$. The identity matrix is denoted $\mathbf{Id}$.

### A. Objective

The goal of this paper is to introduce a method to get a FA$\mu$ST associated to an operator of interest. Consider a linear operator corresponding to the matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$. The objective is to find sparse factors $\mathbf{S}_j \in \mathbb{R}^{a_{j+1}\times a_j}, j \in \{1 \dots J\}$ with $a_1 = n$ and $a_{J+1} = m$ such that $\mathbf{A} \approx \prod_{j=1}^{J} \mathbf{S}_j$. This naturally leads to an optimization problem of the form:

$$\underset{\mathbf{S}_1,\dots,\mathbf{S}_J}{\text{Minimize}} \quad \underbrace{\left\|\mathbf{A} - \prod_{j=1}^{J} \mathbf{S}_j\right\|^2}_{\text{Data fidelity}} + \underbrace{\sum_{j=1}^{J} g_j(\mathbf{S}_j)}_{\text{Sparsity-inducing penalty}} , \tag{2}$$

to trade-off data fidelity and sparsity of the factors.

### B. Expected benefits of FA$\mu$STs

A multi-layer sparse approximation of an operator $\mathbf{A}$ brings several benefits, provided the *relative complexity* of the factorized form is small with respect to the dimensions of $\mathbf{A}$. For the sake of conciseness, let us introduce $s_j = \|\mathbf{S}_j\|_0$ the total amount of non-zero entries in the $j$th factor, and $s_{tot} = \sum_{j=1}^{J} s_j$ the total number of non-zero entries in the whole factorization.

**Definition II.1.** *The Relative Complexity (abbreviated RC) is the ratio between the total number of non-zero entries in the FA$\mu$ST and the number of non-zero entries of $\mathbf{A}$:*

$$RC := \frac{s_{tot}}{\|\mathbf{A}\|_0}. \tag{3}$$

*It is also interesting to introduce the Relative Complexity Gain (RCG), which is simply the inverse of the Relative Complexity ($RCG = 1/RC$).*

The aforementioned condition for the factorized form to be beneficial writes: RC $\ll 1$ or equivalently RCG $\gg 1$.

FA$\mu$STs reduce computational costs in all aspects of their manipulation, namely a lower *Storage cost*, a higher *Speed of multiplication* and an improved *Statistical significance*.

*1) Storage cost:* Using the Coordinate list (COO) storage paradigm [19], one can store a FA$\mu$ST using $\mathcal{O}(s_{tot})$ floats and integers. Indeed each non-zero entry (float) in the factorization can be located using three integers (one for the factor, one for the row and one for the column), which makes $s_{tot}$ floats and $3s_{tot}$ integers to store. One needs also $J+1$ supplementary integers to denote the size of the factors $a_1$ to $a_{J+1}$. In summary the storage gain is of the order of RCG.

*2) Speed of multiplication:* Applying the FA$\mu$ST or its transpose to a vector $\mathbf{v} \in \mathbb{R}^n$ can be easily seen to require at most $\mathcal{O}(s_{tot})$ floating point operations (flops), instead of $\mathcal{O}(mn)$ for a classical dense operator of same dimension, so the computational gain is, like the storage gain, of the order of RCG.

*3) Statistical significance:* Another interesting though less obvious benefit of FA$\mu$STs over dense operators arises when the operator has to be estimated from training data as in dictionary learning. In this case the reduced number of parameters to learn –$\mathcal{O}(s_{tot})$ compared to $\mathcal{O}(mn)$ for dense operators– leads to better statistical properties. More specifically, the sample complexity is reduced [20], and better generalization properties are expected. The sample complexity gain is of the order of RCG, as will be shown in the case of dictionary learning. The impact of these gains will be illustrated experimentally in section VI on image denoising with learned dictionaries.

### C. Related work

Similar matrix factorization problems have been studied in several domains. Some are very classical tools from numerical linear algebra, such as the truncated SVD, while other emerged more recently in signal processing and machine learning.

*1) The truncated SVD:* To reduce the computational complexity of a linear operator, the most classical approach is perhaps to compute a low-rank approximations with the truncated SVD. Figure 2 compares the approximation-complexity

trade-offs achieved via a low-rank approximation (truncated SVD) and via a multi-layer sparse approximation, on a forward operator associated to an MEG inverse problem. The truncated SVD and four FA$\mu$STs computed using different configurations (more details in Section V - Figure 8) are compared in terms of relative operator norm error: $\|\mathbf{A} - \hat{\mathbf{A}}\|_2 / \|\mathbf{A}\|_2$. It is readily observed that the FA$\mu$STs achieve significantly better complexity/error trade-offs.

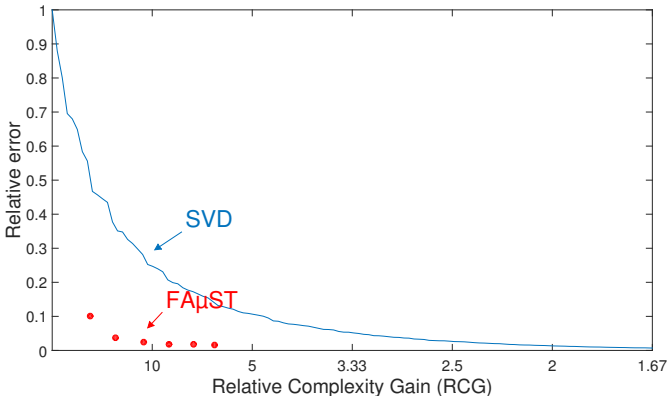


Fig. 2. Comparison between four multi-layer sparse approximations corresponding to different configurations and the truncated SVD. A $204 \times 8193$ matrix associated to an MEG inverse problem is used for this example.

*2) Local low-rank approximations:* Given the limitations of global low-rank approximation by truncated SVD illustrated in Figure 2, the numerical linear algebra community has developed *local* approximations of integral operators by low-rank patches. The patches are built according to the structure of the considered kernel, see [21] and references therein for an introduction to the topic, and [22] for precise examples in the case of Fourier integral operators. This can be seen as approximating the operator by a FA$\mu$ST, where the support of each factor is determined analytically. The approach proposed in this paper is data-driven rather than analytic.

*3) Dictionary learning:* Given a collection of training vectors $\mathbf{y}_\ell$, $1 \leq \ell \leq L$ gathered as the columns of a matrix $\mathbf{Y}$, the objective of dictionary learning [13], [23] is to approximate $\mathbf{Y}$ by the product of a dictionary $\mathbf{D}$ and a coefficients matrix $\mathbf{\Gamma}$ with sparse columns, $\mathbf{Y} \approx \mathbf{D}\mathbf{\Gamma}$.

To learn dictionaries with improved computational efficiency, two main lines of work have begun to explore approaches related to multi-layer sparse approximation. In [14], the authors propose the sparse-KSVD algorithm (KSVDS) to learn a dictionary whose atoms are sparse linear combinations of atoms of a so-called *base dictionary* $\mathbf{D}_{\text{base}}$. The base dictionary should be associated with a fast algorithm (in practice, this means that it is a FA$\mu$ST) so that the whole learned dictionary is itself a FA$\mu$ST. It can be seen as having the $J-1$ leftmost factors fixed in (2), their product being precisely $\mathbf{D}_{\text{base}}$, while the first factor $\mathbf{S}_1$ is the sparse representation of the dictionary over the base dictionary, i.e., $\mathbf{D} = \mathbf{D}_{\text{base}}\mathbf{S}_1$.

A limitation of the sparse-KSVD formulation is that the learned dictionary is highly biased toward the base dictionary, which decreases adaptability to the training data. In [15], the authors propose to learn a dictionary in which each atom is the composition of several circular convolutions using sparse kernels with known supports, so that the dictionary is a sparse operator that is fast to manipulate. This problem can be seen as (2), with the penalties $g_j$s associated to the $J-1$ leftmost factors imposing sparse circulant matrices with prescribed supports. This formulation is powerful, as demonstrated in [15], but limited in nature to the case where the dictionary is well approximated by a product of sparse circulant matrices, and requires knowledge of the supports of the sparse factors.

*4) Inverse problems:* In the context of sparse regularization of linear inverse problems, one is given a signal $\mathbf{y}$ and a measurement matrix $\mathbf{M}$ and wishes to compute a sparse code $\boldsymbol{\gamma}$ such that $\mathbf{y} \approx \mathbf{M}\boldsymbol{\gamma}$, see e.g. [24]. Most modern sparse solvers rely on some form of iterative thresholding and heavily rely on matrix-vector products with the measurement matrix and its transpose. Imposing –and adjusting– a FA$\mu$ST structure to approximate these matrices as proposed here has the potential to further accelerate these methods through fast matrix-vector multiplications. This is also likely to bring additional speedups to recent approaches accelerating iterative sparse solvers through learning [25]–[27].

*5) Statistics – factor analysis:* A related problem is to approximately diagonalize a covariance matrix by a unitary matrix in factorized form (1), which can be addressed greedily [28], [29] using a fixed number of elementary Givens rotations. Here we consider a richer family of sparse factors and leverage recent non-convex optimization techniques.

*6) Machine learning:* Similar models were explored with various points of view in machine learning. For example, sparse multi-factor NMF [30] can be seen as solving problem (2) with the Kullback-Leibler divergence as data fidelity term and all factors $\mathbf{S}_j$s constrained to be non-negative. Optimization relies on multiplicative updates, while the approach proposed here relies on proximal iterations.

*7) Deep learning:* In the context of deep neural networks, identifiability guarantees on the network structure have been established with a generative model where consecutive network layers are sparsely connected at random, and non-linearities are neglected [31], [32]. The network structure in these studies matches the factorized structure (1), with each of the leftmost factors representing a layer of the network and the last one being its input. Apart from its hierarchical flavor, the identification algorithm in [31], [32] has little in comon with the proximal method proposed here.

*8) Signal processing on graphs:* Similar matrix factorizations problems arise in this domain, with the objective of defining wavelets on graphs. First, in [33] the authors propose to approximately diagonalize part of the graph Laplacian operator using elementary rotations. More precisely, the basis in which the Laplacian is expressed is greedily changed, requiring that at each step the change is made by a sparse elementary rotation (so that the wavelet transform is multi-layer sparse), and variables are decorrelated. The Laplacian ends up being diagonal in all the dimensions corresponding to the wavelets and dense in a small part corresponding to the scaling function (the algorithm ends up being very similar to the one proposed in [29]). Second, in [34] the authors propose to define data adaptive wavelets by factorizing some

training data matrix made of signals on the graph of interest. The constraint they impose to the wavelet operator results in a multi-layer sparse structure, where each sparse factor is further constrained to be a lifted wavelet building block. The optimization algorithm they propose relies on deep learning techniques, more precisely layer-wise training of stacked auto-encoders [35].

