Machine Learning
https://doi.org/10.1007/s10994-022-06260-2

Matrix-wise ℓ_0-constrained sparse nonnegative least squares

Nicolas Nadisic[1] · **Jeremy E. Cohen**[2] · **Arnaud Vandaele**[1] · **Nicolas Gillis**[1]

Received: 15 February 2022 / Revised: 23 June 2022 / Accepted: 25 September 2022

Abstract
Nonnegative least squares problems with multiple right-hand sides (MNNLS) arise in models that rely on additive linear combinations. In particular, they are at the core of most nonnegative matrix factorization algorithms and have many applications. The nonnegativity constraint is known to naturally favor sparsity, that is, solutions with few nonzero entries. However, it is often useful to further enhance this sparsity, as it improves the interpretability of the results and helps reducing noise, which leads to the sparse MNNLS problem. In this paper, as opposed to most previous works that enforce sparsity column- or row-wise, we first introduce a novel formulation for sparse MNNLS, with a matrix-wise sparsity constraint. Then, we present a two-step algorithm to tackle this problem. The first step divides sparse MNNLS in subproblems, one per column of the original problem. It then uses different algorithms to produce, either exactly or approximately, a Pareto front for each subproblem, that is, to produce a set of solutions representing different tradeoffs between reconstruction error and sparsity. The second step selects solutions among these Pareto fronts in order to build a sparsity-constrained matrix that minimizes the reconstruction error. We perform experiments on facial and hyperspectral images, and we show that our proposed two-step approach provides more accurate results than state-of-the-art sparse coding heuristics applied both column-wise and globally.

Keywords Nonnegative least squares · Sparsity · Nonnegative matrix factorization

1 Introduction

Nonnegative least squares (NNLS) problems arise in many applications where data points can be represented as additive linear combinations of meaningful components (Lee and Seung 1997). For instance,

Editors: Krzysztof Dembczynski and Emilie Devijver.

✉ Nicolas Nadisic
nicolas.nadisic@umons.ac.be

[1] Department of Mathematics and Operational Research, University of Mons, Mons, Belgium

[2] INSA-Lyon, UCBL, UJM-Saint Etienne, CNRS, Inserm, CREATIS UMR 5220, U1206, Univ Lyon, 69100 Villeurbanne, France

Published online: 15 November 2022

- In facial images, the faces are the nonnegative linear combination of facial features such as eyes, noses and lips (Lee and Seung 1999).
- In hyperspectral images, the spectral signature of a pixel is the nonnegative linear combination of the spectral signature of the materials it contains (Bioucas-Dias et al. 2012).

NNLS problems are also at the core of most approaches to solve nonnegative matrix factorization (NMF); see (Gillis (2020), Chapter 8) and the references therein. The standard NNLS problem can be formulated as follows: given a dictionary matrix $A \in \mathbb{R}^{m \times r}$ and a data vector $b \in \mathbb{R}^m$, solve

$$\min_x \|Ax - b\|_2^2 \quad \text{such that} \quad x \geq 0. \tag{1}$$

Note that (1) is a convex problem.

1.1 Sparsity and NNLS

The nonnegativity constraint is known to naturally produce sparse solutions, that is, solutions with few non-zero entries (Foucart and Koslicki 2014). Sparsity often improves the interpretability of the results by modelling data points as combinations of only a few components. For example, in hyperspectral unmixing, that is, the task of identifying materials in a hyperspectral image, sparsity means that a pixel contains only a few materials.

A natural sparsity measure is the ℓ_0-"norm", defined as the number of non-zero entries in a given vector, $\|x\|_0 = |\{i : x_i \neq 0\}|$. Given a positive integer k, a vector x is said to be k-sparse if $\|x\|_0 \leq k$.

Unfortunately, the sparsity of the solution to an NNLS problem is not guaranteed in general, whereas controlling it can be helpful in many applications. For this reason, numerous techniques have been developed to favor sparsity.

A sparsity-constrained variant of Problem (1), referred to as *k-sparse NNLS*, is the following

$$\min_x \|Ax - b\|_2^2 \quad \text{such that} \quad x \geq 0 \ \text{ and } \ \|x\|_0 \leq k. \tag{2}$$

Several algorithms exist to tackle Problem (2), either exactly or approximately; we detail them in Sect. 2.

In hyperspectral unmixing, this k-sparsity constraint implies that a pixel can be composed of at most k materials. Although this formulation is intuitive, in some cases setting the parameter k is not straightforward. Therefore, we can also consider a *biobjective* formulation where the objectives are, on the one hand, to minimize the reconstruction error, and on the other hand, to maximize the sparsity (that is, minimize the ℓ_0-"norm"),

$$\min_{x \geq 0} \{\|Ax - b\|_2^2, \|x\|_0\}. \tag{3}$$

As sparser solutions lead to higher error, these objectives are conflicting, so there is not an optimal solution to Problem (3) and we need a trade-off between the two objectives. Thus, we seek *Pareto-optimal* solutions.

Given different objectives to optimize, a solution x is said to be Pareto-optimal if there does not exist any solution which is at least as good as x on all objectives and strictly better than x on at least one objective. The set of all Pareto-optimal solutions for a given problem is called the *Pareto front*, see Fig. 1.

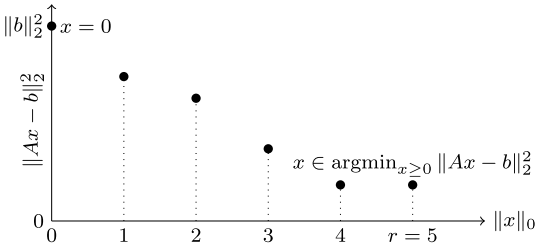


Fig. 1 Example of the Pareto front for a biobjective k-sparse NNLS problem with $r = 5$ variables. The first solution, for $\|x\|_0 = 0$, corresponds to the zero vector. The last solution, for $\|x\|_0 = 5$, corresponds to the NNLS problem with no sparsity constraint. Here the penultimate solution is identical to the last one, meaning that the solution with no sparsity constraint has naturally 1 zero entry

Here, the discreteness of the ℓ_0-"norm" implies that solving Problem (3) conceptually reduces to solving Problem (2) for all possible values of k. To the best of our knowledge, there exist only one algorithm to solve Problem (3) exactly (Nadisic et al. 2021). We will present it in Sect. 2.1. It is also possible to modify some algorithms originally intended for k-sparse NNLS so that they generate an approximation of the Pareto front; we will see examples with greedy algorithms and the homotopy algorithm, respectively in Sects. 2.2 and 2.3.

1.2 Sparsity in NNLS problems with multiple right-hand sides

In many cases, one has to deal with NNLS problems with multiple right-hand sides (MNNLS), that is, problems of the form

$$\min_X \|B - AX\|_F^2 \quad \text{such that} \quad X \geq 0, \tag{4}$$

where $B \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times r}$, and $X \in \mathbb{R}^{r \times n}$. Given a matrix $B \in \mathbb{R}^{m \times n}$, we note $\|B\|_F$ its Frobenius norm, that is $\|B\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n B(i,j)^2}$ where $B(i, j)$ is the entry of B at position (i, j). We note $B(:, j)$ the jth column of the matrix B. Problem (4) can be decomposed into n NNLS subproblems of the form (1), where $B(:, j)$, A, and $X(:, j)$ correspond to b, A, and x, respectively. For example, in the unmixing of a hyperspectral image, every column $B(:, j)$ represents a pixel, and the corresponding column $X(:, j)$ represents its composition, in terms of the abundances of the r materials whose spectral signatures are the columns of A. Note that in this work, for the sake of conciseness, we focus only on the (sparse) optimization of X, but all concepts and algorithms can be applied symmetrically on A.

This is closely related to the nonnegative matrix factorization (NMF) problem, of the form

$$\min_{A,X} \|B - AX\|_F^2 \quad \text{such that} \quad A \geq 0 \text{ and } X \geq 0, \tag{5}$$

in which we aim to find the factors A and X, given B and a factorization rank r. The usual optimization scheme for NMF consists in alternatively optimizing one factor while fixing the other, which is equivalent to solving MNNLS subproblems. Note that in this paper, we focus on MNNLS rather than NMF.