## III. Optimization framework

### A. Objective function

In this paper, the penalties $g_j(\cdot)$ appearing in the general form of the optimization problem (2) are chosen as indicator functions $\delta_{\mathcal{E}_j}(\cdot)$ of constraint sets of interest $\mathcal{E}_j$. To avoid the scaling ambiguities arising naturally when the constraint sets are (positively) homogeneous[2], it is common [15], [30] to normalize the factors and introduce a multiplicative scalar $\lambda$ in the data fidelity term. For that, let us introduce the sets of normalized factors $\mathcal{N}_j = \{\mathbf{S} \in \mathbb{R}^{a_{j+1} \times a_j} : \|\mathbf{S}\|_F = 1\}$, and impose the following form for the constraints sets: $\mathcal{E}_j = \mathcal{N}_j \cap \mathcal{S}_j$, where $\mathcal{S}_j$ imposes sparsity explicitly or implicitly. This results in the following optimization problem:

$$\underset{\lambda, \mathbf{S}_1, \ldots, \mathbf{S}_J}{\text{Minimize}} \quad \Psi(\mathbf{S}_1, \ldots, \mathbf{S}_J, \lambda) := \frac{1}{2} \left\| \mathbf{A} - \lambda \prod_{j=1}^{J} \mathbf{S}_j \right\|_F^2 + \sum_{j=1}^{J} \delta_{\mathcal{E}_j}(\mathbf{S}_j). \tag{4}$$

As will be made clear below, the used minimization algorithm relies on projections onto the constraint sets $\mathcal{E}_j$: the choice of the "sparsity-inducing" part of the constraint sets $\mathcal{S}_j$ is quite free provided that the projection operator onto these sets is known.

A comon choice is to limit the total number of non-zero entries in the factors to $s_j$. The constraint sets then take the form $\mathcal{E}_j = \{\mathbf{S} \in \mathbb{R}^{a_{j+1} \times a_j} : \|\mathbf{S}\|_0 \leq s_j, \|\mathbf{S}\|_F = 1\}$. Another natural choice is to limit to $k_j$ the number of non-zero entries per row or column in the factors, which gives for example in the case of the columns $\mathcal{E}_j = \{\mathbf{S} \in \mathbb{R}^{a_{j+1} \times a_j} : \|\mathbf{s}_i\|_0 \leq k_j \, \forall i, \|\mathbf{S}\|_F = 1\}$. Other possible constraint sets can be chosen to further impose non-negativity, a circulant structure, a prescribed support, etc., see for example [15].

Besides the few examples given above, many more choices of penalties beyond indicator functions of constraint sets can be envisioned in the algorithmic framework described below. Their choice is merely driven by the application of interest, as long as they are endowed with easy to compute projections onto the constraint sets (in fact, efficient proximal operators), and satisfy some technical assumptions (detailed below) that are very often met in practice. We leave the full exploration of this rich field and its possible applications to further work.

### B. Algorithm overview

Problem (4) is highly non-convex, and the sparsity-inducing penalties are typically non-smooth. Stemming on recent advances in non-convex optimization, it is nevertheless possible to propose an algorithm with convergence guarantees to a stationary point of the problem. In [18], the authors consider cost functions depending on $N$ blocks of variables of the form:

$$\Phi(\mathbf{x}_1, \ldots, \mathbf{x}_N) := H(\mathbf{x}_1, \ldots, \mathbf{x}_N) + \sum_{j=1}^{N} f_j(\mathbf{x}_j), \tag{5}$$

where the function $H$ is smooth, and the penalties $f_j$s are proper and lower semi-continuous (the exact assumptions are given below). It is to be stressed that *no convexity* of any kind is assumed. Here, we assume for simplicity that the penalties $f_j$s are indicator functions of constraint sets $\mathcal{T}_j$. To handle this objective function, the authors propose an algorithm called Proximal Alternating Linearized Minimization (PALM) [18], that updates alternatively each block of variable by a proximal (or projected in our case) gradient step. The structure of the PALM algorithm is given in Figure 3, where $P_{\mathcal{T}_j}(\cdot)$ is the projection operator onto the set $\mathcal{T}_j$ and $c_j^i$ defines the step size and depends on the Lipschitz constant of the gradient of $H$.

PALM (summary)

**for** $i \in \{1 \cdots Niter\}$ **do**
  **for** $j \in \{1 \cdots N\}$ **do**
    Set $\mathbf{x}_j^{i+1} = P_{\mathcal{T}_j}\left(\mathbf{x}_j^i - \frac{1}{c_j^i} \nabla_{\mathbf{x}_j} H\left(\mathbf{x}_1^{i+1} \ldots \mathbf{x}_j^i \ldots \mathbf{x}_N^i\right)\right)$
  **end for**
**end for**

Fig. 3. PALM algorithm (summary).

The following conditions are sufficient (not necessary) to ensure that each bounded sequence generated by PALM converges to a stationary point of its objective [18, Theorem 3.1] (the sequence converges, which implies convergence of the value of the cost function):

(i) The $f_j$s are proper and lower semi-continuous.
(ii) $H$ is smooth.
(iii) $\Phi$ is semi-algebraic [18, Definition 5.1].
(iv) $\nabla_{\mathbf{x}_j} H$ is globally Lipschitz for all $j$, with Lipschitz moduli $L_j(\mathbf{x}_1 \ldots \mathbf{x}_{j-1}, \mathbf{x}_{j+1} \ldots \mathbf{x}_N)$.
(v) $\forall i, j$, $c_j^i > L_j(\mathbf{x}_1^{i+1} \ldots \mathbf{x}_{j-1}^{i+1}, \mathbf{x}_{j+1}^i \ldots \mathbf{x}_N^i)$ (the inequality need not be strict for convex $f_j$).

### C. Algorithm details

PALM can be instantiated for the purpose of handling the objective of (4). It is quite straightforward to see that there is a match between (4) and (5) by taking $N = J + 1$, $\mathbf{x}_j = \mathbf{S}_j$ for $j \in \{1 \ldots J\}$, $\mathbf{x}_{M+1} = \lambda$, $H$ as the data fidelity term, $f_j(\cdot) = \delta_{\mathcal{E}_j}(.)$ for $j \in \{1 \ldots J\}$ and $f_{J+1}(\cdot) = \delta_{\mathcal{E}_{J+1}}(\cdot) = \delta_{\mathbb{R}}(\cdot) = 0$ (there is no constraint on $\lambda$). This match allows to apply PALM to compute multi-layer sparse approximations, with guaranteed convergence to a stationary point.

*1) Projection operator:* PALM relies on projections onto the constraint sets for each factor at each iteration, so the projection operator should be simple and easy to compute. For example, in the case where the $\mathcal{E}_j$s are sets of sparse normalized matrices, namely $\mathcal{E}_j = \{\mathbf{S} \in \mathbb{R}^{a_j \times a_{j+1}} : \|\text{vec}(\mathbf{S})\|_0 \leq s_j, \|\mathbf{S}\|_F = 1\}$ for $j \in \{1 \ldots J\}$, then the projection operator $P_{\mathcal{E}_j}(\cdot)$ simply keeps the $s_j$ greatest entries (in absolute value)

[2]This is the case of many standard constraint sets. In particular all unions of subspaces, such as the sets of sparse or low-rank matrices, are homogeneous.

of its argument, sets all the other entries to zero, and then normalizes its argument so that it has unit norm (the proof is given in Appendix A). Regarding $\mathcal{E}_{J+1} = \mathbb{R}$, the projection operator is the identity mapping. The projection operators for other forms of sparsity constraints that could be interesting in concrete applications are also given in Appendix A: Proposition A.1 covers the following examples:

- Global sparsity constraints.
- Row or column sparsity constraints.
- constrained support.
- Triangular matrices constraints.
- Diagonal matrices constraints.

Proposition A.2 covers in addition:

- Circulant, Toeplitz or Hankel matrices with fixed support or prescribed sparsity.
- Matrices that are constant by row or column.
- More general classes of piece-wise constant matrices with possible sparsity constraints.

*2) Gradient and Lipschitz modulus:* To specify the iterations of PALM specialized to the multi-layer sparse approximation problem, let us fix the iteration $i$ and the factor $j$, and denote $\mathbf{S}_j^i$ the factor being updated, $\mathbf{L} := \prod_{\ell=j+1}^{J} \mathbf{S}_\ell^i$ what is on its left and $\mathbf{R} := \prod_{\ell=1}^{j-1} \mathbf{S}_\ell^{i+1}$ what is on its right (with the convention $\prod_{\ell \in \varnothing} \mathbf{S}_\ell = \mathbf{Id}$). These notations give, when updating the $j$th factor $\mathbf{S}_j^i$: $H(\mathbf{S}_1^{i+1}, \ldots, \mathbf{S}_{j-1}^{i+1}, \mathbf{S}_j^i, \ldots, \mathbf{S}_J^i, \lambda^i) = H(\mathbf{L}, \mathbf{S}_j^i, \mathbf{R}, \lambda^i) = \frac{1}{2}\|\mathbf{A} - \lambda^i \mathbf{L}\mathbf{S}_j^i\mathbf{R}\|_F^2$. The gradient of this smooth part of the objective with respect to the $j$th factor reads:

$$\nabla_{\mathbf{S}_j^i} H(\mathbf{L}, \mathbf{S}_j^i, \mathbf{R}, \lambda^i) = \lambda^i \mathbf{L}^T(\lambda^i \mathbf{L}\mathbf{S}_j^i\mathbf{R} - \mathbf{A})\mathbf{R}^T,$$

which has a Lipschitz modulus with respect to $\left\|\mathbf{S}_j^i\right\|_F$: $L_j(\mathbf{L}, \mathbf{R}, \lambda^i) = (\lambda^i)^2 \left\|\mathbf{R}\right\|_2^2 . \left\|\mathbf{L}\right\|_2^2$ (as shown in Appendix B). Once all the $J$ factors are updated, let us now turn to the update of $\lambda$. Denoting $\hat{\mathbf{A}} = \prod_{j=1}^{J} \mathbf{S}_j^{i+1}$ brings: $H(\mathbf{S}_1^{i+1}, \ldots, \mathbf{S}_J^{i+1}, \lambda^i) = \frac{1}{2}\|\mathbf{A} - \lambda^i \hat{\mathbf{A}}\|_F^2$, and the gradient with respect to $\lambda^i$ reads:

$$\nabla_{\lambda^i} H(\mathbf{S}_1^{i+1}, \ldots, \mathbf{S}_J^{i+1}, \lambda^i) = \lambda^i \mathrm{Tr}(\hat{\mathbf{A}}^T\hat{\mathbf{A}}) - \mathrm{Tr}(\mathbf{A}^T\hat{\mathbf{A}}).$$

*3) Default initialization, and choice of the step size:* Except when specified otherwise, the default initialization is with $\lambda^0 = 1$, $\mathbf{S}_1^0 = \mathbf{0}$, and $\mathbf{S}_j^0 = \mathbf{Id}$ for $j \geq 2$, with the convention that for rectangular matrices the identity has ones on the main diagonal and zeroes elsewhere. In practice the step size is chosen by taking $c_j^i = (1+\alpha).(\lambda^i)^2 \left\|\mathbf{R}\right\|_2^2 . \left\|\mathbf{L}\right\|_2^2$ with $\alpha = 10^{-3}$. Such a determination of the step size is computationally costly, and alternatives could be considered in applications (a decreasing step size rule for example).

*4) Summary:* An explicit version of the algorithm, called PALM for Multi-layer Sparse Approximation (`palm4MSA`), is given in Figure 4, in which the factors are updated alternatively by a projected gradient step (line 6) with a step-size controlled by the Lipschitz modulus of the gradient (line 5). We can solve for $\lambda$ directly at each iteration (line 9) because of the absence of constraint on it (thanks to the second part of the convergence condition (v) of PALM).

PALM for Multi-layer Sparse Approximation (`palm4MSA`)

**Input:** Operator $\mathbf{A}$; desired number of factors $J$; constraint sets $\mathcal{E}_j$, $j \in \{1 \ldots J\}$; initialization $\{\mathbf{S}_j^0\}_{j=1}^J$, $\lambda^0$; stopping criterion (e.g., number of iterations $N$).

1: **for** $i = 0$ to $N-1$ **do**
2: **for** $j = 1$ to $J$ **do**
3: $\mathbf{L} \leftarrow \prod_{\ell=j+1}^{J} \mathbf{S}_\ell^i$
4: $\mathbf{R} \leftarrow \prod_{\ell=1}^{j-1} \mathbf{S}_\ell^{i+1}$
5: Set $c_j^i > (\lambda^i)^2 \left\|\mathbf{R}\right\|_2^2 . \left\|\mathbf{L}\right\|_2^2$
6: $\mathbf{S}_j^{i+1} \leftarrow P_{\mathcal{E}_j}\left(\mathbf{S}_j^i - \frac{1}{c_j^i}\lambda^i \mathbf{L}^T(\lambda \mathbf{L}\mathbf{S}_j^i\mathbf{R} - \mathbf{A})\mathbf{R}^T\right)$
7: **end for**
8: $\hat{\mathbf{A}} \leftarrow \prod_{j=1}^{J} \mathbf{S}_j^{i+1}$
9: $\lambda^{i+1} \leftarrow \frac{\mathrm{Tr}(\mathbf{A}^T\hat{\mathbf{A}})}{\mathrm{Tr}(\hat{\mathbf{A}}^T\hat{\mathbf{A}})}$
10: **end for**

**Output:** The estimated factorization:
$\lambda^N$,$\{\mathbf{S}_j^N\}_{j=1}^J$ = `palm4MSA`($\mathbf{A}$, $J$, $\{\mathcal{E}_j\}_{j=1}^J$, ...)

Fig. 4. PALM algorithm for multi-layer sparse approximation.

## IV. Hierarchical factorization

The algorithm presented in Figure 4 factorizes an input matrix corresponding to an operator of interest into $J$ sparse factors and converges to a stationary point of the problem stated in (4). In practice, one is only interested in the stationary points where the data fitting term of the cost function is small, however as for any generic non-convex optimization algorithm there is no general convergence guarantee to such a stationary point. This fact is illustrated by a very simple experiment where the algorithm `palm4MSA` is applied to an input operator $\mathbf{A} \in \mathbb{R}^{n \times n}$ with a known sparse form $\mathbf{A} = \prod_{j=1}^{N} \mathbf{S}_j$, such as the Hadamard transform (in that case $N = \log_2 n$). The naive approach consists in setting directly $J = N$ in `palm4MSA`, and setting the constraints so as to reflect the actual sparsity of the true factors (as depicted in Figure 1). This simple strategy performs quite poorly in practice for most initializations, and the attained local minimum is very often not satisfactory (the data fidelity part of the objective function is large).

### A. Parallel with deep learning

Similar issues were faced in the neural network community, where it was found difficult to optimize the weights of neural networks comprising many hidden layers (called deep neural networks, see [36] for a survey on the topic). Until recently, deep networks were often neglected in favor of shallower architectures. However in the last decade, it was proposed [37] to optimize the network not as one big block, but one layer at a time, and then globally optimizing the whole network using gradient descent. This heuristic was shown experimentally to work well on various tasks [35]. More precisely, what was proposed is to perform first a *pre-training* of the layers (each being fed the features produced by the one just below, and the lowermost being fed the data), in order to initialize the weights in a good region to perform then a global *fine tuning* of all layers by simple gradient descent.

### B. Proposed hierarchical algorithm

We noticed experimentally that taking fewer factors ($J$ small) and allowing more non-zero entries per factor led to better approximations. This suggested to adopt a hierarchical strategy reminiscent of pre-training of deep networks, in order to iteratively compute only factorization with 2 factors. Indeed, when $\mathbf{A} = \prod_{j=1}^{N} \mathbf{S}_j$ is the product of $N$ sparse factors, it is also the product $\mathbf{A} = \mathbf{T}_1\mathbf{S}_1$ of 2 factors with $\mathbf{T}_1 = \prod_{j=2}^{N} \mathbf{S}_j$, so that $\mathbf{S}_1$ is sparser than $\mathbf{T}_1$.

*1) Optimization strategy:* The proposed hierarchical strategy consists in iteratively factorizing the input matrix $\mathbf{A}$ into 2 factors, one being sparse (corresponding to $\mathbf{S}_1$), and the other less sparse (corresponding to $\mathbf{T}_1$). The process is repeated on the less sparse factor $\mathbf{T}_1$ until the desired number $J$ of factors is attained. At each step, a global optimization of all the factors introduced so far can be performed in order to fit the product to the original operator $\mathbf{A}$.

*2) Choice of sparsity constraint:* A natural question is that of how to tune the sparsity of the factors and residuals along the process. Denoting $\mathbf{T}_\ell = \prod_{j=\ell+1}^{J} \mathbf{S}_j$, a simple calculation shows that if we expect each $\mathbf{S}_j$ to have roughly $\mathcal{O}(k)$ non-zero entries per row, then $\mathbf{T}_\ell$ cannot have more than $\mathcal{O}(k^{J-(\ell+1)})$ non-zero entries per row. This suggests to decrease exponentially the number of non-zero entries in $\mathbf{T}_\ell$ with $\ell$ and to keep constant $\mathcal{O}(k)$ the number of non-zero entries per row in $\mathbf{S}_j$. This choice of the sparsity constraints is further studied with experiments in Section V.

*3) Implementation details:* The proposed hierarchical strategy is summarized in the algorithm given in Figure 5, where the constraint sets related to the two factors need to be specified for each step: $\tilde{\mathcal{E}}_\ell$ denotes the constraint set related to the left factor $\mathbf{T}_\ell$, and $\mathcal{E}_\ell$ the one for the right factor $\mathbf{S}_\ell$ at the $\ell$th factorization. Roughly we can say that line 3 of the algorithm is here to yield complexity savings. Line 5 is here to improve data fidelity: this global optimization step with `palm4MSA` is initialized with the current values of $\mathbf{T}_\ell$ and $\{\mathbf{S}_j\}_{j=1}^{\ell}$.

In greedy layerwise training of deep neural networks, the factorizations in two (line 3) would correspond to the pre-training and the global optimization (line 5) to the fine tuning.

Hierarchical factorization

**Input:** Operator $\mathbf{A}$; desired number of factors $J$; constraint sets $\tilde{\mathcal{E}}_\ell$ and $\mathcal{E}_\ell$, $\ell \in \{1 \dots J-1\}$.

1: $\mathbf{T}_0 \leftarrow \mathbf{A}$
2: **for** $\ell = 1$ to $J-1$ **do**
3: Factorize the residual $\mathbf{T}_{\ell-1}$ into 2 factors: $\lambda', \{\mathbf{F}_2, \mathbf{F}_1\}$ = `palm4MSA`($\mathbf{T}_{\ell-1}$, 2, $\{\tilde{\mathcal{E}}_\ell, \mathcal{E}_\ell\}$, `init=default`)
4: $\mathbf{T}_\ell \leftarrow \lambda'\mathbf{F}_2$ and $\mathbf{S}_\ell \leftarrow \mathbf{F}_1$
5: Global optimization: $\lambda, \{\mathbf{T}_\ell, \{\mathbf{S}_j\}_{j=1}^{\ell}\}$ = `palm4MSA`($\mathbf{A}$, $\ell+1$, $\{\tilde{\mathcal{E}}_\ell, \{\mathcal{E}_j\}_{j=1}^{\ell}\}$, `init=current`)
6: **end for**
7: $\mathbf{S}_J \leftarrow \mathbf{T}_{J-1}$

**Output:** The estimated factorization: $\lambda, \{\mathbf{S}_j\}_{j=1}^{J}$.

Fig. 5. Hierarchical factorization algorithm.

**Remark** The hierarchical strategy can also be applied the other way around (starting *from the left*), just by transposing the input. We only present here the version that starts *from the right* because the induced notations are simpler. It is also worth being noted that stopping criteria other than the total number of factors can be set. For example, we could imagine to keep factorizing the residual until the approximation error at the global optimization step starts rising or exceeds some pre-defined threshold.

### C. Illustration: reverse-engineering the Hadamard transform

As a first illustration of the proposed approach, we tested the hierarchical factorization algorithm of Figure 5 when $\mathbf{A}$ is the dense square matrix associated to the Hadamard transform in dimension $n = 2^N$. The algorithm was run with $J = N$ factors, $\tilde{\mathcal{E}}_\ell = \{\mathbf{T} \in \mathbb{R}^{n\times n}, \|\mathbf{T}\|_0 \leq \frac{n^2}{2^\ell}, \|\mathbf{T}\|_F = 1\}$, and $\mathcal{E}_\ell = \{\mathbf{S} \in \mathbb{R}^{n\times n}, \|\mathbf{S}\|_0 \leq 2n, \|\mathbf{S}\|_F = 1\}$.

In stark contrast with the direct application of `palm4MSA` with $J = N$, an exact factorization is achieved. Indeed, the first step reached an exact factorization $\mathbf{A} = \mathbf{T}_1\mathbf{S}_1$ independently of the initialization. With the *default* initialization (Section III-C3), the residual $\mathbf{T}_1$ was observed to be still exactly factorizable. All steps ($\ell > 1$) indeed also yielded exact factorizations $\mathbf{T}_{\ell-1} = \mathbf{T}_\ell\mathbf{S}_\ell$, provided the default initialization was used at each step $\ell \in \{1, \dots, J-1\}$.

Figure 6 illustrates the result of the proposed hierarchical strategy in dimension $n = 32$. The obtained factorization is exact and *as good as the reference one* (cf Figure 1) in terms of complexity savings. The running time[3] of the factorization algorithm is less than a second. Factorization of the Hadamard matrix in dimension up to $n = 1024$ showed identical behaviour, with running times $\mathcal{O}(n^2)$ up to ten minutes.

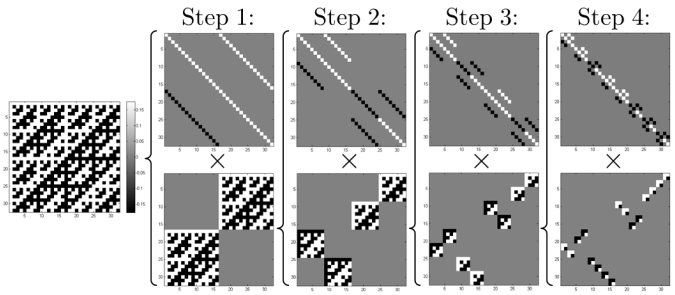


Fig. 6. Hierarchical factorization of the Hadamard matrix of size $32 \times 32$. The matrix is iteratively factorized in 2 factors, until we have $J = 5$ factors, each having $s = 64$ non-zero entries.