To encourage sparsity in MNNLS, one can apply a sparse NNLS model column-wise, leading to

$$\min_X \|B - AX\|_F^2 \quad \text{such that} \quad X \geq 0 \text{ and } \|X(:,j)\|_0 \leq k \text{ for all } j. \tag{6}$$

Solving Problem (6) boils down to solving n independent subproblems of the form (2). However, in some applications, setting the sparsity parameter k is tricky, as the relevant value can vary for different columns. For example, in hyperspectral unmixing, pixels will be composed of different numbers of materials. Therefore, one can consider a more global approach, such as

$$\min_X \|B - AX\|_F^2 \quad \text{such that} \quad X \geq 0 \text{ and } \|X\|_0 \leq q, \tag{7}$$

where $\|X\|_0 = \sum_j \|X(:,j)\|_0$ and q is a matrix-wise sparsity parameter, hence enforcing an *average* sparsity q/n for the columns of X. In the following, Problem (7) is called the matrix-wise q-sparse MNNLS problem, and solving it is the main focus of this paper.

Note that (7) could theoretically be solved by any column-wise k-sparse NNLS algorithm, because (7) is equivalent to the vectorized form

$$\min_{\bar{x}} \|\text{vec}(B) - \underbrace{(A \otimes I)}_{\Omega} \bar{x}\|_2^2 \text{ such that } \bar{x} \geq 0 \text{ and } \|\bar{x}\|_0 \leq q, \tag{8}$$

where $\otimes$ is the Kronecker product, I is the identity matrix of appropriate dimension, and $\text{vec}(B)$ denotes the column vector obtained by stacking the columns of B on top of one another. Problem (8) is a k-sparse NNLS problem, but in practice its dimensions make it difficult to solve directly. Denoting $\Omega = A \otimes I$, we have $\Omega \in \mathbb{R}^{(mn)\times(rn)}$, which is particularly problematic in hyperspectral unmixing where the dimension n can reach tens of thousands.

It is possible to implement some k-sparse NNLS algorithms in a non-naive way to solve (8) efficiently without actually allocating Ω, and we detail such implementation of a greedy algorithm in Sect. 3.2. However, even in this case, when n is large then the problem to solve is huge and the computing time can become too high, see Sect. 4 for an experimental illustration.

1.3 Contribution and outline of the paper

The main goal of this work is to describe a novel method able to solve efficiently the matrix-wise q-sparse MNNLS problem (7), even in large dimensions. This method can be summarized by two main steps:

1. Problem (7) is divided in n subproblems of the form (3) and, for each of them, the Pareto front is computed with existing algorithms.
2. One solution per column (hence per Pareto front) is selected to build a solution to Problem (7), that is, a q-sparse matrix. This combinatorial step is solved exactly with a dedicated algorithm.

To the best of our knowledge, this work is the first to tackle specifically Problem (7). Note that the algorithms used in the first step are not original contributions. The contributions lie rather in the use of these existing algorithms to generate the Pareto fronts of

the subproblems, and the combination of these fronts with a novel algorithm to obtain a q-sparse matrix.

This paper is organized as follows. In Sect. 2, we present existing approaches for sparse MNNLS, and we detail the three algorithms used to generate Pareto fronts. In Sect. 3, we present the main contribution of this work, that is, an algorithm to solve Problem (7). We illustrate the effectiveness of our proposed method with experiments on real-world facial and hyperspectral image datasets and on synthetic datasets in Sect. 4. We conclude in Sect. 5.

2 Related work

Most approaches that tackle sparse MNNLS were actually introduced in the context of sparse NMF. Since its very introduction by Lee and Seung (1999), NMF is appreciated for the sparsity of the produced factors. A variety of works have been proposed to further enhance this sparsity, making *sparse NMF* one of the most popular variants of NMF. Many authors worked on the ℓ_1-penalized formulation, notably Hoyer (2002); Eggert and Korner (2004); Kim and Park (2007); Cichocki et al. (2008); Gillis (2012). This formulation uses the ℓ_1-norm as a convex surrogate of the ℓ_0-"norm" to ease the computation, but it presents several disadvantages, see Sect. 2.3 for a detailed explanation.

To avoid the issues linked to the ℓ_1-penalty, Hoyer (2004) introduced a more explicit sparsity measure based on the ratio between the ℓ_1-norm and the ℓ_2-norm, and he considered an NMF variant with a column-wise constraint on this measure. Other works considered the ℓ_0-"norm" formulation, that can be decomposed into a series of k-sparse NNLS subproblems. We can cite Aharon et al. (2005); Morup et al. (2008); Peharz and Pernkopf (2012). Cohen and Gillis (2019) proposed a method that solves *exactly* the k-sparse NNLS subproblems using a bruteforce approach. Nadisic et al. (2020) extended this work by replacing the bruteforce subroutine by a dedicated branch-and-bound algorithm. To the best of our knowledge, no existing work considered a matrix-wise ℓ_0 constraint.

Some similar models have been studied, such as simultaneous sparse approximation (Tropp et al. 2006; Stojnic et al. 2009), where X is constrained to be block-sparse, that is, to have sparse columns sharing the same support. The assumptions of these models, the algorithms to solve them, and their applications are far from our focus, so detailing them is out of the scope of this article.

In the following, we detail three types of sparse NNLS algorithms on which we will rely to generate Pareto fronts, that is, to solve Problem (3).

2.1 Arborescent

The algorithm Arborescent (Nadisic et al. 2020) is a branch-and-bound algorithm designed to solve exactly k-sparse NNLS problems. In a nutshell, Arborescent enumerates the possible supports (that is, the possible patterns of zeros) on a search tree, as shown in Fig. 2.

Every node represents an over-support $\mathcal{K}$, that is, the set of entries that are not constrained to 0. Exploring a node means solving the NNLS subproblem

$$f^*(\mathcal{K}) = \min ||A(:, \mathcal{K})x(\mathcal{K}) - b||_2^2 \text{ such that } x(\mathcal{K}) \geq 0, \tag{9}$$

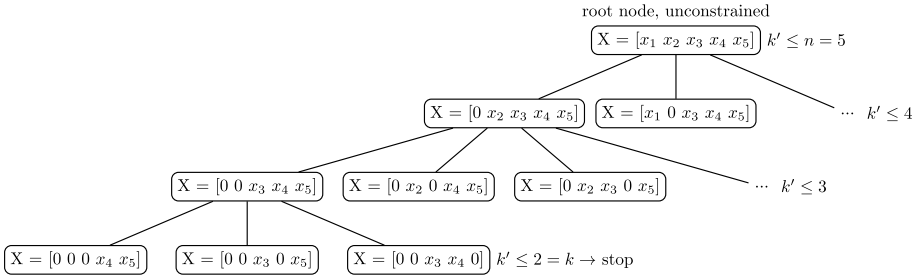


Fig. 2 Example of the search tree explored by Arborescent, for $n = 5$ and $k = 2$

where $x(\mathcal{K})$ denotes the subvector of x with indices in the set $\mathcal{K}$. This subproblem can be solved with any standard NNLS solver, and here it is done with an *active-set* algorithm (Portugal et al. 1994), which provides an exact solution in a finite number of iterations. The value $f^*(\mathcal{K})$ is the *error* associated with the node corresponding to $\mathcal{K}$. To prune this tree, Arborescent uses the fact that in any optimization problem, when adding constraints, the solution cannot improve. By doing a depth-first exploration, we can quickly find feasible solutions and then prune efficiently large parts of the search space.

An extension of this algorithm (Nadisic et al. 2021) computes exactly the Pareto front corresponding to the biobjective problem (3). It is based on the fact that, when computing the k-sparse solution to an NNLS problem, Arborescent also computes all k'-sparse solutions for $k' \in \{k, \dots, r\}$. If we set $k = 1$, then we compute the entire Pareto front. If $k > 1$, we compute only a portion of it.

This algorithm is fast in practice when the dimension r is small, which is generally the case in hyperspectral unmixing. However, it is still computationally expensive and quite slow for problems of large dimensions, when r is larger than a few tens. For this reason, practitioners often prefer other sparsity-inducing approaches, such as greedy algorithms or ℓ_1-regularization.