## V. Accelerating inverse problems

A natural application of FA$\mu$STs is linear inverse problems, where a high-dimensional vector $\boldsymbol{\gamma}$ needs to be retrieved from some observed data $\mathbf{y} \approx \mathbf{M}\boldsymbol{\gamma}$. As already evoked in Section II-B, iterative proximal algorithms can be expected to be

[3]All experiments were performed in Matlab on an laptop with an intel(R) core(TM) i7-3667U @ 2.00GHz (two cores).

significantly sped up if $\mathbf{M}$ is well approximated with a FA$\mu$ST of low relative complexity, for example using the proposed hierarchical factorization algorithm applied to $\mathbf{A} = \mathbf{M}$.

In practice, one needs to specify the total number of factors $J$ and the constraint sets $\tilde{\mathcal{E}}_\ell$, $\mathcal{E}_\ell$. A preliminary study on synthetic data was carried out in our technical report [1], showing that a flexible trade-off between relative complexity and adaptation to the input matrix can be achieved. Here we leverage the rule of thumb presented in Section III-C to deepen the investigation of this question for a matrix $\mathbf{M}$ arising in a real-world biomedical linear inverse problem.

### A. Factorization compromise: MEG operator

In this experiment, we explore the use of FA$\mu$ST in the context of functional brain imaging using magnetoencephalography (MEG) and electroencephalography (EEG) signals. Source imaging with MEG and EEG delivers insights into the active brain at a millisecond time scale in a non-invasive way. To achieve this, one needs to solve the bioelectromagnetic inverse problem. It is a high dimensional ill-posed regression problem requiring proper regularization. As it is natural to assume that a limited set of brain foci are active during a cognitive task, sparse focal source configurations are commonly promoted using convex sparse priors [38], [39]. The bottleneck in the optimization algorithms are the dot products with the forward matrix and its transpose.

The objective of this experiment is to observe achievable trade-offs between relative complexity and accuracy. To this end, an MEG gain matrix $\mathbf{M} \in \mathbb{R}^{204\times 8193}$ ($m = 204$ and $n = 8193$), computed using the MNE software [40] implementing a BEM, was factorized into $J$ sparse factors using the hierarchical factorization algorithm of Figure 5.

*1) Settings:* The rightmost factor $\mathbf{S}_1$ was of the size of $\mathbf{M}$, but with $k$-sparse columns, corresponding to the constraint set $\mathcal{E}_1 = \{\mathbf{S} \in \mathbb{R}^{204\times 8193}, \|\mathbf{s}_i\|_0 \leq k, \|\mathbf{S}\|_F = 1\}$. All other factors $\mathbf{S}_j$, $j \in \{2, \ldots, J\}$ were set square, with global sparsity $s$, i.e. $\mathcal{E}_\ell = \{\mathbf{S} \in \mathbb{R}^{204\times 204}, \|\mathbf{S}\|_0 \leq s, \|\mathbf{S}\|_F = 1\}$.

The "residual" at each step $\mathbf{T}_\ell$, $\ell \in \{1, \ldots, J-1\}$ was also set square, with global sparsity geometrically decreasing with $\ell$, controlled by two parameters $\rho$ and $P$. This corresponds to the constraint sets[4] $\tilde{\mathcal{E}}_\ell = \{\mathbf{T} \in \mathbb{R}^{204\times 204}, \|\mathbf{T}\|_0 \leq P\rho^{\ell-1}, \|\mathbf{T}\|_F = 1\}$ for $\ell \in \{1, \ldots, J-1\}$. The controlling parameters are set to:

- Number of factors: $J \in \{2, \ldots, 10\}$.
- Sparsity of the rightmost factor: $k \in \{5, 10, 15, 20, 25, 30\}$.
- Average row sparsity of other factors: $\frac{s}{m} \in \{2, 4, 8\}$.
- Rate of decrease of the residual sparsity: $\rho = 0.8$.

The parameter $P$ controlling the global sparsity in the residual was found to have only limited influence, and was set to $P = 1.4 \times m^2$. Other values for $\rho$ were tested, leading to slightly different but qualitatively similar complexity/accuracy trade-offs not shown here. The factorization setting is summarized in Figure 7, where the sparsity of each factor is explicitly given.

[4]Compared to a preliminary version of this experiment [17] where the residual was normalized columnwise at the first step, here it is normalized globally. This leads to slightly better results.

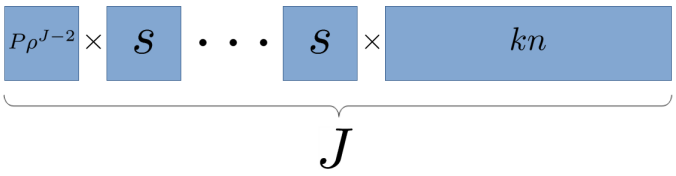


Fig. 7. Factorization setting used. Each factor is represented with its total sparsity.

*2) Results:* Factorizations were computed for 127 parameter settings. The computation time for each factorization was around $(J-1) \times 10$ minutes. Figure 8 displays the trade-off between speed (the RCG measure (3)) and approximation error:

$$\text{RE} := \frac{\left\|\mathbf{M} - \lambda \prod_{j=1}^{J} \mathbf{S}_j\right\|_2}{\|\mathbf{M}\|_2}, \tag{6}$$

of each obtained FA$\mu$ST. We observe that:

- The overall relative complexity of the obtained factorization is essentially controlled by the parameter $k$. This seems natural, since $k$ controls the sparsity of the rightmost factor which is way larger than the other ones.
- The trade-off between complexity and approximation for a given $k$ is mainly driven by the number of factors $J$: higher values of $J$ lead to lower relative complexities, but a too large $J$ leads to a higher relative error. Taking $J = 2$ (black dots) never yields the best compromise, hence the relevance of truly *multi-layer* sparse approximations.
- For a fixed $k$, one can distinguish nearby trade-off curves corresponding to different sparsity levels $s$ of the intermediate factors. The parameter $s$ actually controls the horizontal spacing between two consecutive points on the same curve: a higher $s$ allows to take a higher $J$ without increasing the error, but in turn leads to a higher relative complexity for a given number $J$ of factors.

In summary, one can distinguish as expected a trade-off between relative complexity and approximation error. The configuration exhibiting the lowest relative error for each value of $k$ is highlighted on Figure 8, this gives $\widehat{\mathbf{M}}_{25}$, $\widehat{\mathbf{M}}_{16}$, $\widehat{\mathbf{M}}_{11}$, $\widehat{\mathbf{M}}_{8}$, $\widehat{\mathbf{M}}_{7}$, $\widehat{\mathbf{M}}_{6}$, where the subscript indicates the achieved RCG (rounded to the closest integer). For example, $\widehat{\mathbf{M}}_6$ can multiply vectors with 6 times less flops than $\mathbf{M}$ (saving $84\%$ of computation), and $\widehat{\mathbf{M}}_{25}$ can multiply vectors with 25 times less flops than $\mathbf{M}$ ($96\%$ savings). These six matrices are those appearing on Figure 2 to compare FA$\mu$STs to the truncated SVD. They will next be used to solve an inverse problem and compared to results obtained with $\mathbf{M}$.

**Remark** Slightly smaller approximation errors can be obtained by imposing a global sparsity constraint to the rightmost factor, i.e., $\mathcal{E}_1 = \{\mathbf{S} \in \mathbb{R}^{204\times 8193}, \|\mathbf{S}\|_0 \leq kn, \|\mathbf{S}\|_F = 1\}$. This is shown on Figure 8 by the points linked by a dashed line to the six matrices outlined above. However, such a global sparsity constraint also allows the appearance of null columns in the approximations of $\mathbf{M}$, which is undesirable for the application considered next.

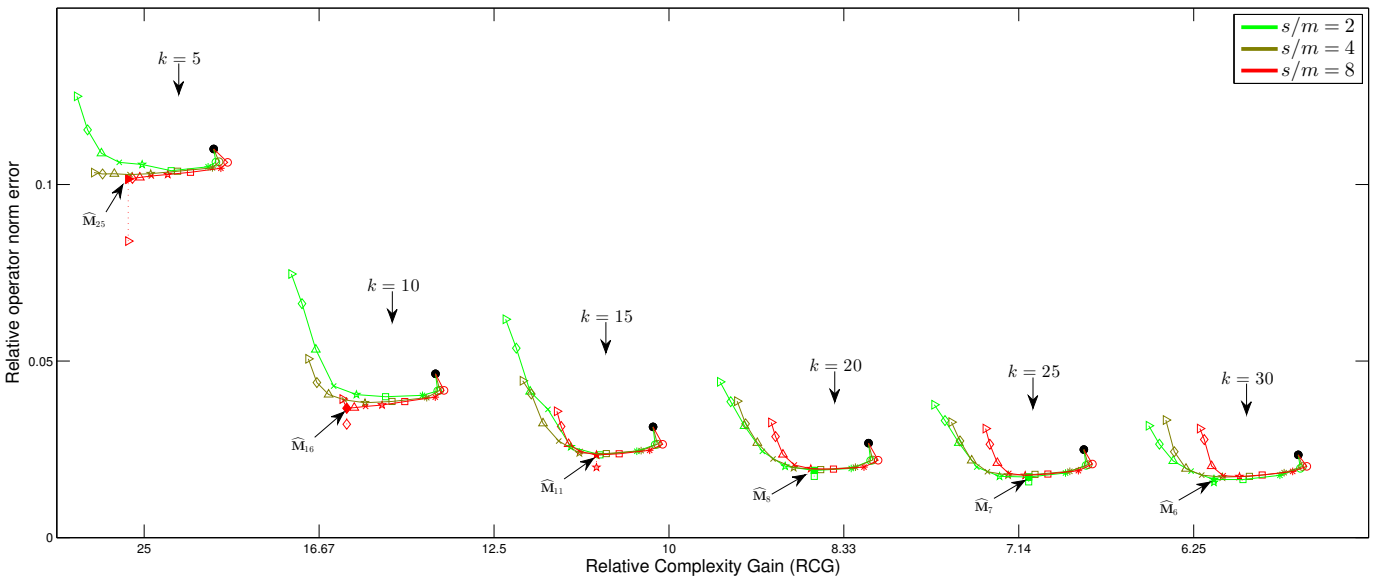


Fig. 8. Results of the factorization of an $m \times n = 204 \times 8193$ MEG matrix. The shape of the symbols denotes the number of factors $J$ ($\bullet : J = 2$, $\bigcirc : J = 3$, $* : J = 4$, $\square : J = 5$, $\star : J = 6$, $\times : J = 7$, $\triangle : J = 8$, $\diamond : J = 9$, $\triangleright : J = 10$), and the color the value of the parameter $s$.

### B. Source localization experiment

We now assess the impact of replacing the MEG gain matrix $\mathbf{M} \in \mathbb{R}^{204\times 8193}$ by a FA$\mu$ST approximation for brain source localization. For this synthetic experiment, two brain sources chosen located uniformly at random were activated with gaussian random weights, giving a 2-sparse vector $\boldsymbol{\gamma} \in \mathbb{R}^{8193}$, whose support encodes the localization of the sources. Observing $\mathbf{y} := \mathbf{M}\boldsymbol{\gamma}$, the objective is to estimate (the support of) $\boldsymbol{\gamma}$. The experiment then amounts to solving the inverse problem to get $\hat{\boldsymbol{\gamma}}$ from the measurements $\mathbf{y} = \mathbf{M}\boldsymbol{\gamma} \in \mathbb{R}^{204}$, using either $\mathbf{M}$ or a FA$\mu$ST $\widehat{\mathbf{M}}$ during the recovery process.

Three recovery methods were tested: Orthogonal Matching Pursuit (OMP) [10] (choosing 2 atoms), $\ell_1$-regularized least squares ($\mathrm{l_1ls}$) [41] and Iterative Hard Thresholding (IHT) [11]). They yielded qualitatively similar results, and for the sake of conciseness we present here only the results for OMP.

The matrices used for the recovery are the actual matrix $\mathbf{M}$ and its FA$\mu$ST approximations $\widehat{\mathbf{M}}_{25}$, $\widehat{\mathbf{M}}_{16}$, $\widehat{\mathbf{M}}_{11}$, $\widehat{\mathbf{M}}_{8}$, $\widehat{\mathbf{M}}_{7}$ and $\widehat{\mathbf{M}}_{6}$. The expected computational gain of using a FA$\mu$ST instead of $\mathbf{M}$ is of the order of RCG, since the computational cost of OMP is dominated by products with $\mathbf{M}^T$.

Three configurations were considered when generating the location of the sources, in term of distance $d$ (in centimeters) between the sources. For each configuration, 500 vectors $\mathbf{y} = \mathbf{M}\boldsymbol{\gamma}$ were generated, and OMP was run using each matrix. The distance between each actual source and the closest retrieved source was measured. Figure 9 displays the statistics of this distance for all scenarios:

- As expected, localization is better when the sources are more separated, independently of the choice of matrix.
- Most importantly, the performance is almost as good when using FA$\mu$STs $\widehat{\mathbf{M}}_6$, $\widehat{\mathbf{M}}_7$, $\widehat{\mathbf{M}}_8$ and $\widehat{\mathbf{M}}_{11}$ than when using the actual matrix $\mathbf{M}$, although the FA$\mu$STs are way more computationally efficient (6 to 11 times less computations). For example, in the case of well separated sources ($d > 8$), the FA$\mu$STs allow to retrieve exactly the sought sources more than 75% of the time, which is almost as good as when using the actual matrix $\mathbf{M}$.
- The performance with the two other FA$\mu$STs $\widehat{\mathbf{M}}_{16}$ and $\widehat{\mathbf{M}}_{25}$ is a bit poorer, but they are even more computationally efficient matrix (16 and 25 times less computations). For example, in the case of well separated sources ($d > 8$), they allow to retrieve exactly the sought sources more than 50% of the time.

These observations confirm it is possible to slighlty trade-off localization performance for substantial computational gains, and that FA$\mu$STs can be used to speed up inverse problems without a large precision loss.

## VI. Learning fast dictionaries

Multi-layer sparse approximations of operators are particularly suited for choosing efficient dictionaries for data processing tasks.

### A. Analytic vs. learned dictionaries

Classically, there are two paths to choose a dictionary for sparse signal representations [13].

Historically, the only way to come up with a dictionary was to analyze mathematically the data and derive a "simple" formula to construct the dictionary. Dictionaries designed this way are called *analytic dictionaries* [13] (e.g.: associated to Fourier, wavelets and Hadamard transforms). Due to the relative simplicity of analytic dictionaries, they usually have a known sparse form such as the Fast Fourier Transform (FFT) [3] or the Discrete Wavelet Transform (DWT) [6].

On the other hand, the development of modern computers allowed the surfacing of automatic methods that learn a dictionary directly from the data [42]–[44]. Given some raw data $\mathbf{Y} \in \mathbb{R}^{m\times L}$, the principle of dictionary learning is to

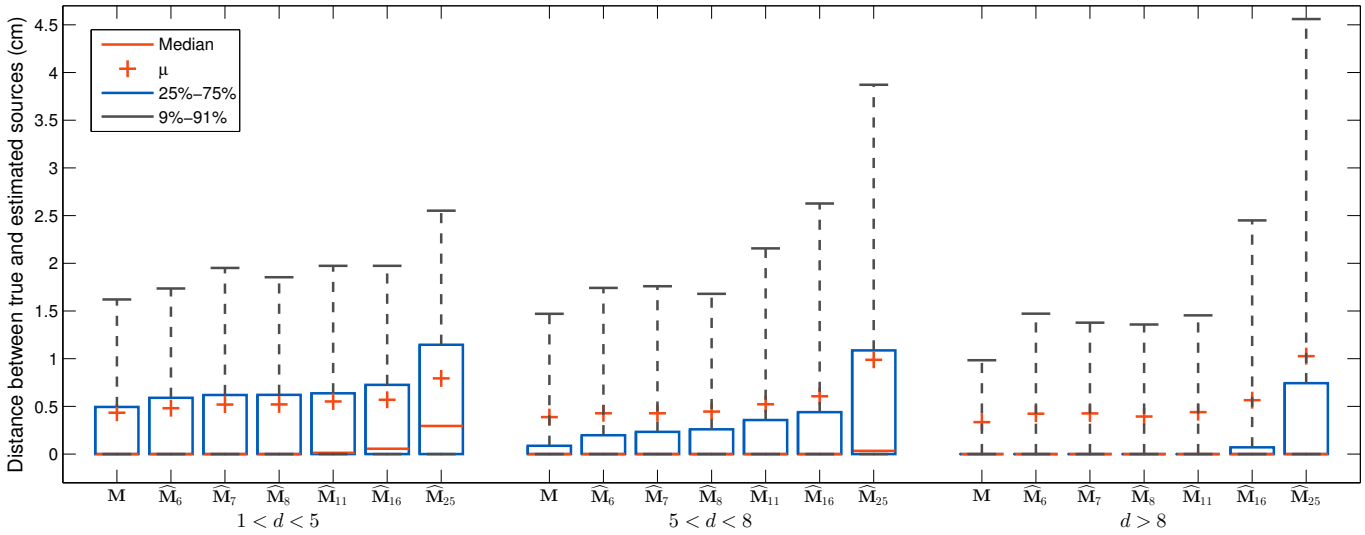


Fig. 9. Localization performance (distance between actual and retrieved source) obtained with various matrices, for different distances between actual sources.

approximate $\mathbf{Y}$ by the product of a dictionary $\mathbf{D} \in \mathbb{R}^{m \times n}$ and a coefficient matrix $\mathbf{\Gamma} \in \mathbb{R}^{n \times L}$ with sparse columns:

$$\mathbf{Y} \approx \mathbf{D}\mathbf{\Gamma}.$$

Such *learned dictionaries* are usually well adapted to the data at hand. However, being in general dense matrices with no apparent structure, they do not lead to fast algorithms and are costly to store. We typically have $L \gg \max(m, n)$ (for sample complexity reasons), which implies to be very careful about the computational efficiency of learning in that case.

### B. The best of both worlds

Can one design dictionaries as well adapted to the data as learned dictionaries, while as fast to manipulate and as cheap to store as analytic ones? This question has begun to be explored recently [14], [15], and actually amounts to learning of dictionary that are FA$\mu$STs. More precisely, given $\mathbf{Y}$, the objective is to learn a dictionary being a FA$\mu$ST (as in (1)):

$$\mathbf{D} = \prod_{j=1}^{J} \mathbf{S}_j.$$

This can be done by inserting a dictionary factorization step into the traditional structure of dictionary learning algorithms [13], as illustrated on Figure 10. A consequence is that the

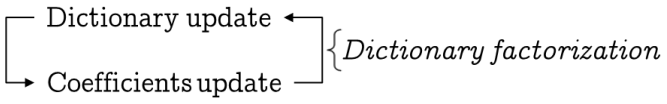


Fig. 10. Classical dictionary learning algorithm structure (in roman). In italic, the added dictionary factorization step, specific to the approach presented here.

coefficients update can be sped up by exploiting the FA$\mu$ST structure of the dictionary. The approach described below uses a batch method for dictionary update, but the approach is a priori also compatible with stochastic gradient descent in the dictionary update for even more efficiency.

In practice we propose to slightly modify the hierarchical factorization algorithm of Figure 5. The idea is to take a dictionary $\mathbf{D}$ learned on some training data $\mathbf{Y}$ (with any classical dictionary learning method, such as K-SVD [43]) and to hierarchically factorize it, taking into account and jointly updating the coefficients matrix $\mathbf{\Gamma}$.

The resulting hierarchical factorization algorithm adapted to dictionary learning is given in Figure 11. The only differences with the hierarchical factorization algorithm given previously is that the coefficients matrix is taken into account (but kept fixed) in the global optimization step, and that an update of the coefficients by sparse coding is added after this global optimization step. This sparse coding step can actually be done by any algorithm (OMP, IHT, ISTA...), denoted by the general `sparseCoding` algorithm in Figure 11.

**Remark** As noted in section IV-B3, the dictionary factorization is presented here starting from the right. It could as well be performed starting from the left.

### C. Image denoising experiment

In order to illustrate the advantages of FA$\mu$ST dictionaries over classical dense ones, an image denoising experiment is performed here. The experimental scenario for this task follows a simplified dictionary based image denoising workflow. First, $L = 10000$ patches $\mathbf{y}_i$ of size $8 \times 8$ (dimension $m = 64$) are randomly picked from an input $512 \times 512$ noisy image (with various noise levels, of variance $\sigma \in \{10, 15, 20, 30, 50\}$), and a dictionary is learned on these patches. Then the learned dictionary is used to denoise the entire input image by computing the sparse representation of all its patches in the dictionary using OMP, allowing each patch to use 5 dictionary atoms. The image is reconstructed by averaging the overlapping patches.

Hierarchical factorization for dictionary learning

**Input:** Data matrix $\mathbf{Y}$; Initial dictionary $\mathbf{D}$ and coefficients $\mathbf{\Gamma}$ (e.g., from K-SVD); desired number of factors $J$; constraint sets $\tilde{\mathcal{E}}_\ell$ and $\mathcal{E}_\ell$, $\ell \in \{1 \dots J-1\}$.