Other works tackled the ℓ_0-constrained problem exactly, but without nonnegativity constraints, see for example Ben Mhenni et al. (2021) and the references therein. It may be possible to adapt them in the nonnegative setting, but this is still an open problem and out of the scope of this article. To best of our knowledge, no other work considered computing the Pareto front of the biobjective sparse problem.

2.2 Greedy algorithms

Greedy algorithms are one of the most popular approaches for solving Problem (2). They start with an empty support ($x_i = 0$ for all i), and select components one by one to enrich the support, until the target sparsity k is reached. The selection of a component is done greedily by choosing the component minimizing the residual error. Orthogonal versions of these methods, such as *Orthogonal Least Squares* (OLS) (Chen et al. 1989) and *Orthogonal Matching Pursuit* (OMP) (Pati et al. 1993) make sure a component can be selected only once. Nonnegative variants have been recently proposed; see for example Nguyen et al. (2019) and the references therein. In general, these algorithms do not give the globally optimal solution. Theoretical recovery guarantees exist, but they are restrictive (Tropp 2004; Soussen et al. 2013).

Interestingly, because they select components one after the other, greedy algorithms can be used as a proxy to compute an approximation of the Pareto front of the corresponding biobjective sparse NNLS problem. Indeed, the solution at the i-th iteration of the algorithm is i-sparse. By running the algorithm with a sparsity target k, we also compute, as a side effect, some k'-sparse solutions for $k' \in \{1, \dots, k\}$. Therefore, to obtain an approximation of the Pareto front, it suffices to return all intermediate solutions instead of only the final one.

In this paper, we will only focus on Nonnegative OMP (NNOMP) for conciseness, but this approach could be easily generalized to similar greedy algorithms. NNOMP was first introduced by Bruckstein et al. (2008), but many variants exist, and implementations details can have a significant impact on the performance of these algorithms. Reviewing them is out of the scope of this article, and we refer the interested reader to Nguyen et al. (2019).

2.3 Homotopy

The ℓ_1-norm, defined as $\|x\|_1 = \sum_{i=1}^{r} |x_i|$, is a convex surrogate of the ℓ_0-"norm", it is therefore easier to optimize while being able to promote sparsity. The ℓ_1-regularization consists in penalizing the solution in the objective function of (1), leading to the following problem, referred to as ℓ_1-NNLS,

$$\min_{x \in \mathbb{R}^r} \|Ax - b\|_2^2 + \lambda \|x\|_1 \quad \text{such that} \quad x \geq 0. \tag{10}$$

Note that without the nonnegativity constraint, this is the well-known LASSO model (Tibshirani 1996). Problem (10) can be thought of as the weighted-sum form of a biobjective problem, where the objectives are minimizing the reconstruction error $\|Ax - b\|_2^2$ on one hand, and minimizing the ℓ_1-norm of the solution $\|x\|_1$ on the other hand. The parameter λ controls the trade-off between the two objectives.

Despite its popularity, this technique suffers from several drawbacks. In particular, there is no explicit relation between the parameter λ and the sparsity of the solution, hence choosing an appropriate value for λ can be tricky, and often involves a tedious trial-and-error process. Also, the ℓ_1-penalty introduces a bias. Although there exist theoretical guarantees for support recovery, such as the *Exact Recovery Condition* (Tropp 2006), they are restrictive and often not realistic in practice.

To overcome these issues, *homotopy* algorithms have been introduced. They generate the full *regularization path* of a given ℓ_1-NNLS problem, that is, the set of the solutions for all possible values of λ. They allow the user to choose the relevant solution within this path, each solution representing a different trade-off between sparsity and reconstruction error. The first homotopy algorithm has been introduced by Osborne et al. (2000), for sparse least squares with no nonnegativity constraint. Variants have been developed by Efron et al. (2004) and Donoho and Tsaig (2008). Kim et al. (2013) introduced a variant to deal specifically with the nonnegativity constraint.

In a nutshell, the homotopy algorithm uses the KKT conditions (necessary conditions for optimality) to first find the value $\lambda_{\max}$ for which the optimal solution of ℓ_1-NNLS is $x = 0$ for any $\lambda \geq \lambda_{\max}$, and then to compute the next smaller values of λ for which the support (that is, the set of non-zero entries) of the optimal solution changes (one zero entry becomes non-zero, or the other way around). This is similar in spirit to active-set methods, akin to the simplex algorithm for linear programming. These values of λ are called

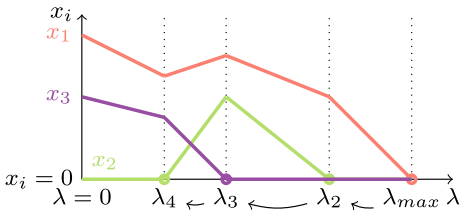


Fig. 3 Example of a solution path of a homotopy algorithm, depending on λ, for a problem of 3 variables. Vertical dotted lines correspond to breakpoints

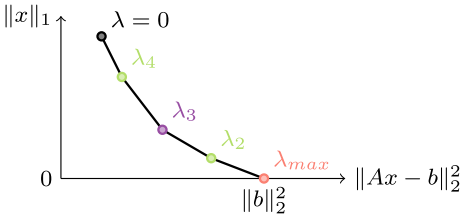


Fig. 4 Trade-off between the ℓ_1 -norm and the error $\|Ax - b\|_2^2$ corresponding to the solution path in Fig. 3

breakpoints, between which the support of the optimal solution does not change; see Figs. 3 and 4 for an illustration.

Although the nonnegative homotopy algorithm is not an original contribution, we include in Appendix 1 its detailed description and justification.

The strength of the homotopy algorithm is to generate the full *regularization path*, that is, the set of optimal solution for all possible values of λ, for the same cost as a standard active-set algorithm. The solutions in this path represent different tradeoffs between reconstruction error and sparsity, and as such this path can be seen as an approximation of the Pareto front in Problem (3).

3 Solving matrix-wise q-sparse MNNLS

In this section, we study how to tackle matrix-wise q-sparse MNNLS, that is, Problem (7). First, we present the key contribution of this work, that is a two-step algorithm to solve Problem (7). Then, we show how the greedy algorithm NNOMP can be implemented specifically for Problem (7) to avoid the costly reformulation from Eq. 8.

3.1 Our key contribution: a two-step algorithm for matrix-wise *q*-sparse MNNLS

In this section we present the main contribution of this paper, that is, a two-step algorithm to solve Problem (7). This algorithm is called Salmon[1] and it is detailed in Algorithm 1. The motivation behind it is to divide the large matrix-wise problem into small one-column subproblems, solve them, and then combine their solutions to build a global solution.

[1] The name stands for SALMON Applies ℓ_0-constraints Matrix-wise On NNLS problems.

Algorithm 1: The algorithm Salmon

Input: Data matrix $B \in \mathbb{R}^{m\times n}$, dictionary $A \in \mathbb{R}^{m\times r}$, sparsity target $q \in \mathbb{N}$
Output: $X^* \in \mathbb{R}^{r\times n}$