1: $\mathbf{T}_0 \leftarrow \mathbf{D}$
2: **for** $\ell = 1$ to $J-1$ **do**
3: *Dictionary factorization*: factorize the residual $\mathbf{T}_{\ell-1}$ into 2 factors
$\lambda', \{\mathbf{F}_2, \mathbf{F}_1\}$ = `palm4MSA`($\mathbf{T}_{\ell-1}$, 2, $\{\tilde{\mathcal{E}}_\ell, \mathcal{E}_\ell\}$, `init=default`)
$\mathbf{T}_\ell \leftarrow \lambda' \mathbf{F}_2$ and $\mathbf{S}_\ell \leftarrow \mathbf{F}_1$
4: *Dictionary update*: global optimization
$\lambda, \left\{\mathbf{T}_\ell, \{\mathbf{S}_j\}_{j=1}^{\ell}, \mathbf{\Gamma}\right\}$ = `palm4MSA`($\mathbf{Y}$, $\ell+2$, $\left\{\tilde{\mathcal{E}}_\ell, \{\mathcal{E}_j\}_{j=1}^{\ell}, \{\mathbf{\Gamma}\}\right\}$, `init=current`)
5: *Coefficients update*:
$\mathbf{\Gamma}$ = `sparseCoding`($\mathbf{Y}$, $\mathbf{T}_\ell \prod_{j=1}^{\ell} \mathbf{S}_j$)
6: **end for**
7: $\mathbf{S}_J \leftarrow \mathbf{T}_{J-1}$

**Output:** The estimated factorization: $\lambda, \{\mathbf{S}_j\}_{j=1}^{J}$.

Fig. 11. Hierarchical factorization algorithm for dictionary learning.

**Experimental settings.** Several configurations were tested. The number of atoms $n$ was taken in $\{128, 256, 512\}$. Inspired by usual fast transforms, a number of factors $J$ close to the logarithm of the signal dimension $m = 64$ was chosen, here $J = 4$. The sizes of the factors were: $\mathbf{S}_J, \dots, \mathbf{S}_2 \in \mathbb{R}^{m \times m}$, $\mathbf{S}_1 \in \mathbb{R}^{m \times n}$, and $\mathbf{\Gamma} \in \mathbb{R}^{n \times L}$. The algorithm of Figure 11 was used, with the initial dictionary learning being done by K-SVD [43] and `sparseCoding` being OMP, allowing each patch to use 5 dictionary atoms. Regarding the constraint sets, we took them exactly like in section V-A, taking $s/m \in \{2, 3, 6, 12\}$, $\rho \in \{0.4, 0.5, 0.7, 0.9\}$, $P = 64^2$ and $k = s/m$. For each dictionary size $n$, this amounts to a total of sixteen different configurations leading to different relative complexity values. The stopping criterion for `palm4MSA` was a number of iterations $N_i = 50$. Note that the usage of OMP here and at the denoising stage is a bit abusive since the dictionary does not have unit-norm columns (the factors are normalized instead), but it was used anyway, resulting in a sort of weighted OMP, where some atoms have more weight than others.

**Baseline.** The proposed method was compared to *Dense Dictionary Learning* (DDL). K-SVD is used here to perform DDL, but other algorithms have been tested (such as online dictionary learning [44]), leading to similar qualitative results. In order to assess the generalization performance and to be as close as possible to the matrix factorization framework studied theoretically in [20], DDL is performed following the same denoising workflow than our method (dictionary learned on 10000 noisy patches used to denoise the whole image, allowing five atoms per patch). The implementation described in [45] was used, running 50 iterations (empirically sufficient to ensure convergence).

Note that better denoising performance can be obtained by inserting dictionary learning into a more sophisticated denoising workflows, see e.g. [46]. State of the art denoising algorithms indeed often rely on clever averaging procedure called "aggregation". Our purpose here is primarily to illustrate the potential of the proposed FA$\mu$ST structure for denoising. While such workflows are fully compatible with the FA$\mu$ST structure, we leave the implementation and careful benchmarking of the resulting denoising systems to future work.

**Results.** The experiment is done on the standard image database taken from [47] (12 standard grey $512 \times 512$ images). In Figure 12 are shown the results for three images: the one for which FA$\mu$ST dictionaries perform worst ("Mandrill"), the one for which they perform best ("WomanDarkHair") and the typical behaviour ("Pirate"). Several comments are in order:

- First of all, it is clear that with the considered simple denoising workflow, FA$\mu$ST dictionaries perform better than DDL at strong noise levels, namely $\sigma = 30$ and $\sigma = 50$. This can be explained by the fact that when training patches are very noisy, DDL is prone to overfitting (learning the noise), whereas the structure of FA$\mu$STs seems to prevent it. On the other hand, for low noise levels, we pay the lack of adaptivity of FA$\mu$STs compared to DDL. Indeed, especially for very textured images ("Mandrill" typically), the dictionary must be very flexible in order to fit such complex training patches, so DDL performs better.
- Second, it seems that sparser FA$\mu$STs (with a lower number of parameters) perform better than denser ones for high noise levels. This can be explained by the fact that they are less prone to overfitting because of their lower number of parameters, implying fewer degrees of freedom. However, this is not true for low noise levels or with too few parameters, since in that case the loss of adaptivity with respect to the training patches is too important.
- We give also the result of BM3D [48] for the "Pirate" image to give an idea of the state of the art perfomance. It seems that the best FA$\mu$ST dictionary for each noise level is approximately 2dB worst than BM3D. This gap can be explained by the relatively simple denoising workflow used here, and is expected to be reduced using a more sophisticated one (e.g., closer to the one of [46]).

### *D. Sample complexity of FA$\mu$STs*

The good performance of FA$\mu$ST dictionaries compared to dense ones observed above may be surprising, since the more constrained structure of such dictionaries (compared to dense dictionaries) may bar them from providing good approximations of the considered patches. A possible element of explanation stems from the notion of sample complexity: as evoked in section II-B, the statistical significance of learned multi-layer sparse operators is expected to be improved compared to that of dense operators, thanks to a reduced sample complexity.

In the context of dictionary learning, the sample complexity indicates how many training samples $L$ should be taken in order for the empirical risk to be (with high probability) uniformly close to its expectation [49], [50]. In [20], a general bound on the deviation between the empirical risk and its

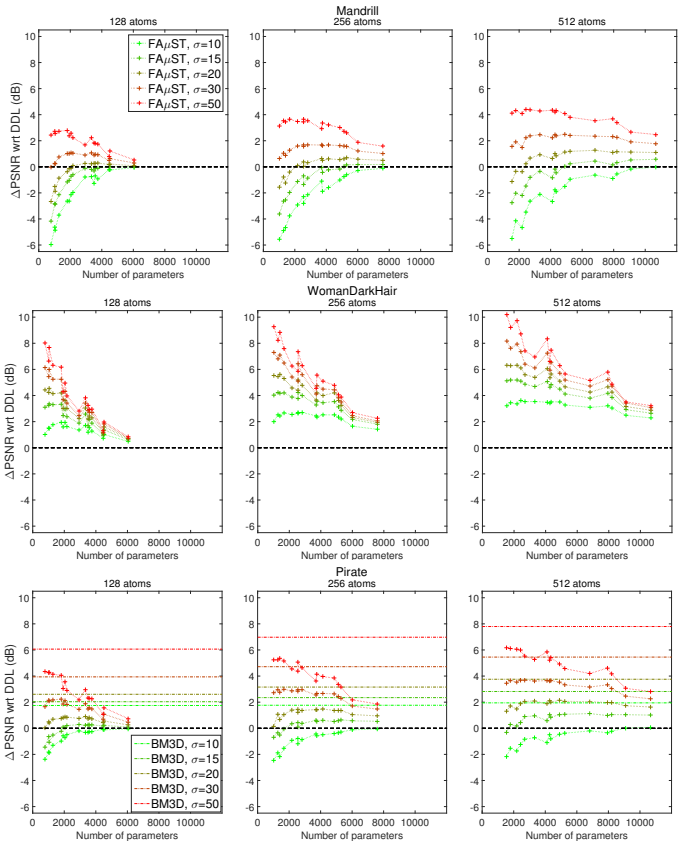


Fig. 12. Denoising results. The relative performance of FA$\mu$ST dictionaries compared to DDL is given for several noise levels $\sigma$, and the horizontal axis measures the total number of parameters of each FA$\mu$ST.

expectation is provided, which is proportional to the covering dimension of the dictionary class.

For dense dictionaries the covering dimension is known to be $\mathcal{O}(mn)$ [20], [49], [50]. For FA$\mu$ST dictionaries we establish in Appendix C the following theorem.

**Theorem VI.1.** *For multi-layer sparse operators, the covering dimension is bounded by* $s_{tot}$.

A consequence is that for the same number of training samples $L$, a better generalization performance is expected from FA$\mu$ST dictionaries compared to dense ones, the gain being of the order of RCG. This fact is likely to explain the empirical success of FA$\mu$STs compared to dense dictionaries observed in section VI-C at low SNR. Of course, when $s_{tot}$ becomes to small, the limited approximation capacity of FA$\mu$ST dictionaries imposes a trade-off between approximation and generalization.

## VII. Conclusion and Future Work

In this paper, a novel multi-layer matrix factorization framework was introduced, which allows to approximate a given linear operator by the composition of several ones. The underlying factorization algorithm stems on recent advances in non-convex optimization and has convergence guarantees. The proposed approach consists in hierarchically factorizing the input matrix in the hope of attaining better local minima, as is done for example in deep learning. The factorization algorithm that is used is pretty general and is able to take into account various constraints. One practical constraint of interest is sparsity (of various forms), which has several interesting properties. Indeed, multi-layer sparsely factorized linear operators have several advantages over classical dense ones, such as an increased speed of manipulation, a lighter storage footpring, and a higher statistical significance when estimated on training data.

The interest and versatility of the proposed factorization approach was demonstrated with various experiments, including a source localization one where the proposed method performs well with a greatly reduced computational cost compared to previous techniques. We performed also image denoising experiments demonstrating that the proposed method has better generalization performance than dense dictionary learning with an impact at low SNR.

In the future, several developments are expected. On the theoretical side, we envision bounds on the trade-off between approximation quality and relative complexity. On the experimental side, new applications for FA$\mu$STs are to be explored. For example, signal processing on graphs is a relatively new discipline where computationally efficient operators can be envisioned using learned FA$\mu$ST dictionaries, or a FA$\mu$ST approximation of graph Fourier transforms.

## Appendix A
## Projection Operators

In this appendix are given the projection operators onto several constraint sets of interest for practical applications.

### A. General sparsity constraint

Sparsity is the most obvious constraint to put on the factors for operator sparse approximations. Consider first the following general sparsity constraint set:

$$\mathcal{E} := \{\mathbf{S} \in \mathbb{R}^{p\times q} : \|\mathbf{S}_{\mathcal{H}_i}\|_0 \leq s_i \forall i \in \{1,\dots,K\}, \|\mathbf{S}\|_F = 1\}, \tag{7}$$

where $\{\mathcal{H}_1,\dots,\mathcal{H}_K\}$ forms a partition of the index set, $s_i \in \mathbb{N}, \forall i \in \{1,\dots,K\}$, and $\mathbf{S}_{\mathcal{T}}$ is the matrix whose entries match those of $\mathbf{S}$ on $\mathcal{T}$ and are set to zero elsewhere. Given some matrix $\mathbf{U} \in \mathbb{R}^{p\times q}$, we wish to compute its projection onto the set $\mathcal{E}$:

$$P_{\mathcal{E}}(\mathbf{U}) \in \arg\min_{\mathbf{S}} \{\|\mathbf{S}-\mathbf{U}\|_F^2 : \mathbf{S} \in \mathcal{E}\} \tag{8}$$

**Proposition A.1.** *Projection operator formula.*

$$P_{\mathcal{E}}(\mathbf{U}) = \frac{\mathbf{U}_{\mathcal{I}}}{\|\mathbf{U}_{\mathcal{I}}\|_F}$$

*where* $\mathcal{I}$ *is the index set corresponding to the union of the* $s_i$ *entries of* $\mathbf{U}_{\mathcal{H}_i}$ *with largest absolute value,* $\forall i \in \{1,\dots,K\}$.