1 Init matrix $C \in \mathbb{R}^{r\times n}$, for all i and j, $C(i,j) \leftarrow \infty$
2 Init solutions $Sol_{i,j}$ for all i and j
3 **for each** $j \in \{1,\dots,n\}$
4 $\{x_t^*\}_{\forall t} \leftarrow$ front_generator(A,$B(:,j)$) `/* Arborescent, NNOMP, or homotopy */`
5 **for each** t
6 $k \leftarrow \|x_t^*\|_0$
7 $err \leftarrow \|B(:,j) - Ax_t^*\|_2^2$
8 **for each** $i \in \{k,\dots,r\}$
9 **if** $err < C(i,j)$ **then**
10 $C(i,j) \leftarrow err$
11 $Sol_{i,j} \leftarrow x_t^*$
12 Init cursors, for all $j \in \{1,\dots,n\}, k_j \leftarrow 0$
13 Init matrix $\mathcal{G} \in \mathbb{R}^{r\times n}$
14 **for each** $i \in \{1,\dots,r\}, j \in \{1,\dots,n\}$
15 $\mathcal{G}(i,j) \leftarrow (C(0,j) - C(i,j))/i$
16 **while** $\sum_j k_j < q$ **and** $\max_{(i,j)} \mathcal{G}(i,j) > 0$ **do**
17 $i^*, j^* \leftarrow \operatorname{argmax}_{(i,j)} \mathcal{G}(i,j)$
18 $\delta \leftarrow i^* - k_{j^*}$
19 $k_{j^*} \leftarrow i^*$
20 **for each** $i \in \{1,\dots,i^*\}$
21 $\mathcal{G}(i,j^*) \leftarrow 0$
22 **for each** $i \in \{i^*+1,\dots,r\}$
23 $\mathcal{G}(i,j^*) \leftarrow \frac{C(i^*,j^*)-C(i,j^*)}{i-i^*}$
24 **if** $\sum_j k_j > q - r + 1$ **then**
25 **for each** (i,j) such that $i \in \{k_j+1,\dots,\min(r, k_j + q - \sum_{l>j} k_l)\}$
26 $\mathcal{G}(i,j) \leftarrow 0$
27 **for each** $j \in \{1,\dots,n\}$
28 $X^*(:,j) \leftarrow Sol_{k_j,j}$

Step 1 corresponds to lines 1 to 11. It consists in, given the data matrix B and the dictionary A, running an algorithm to generate a Pareto front for every column of B, that is, with input A and $b = B(:\,j)$ for all j. This can be done by any of the three Pareto-front-generating methods presented in Sect. 2; exactly with Arborescent and approximately with NNOMP and the homotopy algorithm. From these fronts, we build a cost matrix C where each column represents a column j of X, each row represents a k-sparsity between 0 and r, and each entry is the reconstruction error of the k-sparse solution of column j. Formally, for all $i \in \{0,\dots,r\}$ and $j \in \{1,\dots,n\}$,

$$C(i,j) \quad \approx \quad \min_{x\geq 0} \|B(:,j) - Ax\|_2^2 \text{ s.t. } \|x\|_0 \leq i,$$

and $Sol_{i,j}$ stores the corresponding argmin, that is the best i-sparse solution for column j.

Remark 1 The algorithms used to generate the Pareto front in step 1 do not necessarily generate one k-sparse solution for a each k; they may generate more than one solution, or no

solution at all for a given k. For example, NNOMP selects columns of A sequentially, but some of them may not have a positive weight when solving the corresponding NNLS. The loop on line 8 ensures that, if there exists some k for which no k-sparse solution is generated, then the $(k-1)$-sparse solution is used instead (or the $(k-2)$-sparse one if no $(k-1)$ -sparse solution is generated, and so on). The condition on line 9 ensures that, if there are several k-sparse solutions for a given k, then only the best one is kept.

Once C is computed, step 2 consists in selecting one solution per column to build the solution matrix X, that is, it consists in choosing the sparsity level for each column of X. This selection step is a combinatorial problem, similar to an assignment problem. Let us define the binary variables $z_{i,j} \in \{0,1\}$ for $i \in \{0,1,\dots r\}$ and $j \in \{1,2,\dots n\}$ such that $z_{i,j} = 1$ if and only if the jth column of X is i-sparse. Note that i can be equal to 0, corresponding the the zero vector. The variable z encodes which sparsity level is selected for each column of X. Given the cost matrix C computed in step 1, step 2 requires to solve the following integer program

$$\begin{aligned} &\min_{z \in \{0,1\}^{r \times n}} \sum_{i,j} z_{i,j} C(i,j) \\ &\text{such that } \sum_{i} z_{i,j} = 1 \text{ for all j, and } \sum_{i,j} i\, z_{i,j} \leq q. \end{aligned} \tag{11}$$

The objective is to minimize the reconstruction error, while the first constraints impose that each column of X has a single sparsity level, and the second that the total number of non-zero entries of X does not exceed q.

We propose a greedy selection algorithm to solve (11), see lines 12 to 28 of Algorithm 1. As we will prove in Theorem 1, this greedy strategy is nearly optimal. It works as follows. For each column j, the scalar k_j indicates its sparsity level at the current iteration, that is, at every iteration, the jth column of X is k_j-sparse. We initialize the algorithm with the 0-sparse solution (the vector of all zeros) for each column (line 12), that is, $k_j = 0$ for all j. Note that $\sum_j k_j$ corresponds to the current sparsity level of the solution.

At each iteration, we will decrease the sparsity of a single column of X. To pick that column in an optimal way, let us define the matrix $\mathcal{G}$, with the same dimensions as C, as follows: at any iteration, the entry $\mathcal{G}(i,j)$ is equal to the potential gain in reconstruction error if the jth column of X goes from k_j-sparse to i-sparse divided by the sparsity difference between these two solutions (that is, $i - k_j$). In particular, at the first iteration (line 15), when $X = 0$, we have

$$\mathcal{G}(i,j) = \frac{C(0,j) - C(i,j)}{i} \quad \text{for all } i,j.$$

Let us denote by (i^*,j^*) the position of the largest entry of $\mathcal{G}$. Given a current solution, the column that will decrease the error $\|B - AX\|_F^2$ the most by decreasing its sparsity is the one corresponding to the column of $\mathcal{G}$ with the largest entry, that is, the j^*th column. Finding this entry is cheap in practice, as we update a sorted list of the maximum entry of each column of $\mathcal{G}$. We denote the quantity δ as the difference in sparsity of the selected column before and after it has been updated, that is, $\delta = i^* - k_{j^*}$. After the value of k_{j^*} has been updated to i^*, we update the entries of the j^*th column of $\mathcal{G}$ accordingly (line 23). Note that $\mathcal{G}(i,j^*) = 0$ for all $i \leq i^*$.

To avoid generating a too dense solution (recall $\sum_j k_j$ must be smaller than q), the procedure on lines 24 to 26 sets to zero the entries of $\mathcal{G}$ whose selection would lead to a total

sparsity $\sum_j k_j$ larger than q. Note that thanks to the condition on line 9, this procedure is executed only for the last few iterations of Salmon. Indeed, the maximum increase δ is r, so this procedure would not be useful as long as $\sum_j k_j \leq q - r$.

When the sum of the sparsity levels equals q, or when all entries of $\mathcal{G}$ are zero, we stop the procedure and we build the final q-sparse solution X by selecting for each column the solution corresponding to its final sparsity level (line 27).

Although the selection algorithm of step 2 is greedy, since we perform an optimal selection at each iteration, it is able to generate a near-optimal solution to Problem (11), as shown below.

Theorem 1 (Near-optimality of the selection step) *Given that C is non-increasing by columns, that is, $C(i,j) \leq C(i',j)$ for all $i' \leq i$ and all j, the proposed selection step of Salmon (lines 12 to 28 of Algorithm 1) computes a near-optimal solution of (11) in the following sense. Denoting*

- $f(z) = \sum_{i,j} z_{i,j} C(i,j)$ the value of the objective function of (11) for a solution z,
- z^* an optimal solution to Problem (11), and
- z_{Sal} the solution computed by the selection step of Salmon,

we have

$$f(z^*) \leq f(z_{Sal}) \leq f(z^*) + \max_j \|C(:,j)\|_\infty.$$

Proof First, note that our proposed greedy algorithm generates a feasible solution of (11) since we make sure that $\sum_j k_j$ remains smaller than q, and hence, by optimality of z^*, $f(z^*) \leq f(z_{Sal})$.

Let us now show that $f(z_{Sal}) \leq f(z^*) + \max_j \|C(:,j)\|_\infty$. Our greedy procedure is similar to a dynamic programming approach. In fact, let us denote the optimal objective function value of (11) as $f^*(q)$ that depends on the parameter q, the global sparsity level allowed; note that $f(z^*) = f^*(q)$. The greedy algorithm is initialized with $z_{0,j} = 1$ for all j, that is, $X = 0$, which is optimal for $q = 0$ (it is the only feasible solution), and hence gives $f^*(0) = \sum_j C(0,j)$. It then progressively decreases the sparsity to reduce the objective the most at each iteration. At each iteration of the greedy algorithm, the support of a single column of X is increased in order to maximize the ratio between the decrease in objective function value and the decrease of sparsity. Since the columns of X do not interact with each other in the objective function and since C is non-increasing in each column, the greedy solution cannot possibly be improved as long as $\delta \leq q - \sum_j k_j$, that is, as long as this optimal way of picking a column is allowed by the global sparsity level. In summary, our greedy algorithm produces intermediate optimal solutions of (11) with objective $f^*(\sum_j k_j + \delta)$ as long as $\sum_j k_j + \delta \leq q$.

The only moment when the greedy algorithm might fall short of global optimality is during the last iterations: if at some point the optimal move is to increase the support of a column in such a way that the total sparsity would exceed q (that is, $\sum_j k_j + \delta > q$), then our greedy algorithm may not be optimal because, to allow that move, we might need to reduce the support of another column, which the greedy approach does not allow. Making that move anyway would generate an optimal solution with global sparsity $q' = \sum_j k_j + \delta > q$ which would not be feasible for (11). Observe that

- $f(z_{Sal}) \leq f^*(q' - \delta)$ since the greedy algorithm keeps improving the $(q' - \delta)$-sparse solution in its next iterations (although possibly not optimally).
- $f^*(q') \geq f^*(q' - \delta) - \max_j \|C(:,j)\|_\infty$ since the move from $q' - \delta$ to q' using the greedy strategy is optimal and, in the worst case, will decrease the sparsity level of a column from r to 0 reducing the error by at most $\max_j \|C(:,j)\|_\infty$.
- $f^*(q') \leq f^*(q) \leq f^*(q' - \delta)$ since $q' - \delta \leq q \leq q'$.

Combining these observations, we obtain

$$f^*(q) = f(z^*) \geq f^*(q') \geq f^*(q' - \delta) - \max_j \|C(:,j)\|_\infty \geq f(z_{Sal}) - \max_j \|C(:,j)\|_\infty,$$

which gives the result.

Note that, in most practical cases, such as hyperspectral unmixing, we have $n \gg r$ and hence $\max_j \|C(:,j)\|_\infty$ is negligible compared to $f(z^*)$. In addition, it is rather unlikely that the greedy algorithm needs to increase the sparsity level of one column from 0 to r at the last step. In fact, in our experiments, we observed that δ is in most cases equal to 1. When δ is equal to 1 in the last r steps, the greedy algorithm is globally optimal (or, more generally, when $\delta = q - \sum_k k_j$ in the last step). For example, for the datasets used in the numerical experiments (Sect. 4) and with the three sparse NNLS algorithms to generate the Pareto fronts (that is, the matrix C), the greedy selection algorithm generated a guaranteed globally optimal solution (that is, $\delta = q - \sum_k k_j$ in the last step) in 19 out of 22 cases.

In practice, the global optimality of Salmon to solve the sparse MNNLS problem (7) therefore heavily relies on step 1. If step 1 is done with Arborescent, then the Pareto fronts are computed optimally and Salmon computes a near-optimal solution to Problem (7). Otherwise, it only computes an approximate solution, although there exist some conditions under which NNOMP or the homotopy algorithm do recover the true Pareto fronts, see Sects. 2.2 and 2.3.

3.1.1 Computational cost of Salmon

The cost of step 1 depends on the algorithm used to generate the Pareto fronts. In all cases, the n biobjective subproblems are solved independently, so the cost of step 1 grows linearly with n. For one subproblem, that is, to generate one Pareto front, we have that

- The cost of Arborescent depends on the number of nodes explored in the branch and bound. In the worst case, it is of the same order as the bruteforce algorithm, and requires to solve $\binom{r}{k}$ NNLS subproblems, while, in the best case, it is of the order of r (Nadisic et al. 2020). Empirically, the cost is far from the worst case but grows faster than linear with r.
- The cost of NNOMP is in $\mathcal{O}(mr^4)$ operations (Yaghoobi et al. 2015).
- The cost of the homotopy algorithm is of the same order as an active-set algorithm and requires at least $\mathcal{O}(r^4)$ operations, see Appendix 1.

Given C, the selection step consists in building $\mathcal{G}$ in $\mathcal{O}(rn)$ operations, then iterating q times (in the worst case) to select a solution. As we maintain updated a sorted list to avoid recomputing the maximum at each iteration, this is done in $\mathcal{O}(q \log(n))$ operations. Since q is

in the order of rn in the worst case, the cost of the selection step is dominated by the cost of the Pareto-front-generating step.

Algorithm 2: The greedy algorithm NNOMP adapted for matrix-wise q-sparse MNNLS.

Input: $B \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times r}$, $q \in \mathbb{N}$
Output: X, approximate solution to Problem (7)
1 $X \leftarrow 0^{r \times n}$; $S \leftarrow \emptyset$; $R_S \leftarrow B$
2 **while** $|S| < q$ *and* $\max_{(i,j)}(A^\top R_S)_{i,j} > 0$ **do**
3 $\quad (i^*, j^*) \leftarrow \operatorname{argmax}_{(i,j) \notin S} A^\top R_S$
4 $\quad S \leftarrow S \cup (i^*, j^*)$
5 $\quad X \leftarrow \min_X \|A_S - B_S X_S\|_F^2$
6 $\quad R_S \leftarrow B - AX$
7 $\quad S \leftarrow \{(i,j) : X(i,j) > 0\}$

3.2 Adapting NNOMP for matrix-wise q-sparse MNNLS

Let us now adapt NNOMP for the matrix-wise q-sparse MNNLS problem. To avoid the overcost of solving the vectorized Problem (8) directly with NNOMP, we can adapt NNOMP to handle the matrix form of Problem (7). We detail the adaptation in Algorithm 2; note that it is not an original contribution and seems to be common knowledge in the sparse approximation community. The solution produced by this algorithm is strictly equivalent to the one produced by direct solving of (8). Our goal in introducing it is to show how using NNOMP within the two-step algorithm Salmon is advantageous compared to using NNOMP directly on Problem (7). On line 1, we initialize X as a zero matrix, S as an empty support, and the residual R_S as equal to B. Then, we select greedily entries to add to the support, while the cardinality of the support is lower than q and the residual greater than zero (line 2). The greedy selection on line 3 consists in choosing the entry that maximizes the decrease of the residual error; this entry is added to the support on line 4. Then, we update X on line 5 (this is an NNLS problem, that we solve with an active-set algorithm), and the residual R_S on line 6; note that in practice we only need to update the j^*-th column of X and R_S. On line 7 we perform a support compression, that is, we restrain the support to the non-zero entries of X. This last step is necessary because of the nonnegativity constraint, that may put to zero an entry that was selected at a previous iteration.

Other greedy algorithms could be adapted similarly, but here we focus only on NNOMP for conciseness. Also, our goal is to study how the original NNOMP compares to NNOMP used within the two-step approach Salmon, rather than comparing different greedy algorithms with each other. The homotopy algorithm may also be similarly adaptable, but this is not trivial and out of the scope of this article.

4 Experiments

In this section, we study the performance of the proposed algorithm Salmon on the unmixing of 7 datasets: 3 faces datasets and 4 hyperspectral images. Then, we study the evolution of the computing time of the algorithm when the sparsity parameter q varies. Finally, we

Table 1 Summary of the datasets, for which $B \in \mathbb{R}^{m \times n}$ and $A \in \mathbb{R}^{m \times r}$

Dataset	Type	m	n	r
CBCL	Faces	2429	$19 \times 19 = 361$	49
Frey	Faces	1965	$20 \times 28 = 560$	36
Kuls	Faces	20	$64 \times 64 = 4096$	5
Jasper	Hyperspectral	198	$100 \times 100 = 10000$	4
Samson	Hyperspectral	156	$95 \times 95 = 9025$	3
Urban	Hyperspectral	162	$307 \times 307 = 94249$	6
Cuprite	Hyperspectral	188	$250 \times 191 = 47750$	12

test on synthetic datasets the ability of Salmon to recover the underlying solution in the presence of noise, when the sparsity varies between columns, with both well-conditioned and ill-conditioned data.