*Proof.* Let $\mathbf{S}$ be an element of $\mathcal{E}$ and $\mathcal{J}$ its support. We have

$$\|\mathbf{S}-\mathbf{U}\|_F^2 \quad = \quad 1 + \|\mathbf{U}\|_F^2 - 2\langle \mathrm{vec}(\mathbf{U}_{\mathcal{J}}), \mathrm{vec}(\mathbf{S})\rangle.$$

For a given support, the matrix $\mathbf{S} \in \mathcal{E}$ maximizing $\langle \text{vec}(\mathbf{U}_\mathcal{J}), \text{vec}(\mathbf{S})\rangle$ is $\mathbf{S} = \mathbf{U}_\mathcal{J} / \|\mathbf{U}_\mathcal{J}\|_F$. For this matrix

$$\langle \text{vec}(\mathbf{U}_\mathcal{J}), \text{vec}(\mathbf{S})\rangle = \|\mathbf{U}_\mathcal{J}\|_F = \sqrt{\sum_{i=1}^{K} \|\mathbf{U}_{\mathcal{J}\cap\mathcal{H}_i}\|_F^2}$$

which is maximized if $\mathcal{J} \cap \mathcal{H}_i$ corresponds to the $s_i$ entries with largest absolute value of $\mathbf{U}$ within $\mathcal{H}_i$, $\forall i \in \{1, \dots, K\}$. □

### B. Sparse and piecewise constant constraints

Given $K$ pairwise disjoint sets $\mathcal{C}_i$ indexing matrix entries, consider now the constraint set corresponding to unit norm matrices that are constant over each index set $\mathcal{C}_i$, zero outside these sets, with no more than $s$ non-zero areas. In other words: $\mathcal{E}_c := \{\mathbf{S} \in \mathbb{R}^{p\times q} : \exists \tilde{\mathbf{a}} = (\tilde{a}_i)_{i=1}^K, \|\tilde{\mathbf{a}}\|_0 \leq s, \mathbf{S}_{\mathcal{C}_i} = \tilde{a}_i \forall i \in \{1, \dots, K\}, \mathbf{S}_{\overline{\bigcup_i \mathcal{C}_i}} = 0, \text{and } \|\mathbf{S}\|_F = 1\}$.

Define $\tilde{\mathbf{u}} := (\tilde{u}_i)_{i=1}^K$ with $\tilde{u}_i := \sum_{(m,n)\in\mathcal{C}_i} u_{mn}$, and denote $\tilde{\mathcal{J}} \subset \{1, \dots, K\}$ the support of $\tilde{\mathbf{a}}$.

**Proposition A.2.** *The projection of* $\mathbf{U}$ *onto* $\mathcal{E}_c$ *is obtained with* $\tilde{\mathcal{J}}$ *the collection of* $s$ *indices* $i$ *yielding the highest* $|\tilde{u}_i|/\sqrt{|\mathcal{C}_i|}$, $\tilde{a}_i := \tilde{u}_i / \sqrt{\sum_{i\in\tilde{\mathcal{J}}} |\mathcal{C}_i| \tilde{u}_i^2})$ *if* $i \in \tilde{\mathcal{J}}$, $\tilde{a}_i := 0$ *otherwise.*

*Proof.* Let $\mathbf{S}$ be an element of $\mathcal{E}_c$, and $\tilde{\mathcal{J}} \subset \{1, \dots, K\}$ be the support of the associated $\tilde{\mathbf{a}}$. We proceed as for the previous proposition and notice that

$$\begin{aligned}\langle \text{vec}(\mathbf{U}), \text{vec}(\mathbf{S})\rangle &= \textstyle\sum_{i\in\tilde{\mathcal{J}}} \langle \text{vec}(\mathbf{U}_{\mathcal{C}_i}), \text{vec}(\mathbf{S})\rangle \\ &= \textstyle\sum_{i\in\tilde{\mathcal{J}}} \tilde{u}_i \tilde{a}_i = \langle \tilde{\mathbf{u}}_{\tilde{\mathcal{J}}}, \tilde{\mathbf{a}}\rangle.\end{aligned}$$

By the changes of variable $\tilde{b}_i = \sqrt{|\mathcal{C}_i|}.\tilde{a}_i$ and $\tilde{v}_i = \tilde{u}_i/\sqrt{|\mathcal{C}_i|}$ we get $\langle \tilde{\mathbf{u}}_{\tilde{\mathcal{J}}}, \tilde{\mathbf{a}}\rangle = \langle \tilde{\mathbf{v}}_{\tilde{\mathcal{J}}}, \tilde{\mathbf{b}}\rangle$ with $\tilde{\mathbf{b}} := (\tilde{b}_i)_{i=1}^K$. Given $\tilde{\mathcal{J}}$, maximizing this scalar product under the constraint $1 = \|\mathbf{S}\|_F = \|\tilde{\mathbf{b}}\|_2$ yields $\tilde{\mathbf{b}}^* := \tilde{\mathbf{v}}_{\tilde{\mathcal{J}}}/\|\tilde{\mathbf{v}}_{\tilde{\mathcal{J}}}\|_2$, and $\langle \tilde{\mathbf{v}}_{\tilde{\mathcal{J}}}, \tilde{\mathbf{b}}^*\rangle = \|\tilde{\mathbf{v}}_{\tilde{\mathcal{J}}}\|_2$. Maximizing over $\tilde{\mathcal{J}}$ is achieved by selecting the $s$ entries of $\tilde{\mathbf{v}} := (\tilde{\mathbf{v}})_{i=1}^K$ with largest absolute value (Proposition A.1). Going back to the original variables gives the result. □

## APPENDIX B
## LIPSCHITZ MODULUS

To estimate the Lipschitz modulus of the gradient of the smooth part of the objective we write:

$$\begin{aligned}&\left\|\nabla_{\mathbf{S}_j^i} H(\mathbf{L}, \mathbf{S}_1, \mathbf{R}, \lambda^i) - \nabla_{\mathbf{S}_j^i} H(\mathbf{L}, \mathbf{S}_2, \mathbf{R}, \lambda^i))\right\|_F \\ =\ & (\lambda^i)^2 \left\|\mathbf{L}^T\mathbf{L}(\mathbf{S}_1 - \mathbf{S}_2)\mathbf{R}\mathbf{R}^T\right\|_F \\ =\ & (\lambda^i)^2 \left\|(\mathbf{R}\mathbf{R}^T) \otimes (\mathbf{L}^T\mathbf{L}).\text{vec}(\mathbf{S}_1 - \mathbf{S}_2)\right\|_2 \\ \leq\ & (\lambda^i)^2 \|\mathbf{R}\|_2^2 . \|\mathbf{L}\|_2^2 \|\mathbf{S}_1 - \mathbf{S}_2\|_F .\end{aligned}$$

The following quantity is thus a Lipschitz modulus: $L_j(\mathbf{L}, \mathbf{R}, \lambda^i) = (\lambda^i)^2 \|\mathbf{R}\|_2^2 . \|\mathbf{L}\|_2^2$.

## APPENDIX C
## COVERING DIMENSION

The covering number $\mathcal{N}(\mathcal{A}, \epsilon)$ of a set $\mathcal{A}$ is the minimum number of balls of radius $\epsilon$ needed to cover it. The precise definition of covering numbers is given in [20]. The upper-box counting dimension of the set, loosely referred to as the covering dimension in the text is $d(\mathcal{A}) = \lim_{\epsilon\to 0} \frac{\log \mathcal{N}(\mathcal{A},\epsilon)}{\log 1/\epsilon}$. We are interested in the covering dimension of the set of FA$\mu$STs $\mathcal{D}_{\text{spfac}}$. We begin with the elementary sets $\mathcal{E}_j = \{\mathbf{A} \in \mathbb{R}^{a_j \times a_{j+1}} : \|\mathbf{A}\|_0 \leq s_j, \|\mathbf{A}\|_F = 1\}$. These sets can be seen otherwise as sets of sparse normalized vectors of size $a_j \times a_{j+1}$. This leads following [20] (balls defined wrt the Frobenius norm) to:

$$\mathcal{N}(\mathcal{E}_j, \epsilon) \leq \binom{a_j a_{j+1}}{s_j} \left(1 + \frac{2}{\epsilon}\right)^{s_j}.$$

Defining $\mathcal{M} := \mathcal{E}_1 \times \ldots \times \mathcal{E}_J$ and using [20, lemma 16] gives $\mathcal{N}(\mathcal{M}, \epsilon) \leq \prod_{j=1}^J \binom{a_j a_{j+1}}{s_j} (1 + \frac{2}{\epsilon})^{s_j}$ (wrt to the max metric over the index $j$ using the Frobenius norm). Using $\sum_{j=1}^J \|x_j - y_j\|_F \leq J \max_j \|x_j - y_j\|_F$ gives $\mathcal{N}(\mathcal{M}, \epsilon) \leq \prod_{j=1}^J \binom{a_j a_{j+1}}{s_j} (1 + \frac{2J}{\epsilon})^{s_j}$ wrt to the metric defined by $\rho(x, y) = \sum_{j=1}^J \|x_j - y_j\|_F$. Defining the mapping:

$$\begin{aligned}\Phi :\ & \mathcal{M} := \mathcal{E}_1 \times \ldots \times \mathcal{E}_J \ \to \mathcal{D}_{\text{spfac}} \\ & (\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_J) \ \mapsto \mathbf{D}_J \ldots \mathbf{D}_2 \mathbf{D}_1,\end{aligned}$$

where $\mathcal{D}_{\text{spfac}}$ is the set of FA$\mu$STs of interest, and using the distance measures $\rho(x, y) = \sum_{j=1}^J \|x_j - y_j\|_F$ in $\mathcal{M}$ and $\rho_1(x, y) = \|x - y\|_F$ in $\mathcal{D}_{\text{spfac}}$ we get that the mapping $\Phi$ is a contraction (by induction). We can conclude that:

$$\mathcal{N}(\mathcal{D}_{\text{spfac}}, \epsilon) \leq \prod_{j=1}^J \binom{a_j a_{j+1}}{s_j} \left(1 + \frac{2J}{\epsilon}\right)^{s_j},$$

wrt to the $\rho_1$ metric.