4.1 Data

In the faces datasets, each column of B corresponds to a pixel and each row to an image (that is, $B(i, j)$ is the intensity of pixel j in image i). It is well-known that NMF will extract facial features as the rows of matrix X (Lee and Seung 1999). As no groundtruth is available, we first compute A with SNPA (Gillis 2014), an algorithm for separable NMF, setting the factorization rank r as in the literature. We then compute X with our sparsity-enhancing method. Imposing sparsity on X means that we require that only a few pixels are contained in each facial feature (Hoyer 2004). We consider the 3 widely used face datasets CBCL[2], Frey[3], and Kuls[4].

Similarly a hyperspectral image is an image-by-pixel matrix where each image corresponds to a different wavelength. The columns of A represent the spectral signature of the pure materials (also called endmembers) present in the image (Bioucas-Dias et al. 2012), and we use the ground truth A from Zhu (2017). We compute X, whose columns represent the abundance of materials in each pixel. It makes sense to impose X to be sparse as most pixels contain only a few endmembers (Ma et al. 2013). We consider the 4 widely used datasets[5] Jasper, Samson, Urban, and Cuprite. The characteristics of these datasets are summarized in Table 1.

4.2 Methods

All methods have been implemented in Julia and run on a computer with a processor Intel Core i5-2520M @2.50GHz. The code and experiment scripts are provided in an online repository.[6]

We compare the following methods:

[2] Downloaded from http://poggio-lab.mit.edu/codedatasets.
[3] Downloaded from https://cs.nyu.edu/~roweis/data.html.
[4] Downloaded from http://www.robots.ox.ac.uk/.
[5] Downloaded from http://lesun.weebly.com/hyperspectral-data-set.html.
[6] https://gitlab.com/nnadisic/giant.jl.

- AS denotes an active-set algorithm that solves exactly the NNLS problem without sparsity constraint. It serves as a baseline to compare with the sparsity-constrained methods.
- ℓ_1-CD denotes a coordinate descent algorithm with an ℓ_1 penalty. The penalty parameter λ is fixed and the same for the whole matrix, as in Kim and Park (2007). It has been tuned manually to reach the target sparsity. We compute the unbiased solution by running an active-set NNLS algorithm restrained to the non-zero elements of the ℓ_1-penalized solution.
- Hcw denotes the homotopy algorithm described in Sect. 2.3, that solves the column-wise k-sparse problem, as defined in (6). For each column, we generate the regularization path, take the best k-sparse solution, and unbias it as described above.
- H+S corresponds to the two-step algorithm Salmon using the homotopy algorithm in step 1 (solutions are unbiased as above).
- OGcw stands for orthogonal greedy and denotes the NNOMP algorithm described in Sect. 2.2, that solves the column-wise k-sparse problem.
- OGg denotes the matrix-wise variant of NNOMP described in Sect. 3.2.
- OG+S corresponds to the two-step algorithm Salmon using NNOMP in step 1. We generate the whole Pareto fronts, hence the column-wise sparsity target for NNOMP is $k = r$.
- ARBOcw denotes the branch-and-bound algorithm Arborescent described in Sect. 2.1, that solves the column-wise k-sparse problem.
- ARBO+S corresponds to the two-step algorithm Salmon using Arborescent in step 1. We generate the whole Pareto fronts, hence the column-wise sparsity target for Arborescent is $k = 1$.

4.3 Experiment 1: hyperspectral unmixing

In this experiment, we compare the performance of different variants of Salmon with the corresponding column-wise algorithms and with OGg. For each dataset, we choose the parameter k by trial-and-error. Unless stated otherwise, we define the sparsity parameter of matrix-wise methods as $q = k \times n$, which is equivalent to an average column-wise k-sparsity constraint.

For every dataset, we run the nine methods, and measure the average column-sparsity of the given solutions (defined as the number of non-zero entries divided by the number of columns), the relative reconstruction error $\frac{\|B-AX\|_F}{\|B\|_F}$, and the computing time, for which we measure the median over 10 runs. We set a timeout of 6000 seconds. Note that, for a given dataset and with the same parameters, a given algorithm always gives the same output. For the 3 methods based on NNOMP, we normalize the columns of the matrix A before the computation.

The results of the experiments are summarized in Table 2.

We first note the natural sparsity of the data: without sparsity constraint, AS already produces very sparse solutions, and column-wise methods produce solutions with an average sparsity below the sparsity target k, meaning that some columns are naturally sparser than k. We observe that the column-wise methods give relatively bad results in terms of reconstruction error, while the variants of Salmon are able to enforce sparsity while limiting the loss in error. H+S is only slightly slower than Hcw, meaning that the selection (step 2) takes less time than the homotopy (step 1). On the other hand, OG+S and ARBO+S are slower than their column-wise counterparts because they need to be run with a different

Table 2 Results of the experiments, for the unmixing of facial and hyperspectral datasets

		AS	ℓ_1-CD	Hcw	**H+S**	OGcw	OGg	**OG+S**	ARBOcw	**ARBO+S**
CBCL	Time	0.2	0.1	0.71	0.81	0.08	0.31	3.7	Timeout	Timeout
$r = 49$	Error	12.04	17.37	16.19	13.22*	13.12	12.35	**12.3***	–	–
$k = 3$	Sparsity	6.64	3	2.69	3	2.37	3	3	–	–
Frey	Time	0.22	0.08	1.12	1.27	0.18	0.61	3.97	Timeout	Timeout
$r = 36$	Error	19.35	21.76	23.22	20.75*	21.35	19.86	**19.8**	–	–
$k = 6$	Sparsity	12.29	6	5.52	6	4.64	6	6	–	–
Kuls	Time	0.17	0.12	0.18	0.17	0.28	1.82	0.5	0.67	1.41
$r = 5$	Error	19.05	19.61	20.13	19.12*	19.46	**19.06**	**19.06***	19.42	**19.06***
$k = 3$	Sparsity	3.45	3	2.86	2.99	2.7	3	3	2.76	3
Jasper	Time	0.34	0.22	0.38	0.48	0.39	6.08	1.12	1.21	1.93
$r = 4$	Error	5.71	**5.72**	6.99	**5.72***	7.49	5.76	5.73	6.18	**5.71***
$k = 2$	Sparsity	2.27	2	1.78	1.99	1.72	2	2	1.78	2
Jasper	Time	–	0.18	–	0.44	–	5.26	1.15	–	1.7
$r = 4$	Error	–	7.87	–	5.95*	–	6.06	**5.77***	–	**5.74***
$q/n = 1.8$	Sparsity	–	1.8	–	1.79	–	1.8	1.8	–	1.8
Samson	Time	0.22	0.24	0.2	0.26	0.31	3.67	0.57	0.52	0.8
$r = 3$	Error	3.3	**3.3**	3.34	**3.3***	6.76	3.32	**3.3***	3.4	**3.3***
$k = 2$	Sparsity	2.2	2	1.85	2	1.6	1.99	1.99	1.83	2
Urban	Time	5.08	4.31	4.86	7.79	3.38	958	16.4	33.5	73.1
$r = 6$	Error	7.67	8.13	8.62	7.83*	8.97	8.07	**7.76***	8.27	**7.71***
$k = 2$	Sparsity	2.63	2	1.9	2	1.7	2	2	1.83	2
Cuprite	Time	5.19	3.32	7.86	10.1	5.06	620	31.5	784	4829
$r = 12$	Error	1.74	3.17	2.37	2.01	2.32	1.97	**1.89***	1.93	**1.83***
$k = 4$	Sparsity	6.61	4	3.92	4	3.53	4	4	3.81	4

Time is in seconds, relative error in percent, and sparsity is the average number of non-zero entries per column. Numbers in bold represent, for a given setting, the error of ARBO+S and the best error among the other sparse methods. For the variants of Salmon, a star * indicates that the greedy selection (step 2 of Salmon) is optimal (which can be checked easily: it requires $\delta = q - \sum_k k_j$ at the last iteration). Jasper is processed once with all algorithms for $k = q/n = 2$, and once with matrix-wise algorithms for $q/n = 1.8$ (which is not possible with column-wise algorithms)

sparsity target, respectively $k = r$ and $k = 1$, to compute the whole Pareto front. The computing times of H+S and OG+S seem proportional, and differ by a factor between 2 and 3, which is consistent with our discussion about computational cost in Sect. 3.1. ℓ_1-CD is very fast, and produces good solutions with some datasets (Jasper, Samson), but it is outperformed by Salmon in all cases and sometimes produces solutions with high error (CBCL, Cuprite).

Comparing OGg and OG+S, we observe that OGg is faster for tall matrices, while OG+S is faster for short and fat matrices. Also, OG+S is always better in terms of reconstruction error, meaning that performing the heuristic column-wise and then recombining solutions is more efficient than applying the same heuristic matrix-wise. As regards Arborescent, we see a clear improvement of reconstruction error with Salmon, at the cost of a larger computation time. The two-step approach of Salmon allows Arborescent to be applied for matrix-wise q-sparse MNNLS, which would be impossible otherwise. Another

(a) AS (no sparsity constraint)

(b) ℓ_1-CD

(c) Hcw

(d) H+S

Fig. 5 Abundance maps (that is, reshaped rows of X) from the unmixing of the faces dataset Kuls by different algorithms

advantage of the matrix-wise formulation is the capacity to tune the sparsity parameter more finely; this is illustrated on Jasper with $q/n = 1.8$, for which we reach an average sparsity similar to that of column-wise metods and still get lower errors. For the three variants of Salmon, the stars indicates that the selection (step 2) is done optimally as discussed in Theorem 1; here it is the case for 19 out of 22 settings.

The abundance maps corresponding to the faces dataset Kuls are shown in Fig. 5; the other abundance maps from our experiments are available in Appendix 2.

The features extracted in Fig. 5 correspond to different directions of light. Without sparsity constraint, the quality of the images is good, but the features are not well separated. Column-wise methods and ℓ_1-CD fail to retrieve the features and produce noisier images with pixelated regions. Salmon, using any of the 3 possible methods for step 1, produces better-separated features, with a better spatial coherence.

(a) OGcw

(b) OGg

(c) OG+S

(d) ARBOcw

(e) ARBO+S

Fig. 5 (continued)

4.4 Experiment 2: evolution of the computing time when q varies

In this experiment, we study the impact of the sparsity parameter q on the running time of the matrix-wise sparse MNNLS algorithms. For each setting, we run each algorithm 10 times and keep the minimum running time. All algorithms are deterministic, so for a given setting the number of operations does not vary from one run to another and the differences in running time are due to the operating system, therefore taking the minimum time among several runs is a robust measure. We also show the running time of the non-sparse method AS as a baseline.

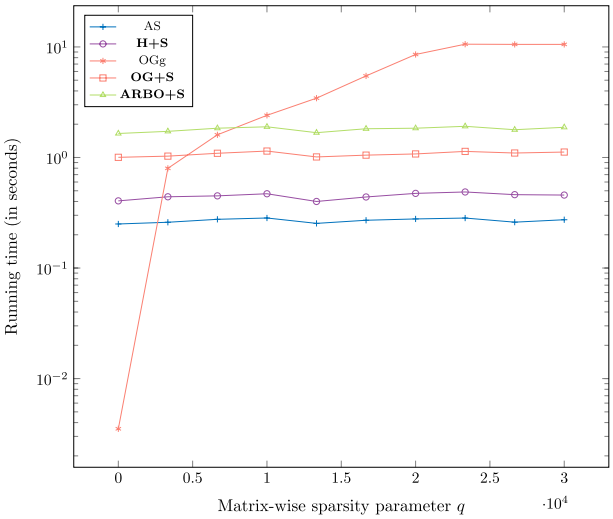


Fig. 6 Evolution of the computing time of matrix-wise sparse NNLS algorithms, when q varies, for the unmixing of the hyperspectral image Jasper

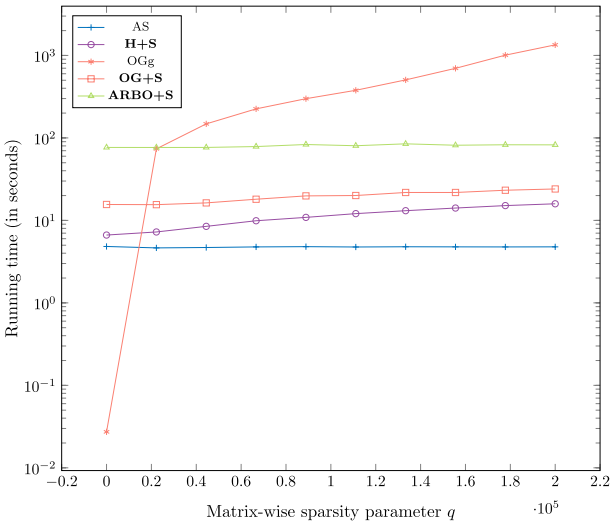


Fig. 7 Evolution of the computing time of matrix-wise sparse NNLS algorithms, when q varies, for the unmixing of the hyperspectral image Urban

We consider the unmixing of the hyperspectral images Jasper (Fig. 6) and Urban (Fig. 7) for different values of q. On Jasper, the variants of Salmon have an almost constant running time, meaning that the cost of the selection step (step 2) is negligible compared to the cost of generating the Pareto fronts (step 1). On the other hand, the running time of OGg grows exponentially (note the log scale of the vertical axis), although it is faster than Salmon when q is small. On Urban, the results are very similar. The behaviour of ARBO+S and OGg is the same. For H+S, and to a lesser extent OG+S, the computing time slightly grows as q grows, meaning that the cost of the selection step is not negligible, but it is still dominated by the cost of step 1.

In this experiment, we find similar results on Jasper, a small image with $r = 4$ where the column-wise subproblems of step 1 are very small, and Urban, a large image with $r = 12$

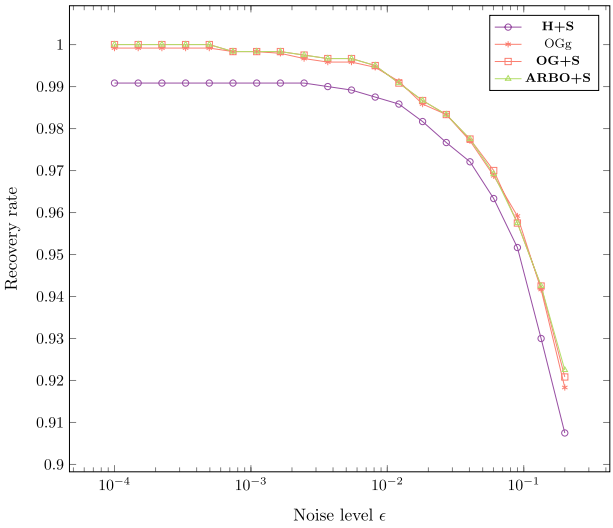


Fig. 8 Evolution of the proportion of entries correctly recovered by matrix-wise sparse NNLS algorithms, on a synthetic well-conditioned dataset, when the noise level varies. The rate plotted is the average over 10 runs

where these subproblems are quite large. This means that our selection algorithm is very fast and that the parameter q does not increase significantly the computing time of Salmon.

4.5 Experiment 3: recovery of underlying solution on synthetic datasets

In this experiment, we study the ability of the matrix-wise sparse MNNLS algorithms to recover the true solution in synthetic data sets where X is generated with columns of different sparsities. First, we generate $A \in \mathbb{R}^{100\times 6}$ following the uniform distribution in [0, 1]. If we want A to be ill-conditioned, we then compute the SVD of $A = U\Sigma V^{\mathsf{T}}$, replace the diagonal entries of Σ by values between 10^{-4} and 1 equally spaced in a log scale, and reconstruct $A = U\Sigma V^{\mathsf{T}}$. Next, we generate $X \in \mathbb{R}^{6\times 200}$ such that all columns are k-sparse with $k \in \{2, 3, 4\}$ chosen uniformly at random, while the nonzero entries are generated uniformly at random in the interval [0,1]. We then compute $B = AX$. For the noise to be added to B, we first generate a matrix N in which each entry is drawn from the normal distribution of mean 0 and variance 1, then rescale $N \leftarrow \epsilon \frac{N}{\|N\|_2}\|B\|_2$ so that $\|N\|_2 = \epsilon\|B\|_2$, where ϵ is the noise level.