Using $\binom{n}{p} \leq \frac{n^p}{p!}$ and $n! \geq \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$ yields $\mathcal{N}(\mathcal{D}_{\text{spfac}}, \epsilon) \leq \prod_{j=1}^J \frac{1}{\sqrt{2\pi s_j}} \left(\frac{e}{s_j} a_j a_{j+1} (1 + \frac{2J}{\epsilon})\right)^{s_j}$.

Defining $C_j = \frac{e.a_j.a_{j+1}.(2J+1)}{s_j . \sqrt[2s_j]{2\pi s_j}}$, we have $\mathcal{N}(\mathcal{D}_{\text{spfac}}, \epsilon) \leq (\frac{C}{\epsilon})^h$, with $h = \sum_{j=1}^J s_j$ and $C = \max_j C_j = \max_j \frac{e.a_j.a_{j+1}.(2J+1)}{s_j . \sqrt[2s_j]{2\pi s_j}}$. The upper-box counting dimension thus satisfies $d(\mathcal{D}_{\text{spfac}}) \leq h = \sum_{j=1}^J s_j = s_{tot}$.

## ACKNOWLEDGMENT

The authors wish to thank François Malgouyres and Olivier Chabiron for fruitful discussions that helped in producing this work. The authors also express their gratitude to Alexandre Gramfort for providing the MEG data and contributing to [17].

## REFERENCES

[1] L. Le Magoarou and R. Gribonval, "Learning computationally efficient dictionaries and their implementation as fast transforms," *CoRR*, vol. abs/1406.5388, 2014. [Online]. Available: http://arxiv.org/abs/1406.5388

[2] L. Le Magoarou and R. Gribonval, "Chasing butterflies: In search of efficient dictionaries," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, April 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01104696

[3] J. Cooley and J. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

[4] J. Shanks, "Computation of the fast walsh-fourier transform," *Computers, IEEE Transactions on*, vol. C-18, no. 5, pp. 457–459, May 1969.

[5] W.-H. Chen, C. Smith, and S. Fralick, "A fast computational algorithm for the discrete cosine transform," *Communications, IEEE Transactions on*, vol. 25, no. 9, pp. 1004–1009, Sep 1977.

[6] S. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 11, no. 7, pp. 674–693, Jul 1989.

[7] J. Morgenstern, "The linear complexity of computation," *J. ACM*, vol. 22, no. 2, pp. 184–194, Apr. 1975. [Online]. Available: http://doi.acm.org/10.1145/321879.321881

[8] S. Mallat and Z. Zhang, "Matching pursuits with time-frequency dictionaries," *Signal Processing, IEEE Transactions on*, vol. 41, no. 12, pp. 3397–3415, Dec 1993.

[9] I. Daubechies, M. Defrise, and C. De Mol, "An iterative thresholding algorithm for linear inverse problems with a sparsity constraint," *Communications on Pure and Applied Mathematics*, vol. 57, no. 11, pp. 1413–1457, 2004. [Online]. Available: http://dx.doi.org/10.1002/cpa.20042

[10] J. Tropp and A. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *Information Theory, IEEE Transactions on*, vol. 53, no. 12, pp. 4655–4666, Dec 2007.

[11] T. Blumensath and M. E. Davies, "Iterative thresholding for sparse approximations," *Journal of Fourier Analysis and Applications*, vol. 14, no. 5-6, pp. 629–654, 2008. [Online]. Available: http://dx.doi.org/10.1007/s00041-008-9035-z

[12] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Img. Sci.*, vol. 2, no. 1, pp. 183–202, Mar. 2009. [Online]. Available: http://dx.doi.org/10.1137/080716542

[13] R. Rubinstein, A. Bruckstein, and M. Elad, "Dictionaries for Sparse Representation Modeling," *Proceedings of the IEEE*, vol. 98, no. 6, pp. 1045 –1057, 2010. [Online]. Available: http://hal.inria.fr/inria-00565811

[14] R. Rubinstein, M. Zibulevsky, and M. Elad, "Double sparsity: Learning sparse dictionaries for sparse signal approximation," *Signal Processing, IEEE Transactions on*, vol. 58, no. 3, pp. 1553–1564, March 2010.

[15] O. Chabiron, F. Malgouyres, J.-Y. Tourneret, and N. Dobigeon, "Toward fast transform learning," Tech. Rep., Nov. 2013. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00862903

[16] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains," *Signal Processing Magazine, IEEE*, vol. 30, no. 3, pp. 83–98, 2013.

[17] L. Le Magoarou, R. Gribonval, and A. Gramfort, "FA$\mu$ST: speeding up linear transforms for tractable inverse problems," in *EUSIPCO* , Nice, France, Aug. 2015. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01156478

[18] J. Bolte, S. Sabach, and M. Teboulle, "Proximal Alternating Linearized Minimization for nonconvex and nonsmooth problems," *Mathematical Programming*, pp. 1–36, 2013. [Online]. Available: http://dx.doi.org/10.1007/s10107-013-0701-9

[19] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–. [Online]. Available: http://www.scipy.org/

[20] R. Gribonval, R. Jenatton, F. Bach, M. Kleinsteuber, and M. Seibert, "Sample complexity of dictionary learning and other matrix factorizations," *Information Theory, IEEE Transactions on*, vol. 61, no. 6, pp. 3469–3486, June 2015.

[21] W. Hackbusch, "A Sparse Matrix Arithmetic Based on H-matrices. Part I: Introduction to H-matrices," *Computing*, vol. 62, no. 2, pp. 89–108, May 1999. [Online]. Available: http://dx.doi.org/10.1007/s006070050015

[22] E. J. Candès, L. Demanet, and L. Ying, "A fast butterfly algorithm for the computation of Fourier integral operators." *Multiscale Modeling & Simulation*, vol. 7, no. 4, pp. 1727–1750, 2009.

[23] I. Tosic and P. Frossard, "Dictionary learning," *Signal Processing Magazine, IEEE*, vol. 28, no. 2, pp. 27–38, March 2011.

[24] A. M. Bruckstein, D. L. Donoho, and M. Elad, "From Sparse Solutions of Systems of Equations to Sparse Modeling of Signals and Images," *SIAM Review*, vol. 51, no. 1, pp. 34–81, Feb. 2009.

[25] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proceedings of the 27th Annual International Conference on Machine Learning*, ser. ICML '10, 2010, pp. 399–406.

[26] P. Sprechmann, A. M. Bronstein, and G. Sapiro, "Learning efficient sparse and low rank models," *CoRR*, vol. abs/1212.3631, 2012.

[27] A. Makhzani and B. Frey, "k-sparse autoencoders," *CoRR*, vol. abs/1312.5663, 2013. [Online]. Available: http://arxiv.org/abs/1312.5663

[28] A. B. Lee, B. Nadler, and L. Wasserman, "Treelets - an adaptive multi-scale basis for sparse unordered data," *The Annals of Applied Statistics*, vol. 2, no. 2, pp. 435–471, July 2008.

[29] G. Cao, L. Bachega, and C. Bouman, "The sparse matrix transform for covariance estimation and analysis of high dimensional signals," *Image Processing, IEEE Transactions on*, vol. 20, no. 3, pp. 625–640, 2011.

[30] S. Lyu and X. Wang, "On algorithms for sparse multi-factor NMF," in *Advances in Neural Information Processing Systems 26*, 2013, pp. 602–610.

[31] B. Neyshabur and R. Panigrahy, "Sparse matrix factorization," *CoRR*, vol. abs/1311.3315, 2013.

[32] S. Arora, A. Bhaskara, R. Ge, and T. Ma, "Provable bounds for learning some deep representations," *CoRR*, vol. abs/1310.6343, 2013.

[33] V. G. Risi Kondor, Nedelina Teneva, "Multiresolution matrix factorization," *JMLR*, vol. 32 (1), pp. 1620–1628, 2014.

[34] R. Rustamov and L. Guibas, "Wavelets on graphs via deep learning," in *Advances in Neural Information Processing Systems 26*. Curran Associates, Inc., 2013, pp. 998–1006. [Online]. Available: http://papers.nips.cc/paper/5046-wavelets-on-graphs-via-deep-learning.pdf

[35] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *Advances in neural information processing systems*, vol. 19, p. 153, 2007.

[36] Y. Bengio, "Learning deep architectures for ai," *Found. Trends Mach. Learn.*, vol. 2, no. 1, pp. 1–127, Jan. 2009. [Online]. Available: http://dx.doi.org/10.1561/2200000006

[37] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006. [Online]. Available: http://www.sciencemag.org/content/313/5786/504.abstract

[38] S. Haufe, V. V. Nikulin, A. Ziehe, K.-R. Müller, and G. Nolte, "Combining sparsity and rotational invariance in EEG/MEG source reconstruction," *NeuroImage*, vol. 42, no. 2, pp. 726–738, August 2008.

[39] A. Gramfort, M. Kowalski, and M. S. Hämäläinen, "Mixed-norm estimates for the M/EEG inverse problem using accelerated gradient methods," *Phys. Med. Biol.*, vol. 57, no. 7, pp. 1937–1961, Apr. 2012.

[40] A. Gramfort, M. Luessi, E. Larson, D. A. Engemann, D. Strohmeier, C. Brodbeck, L. Parkkonen, and M. S. Hämäläinen, "MNE software for processing MEG and EEG data ," *NeuroImage*, vol. 86, no. 0, pp. 446 – 460, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1053811913010501

[41] S.-J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky, "An interior-point method for large-scale l1-regularized least squares," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 1, no. 4, pp. 606–617, Dec 2007.

[42] K. Engan, S. Aase, and J. Hakon Husoy, "Method of optimal directions for frame design," in *Acoustics, Speech, and Signal Processing, 1999. Proceedings., 1999 IEEE International Conference on*, vol. 5, 1999, pp. 2443–2446 vol.5.

[43] M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *Signal Processing, IEEE Transactions on*, vol. 54, no. 11, pp. 4311–4322, Nov 2006.

[44] J. Mairal, F. Bach, J. Ponce, and G. Sapiro, "Online learning for matrix factorization and sparse coding," *Journal of Machine Learning Research*, vol. 11, no. 1, pp. 19–60, Jan. 2010. [Online]. Available: http://hal.inria.fr/inria-00408716

[45] R. Rubinstein, M. Zibulevsky, and M. Elad, "Efficient Implementation of the K-SVD Algorithm using Batch Orthogonal Matching Pursuit," Tech. Rep., 2008.

[46] M. Elad and M. Aharon, "Image denoising via sparse and redundant representations over learned dictionaries," *Image Processing, IEEE Transactions on*, vol. 15, no. 12, pp. 3736–3745, Dec 2006.

[47] "http://www.imageprocessingplace.com/."

[48] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *Image Processing, IEEE Transactions on*, vol. 16, no. 8, pp. 2080–2095, Aug 2007.

[49] D. Vainsencher, S. Mannor, and A. M. Bruckstein, "The sample complexity of dictionary learning," *The Journal of Machine Learning Research*, vol. 12, pp. 3259–3281, 2011.

[50] A. Maurer and M. Pontil, "K-Dimensional Coding Schemes in Hilbert Spaces," *Information Theory, IEEE Transactions on*, vol. 56, no. 11, pp. 5839–5846, Nov 2010.