We generate this way 10 well-conditioned datasets and 10 ill-conditioned, and for different values of ϵ we generate and add noise to B. We then try to compute X with different algorithms, given A and the noisy B, and we measure the recovery rate, defined as the proportion of entries of the computed X having the same value (in the sense zero or non-zero) than the corresponding entry of the generated X. For each setting, we then average the recovery rate over the 10 datasets. Figures 8 and 9 show the results for the well-conditioned and the ill-conditioned data, respectively.

Surprisingly, for the well-conditioned dataset, all matrix-wise algorithms perform similarly, expect H+S that performs slightly worst. When the noise level is below $10^{-2} = 1\%$, they almost perfectly recover the supports of the columns of X. For higher noise levels, the recovery rate drops rapidly.

For the ill-conditioned dataset, ARBO+S (the variant of Salmon using the exact algorithm Arborescent in step 1) has a recovery rate close to 100% for noise smaller than 0.1%,

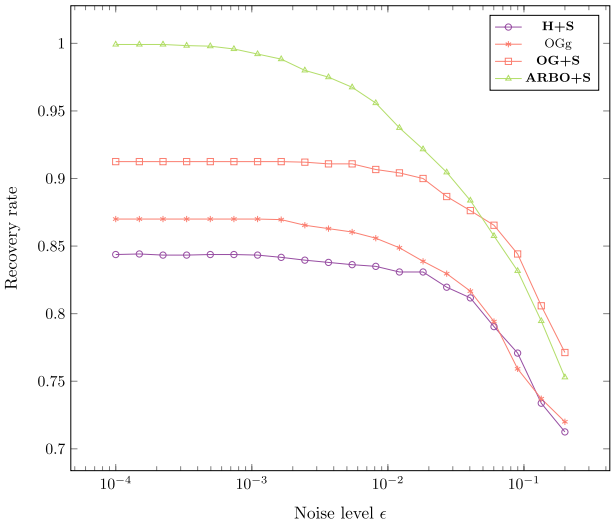


Fig. 9 Evolution of the proportion of entries correctly recovered by matrix-wise sparse NNLS algorithms, on a synthetic ill-conditioned dataset, when the noise level varies. The rate plotted is the average over 10 runs

while other variants do not perform as well. The recovery rate of ARBO+S drops for noise above 0.1% but it still performs better than the other variants of Salmon, until 1% noise where OG+S becomes competitive.

As a conclusion, we see that using Arborescent in step 1 is especially effective with ill-conditioned data, which is often the case in real-world settings. OG+S is also a good solution when the noise level is higher. Using Arborescent is not very interesting when data is well-conditioned.

What algorithm should one use for step 1 of Salmon? When the dimension r is small or when the computing time is not critical, the best option is to use Arborescent in step 1 as it is the only algorithm to compute the Pareto fronts exactly. It is also the only algorithm to properly handle ill-conditioned data. In other cases, using NNOMP in step 1 generally produces better solutions than using the homotopy algorithm. The homotopy algorithm is the fastest so it is appropriate when processing very large datasets with a limited time. Note that there exist many other sparse NNLS algorithms that could be adapted to perform step 1, see for example Mohimani et al. (2007) or Blumensath and Davies (2009). We do not detail them for the sake of conciseness, and because our contribution lies not in the column-wise sparse methods but rather in how to use them in the two-step algorithm Salmon.

5 Conclusion

In this paper, we focused on the multiple nonnegative least squares problem with a matrix-wise ℓ_0 constraint. We introduced Salmon (Algorithm 1), that first computes for each column a Pareto front (step 1) and then applies a provably near-optimal selection strategy to build a solution matrix X (step 2). We computed the Pareto fronts with three existing algorithms: one exact but slow branch-and-bound algorithm, and two fast heuristics. We illustrated the advantages of Salmon for the unmixing of real-world facial and hyperspectral images, for which it outperformed state-of-the-art methods. We also compared the different variants of Salmon on real-world and synthetic data sets to highlight their advantages and drawbacks.

Appendix 1: The homotopy algorithm

In this section, we detailed the homotopy algorithm mentioned in Sect. 2.3.

Given an ℓ_1-NNLS problem, the homotopy algorithm computes sequentially all optimal solutions for the different values of λ. In a nutshell, it uses the KKT conditions (necessary conditions for optimality) to first find the value $\lambda_{\max}$ for which the optimal solution of ℓ_1 -NNLS is $x = 0$ for any $\lambda \geq \lambda_{\max}$, and then to compute the next smaller values of λ for which the support (that is, the set of non-zero entries) of the optimal solution changes (one zero entry becomes non-zero, or the other way around). This is similar in spirit to active-set methods, akin to the simplex algorithm for linear programming. These values of λ are called *breakpoints*, between which the support of the optimal solution does not change. We denote the first breakpoint $\lambda_{\max}$, and the following ones $\lambda_2, \lambda_3, \cdots$; see Figs. 3 and 4 for an illustration.

Appendix 1.1: Optimality conditions

The homotopy algorithm uses the first-order optimality conditions, that is, the KKT conditions, to determine the breakpoints and the supports of the corresponding solutions. Because x is nonnegative, we have $\|x\|_1 = \sum_i x_i = e^T x$, where e is the vector of all ones whose dimension will be clear from context. Therefore, the ℓ_1-NNLS problem can be written as follows

$$\min_{x \geq 0} f(x), \text{ where } f(x) = \frac{1}{2}\|Ax - b\|_2^2 + \lambda e^T x. \tag{12}$$

The KKT conditions are $x \geq 0$,

$$\nabla f(x) = \underbrace{A^\top A}_{P} x - \underbrace{A^\top b}_{\ell} + \lambda e \leq 0, \tag{13}$$

$$x_i (A^\top A x - A^\top b + \lambda e)_i = 0 \text{ for all } i. \tag{14}$$

Equation 14 is the complementary condition; for every entry of x, either the entry itself or the corresponding gradient entry is equal to zero. To simplify the notation, let us define $P = A^\top A$ and $\ell = A^\top b$.

As f is a convex function and the feasible set contains a Slater point (e.g., $x = e$), the KKT conditions are necessary and sufficient. Therefore, any solution x satisfying them is optimal. Suppose we know the optimal support, that is, the set $\mathcal{K}$ such that $x(\mathcal{K}) > 0$ and $x(\bar{\mathcal{K}}) = 0$, where $\bar{\mathcal{K}} = \{1, 2, \ldots, n\} \setminus \mathcal{K}$. Then

$$\begin{aligned} x(\mathcal{K}) > 0 &\Rightarrow P(\mathcal{K}, \mathcal{K}) x(\mathcal{K}) - \ell(\mathcal{K}) + \lambda e = 0 \\ &\Rightarrow x(\mathcal{K}) = P(\mathcal{K}, \mathcal{K})^{-1} (\ell(\mathcal{K}) - \lambda e) \geq 0, \end{aligned} \tag{$\mathcal{C}1$}$$

and

$$Px - \ell + \lambda e \geq 0 \Rightarrow P(\bar{\mathcal{K}}, \mathcal{K}) x(\mathcal{K}) - \ell(\bar{\mathcal{K}}) + \lambda e \geq 0. \tag{$\mathcal{C}2$}$$

Replacing $x(\mathcal{K})$ in ($\mathcal{C}2$) by ($\mathcal{C}1$), we have

$$P(\bar{\mathcal{K}}, \mathcal{K})[P(\mathcal{K}, \mathcal{K})^{-1}(\ell(\mathcal{K}) - \lambda e)] - \ell(\bar{\mathcal{K}}) + \lambda e \geq 0.$$

Let us simplify the notation. Let

$$a_{\mathcal{K}} = P(\mathcal{K}, \mathcal{K})^{-1}\ell(\mathcal{K}), \text{ and } b_{\mathcal{K}} = P(\mathcal{K}, \mathcal{K})^{-1}e. \tag{15}$$

We can rewrite ($\mathcal{C}1$) as

$$a_{\mathcal{K}} - \lambda b_{\mathcal{K}} \geq 0. \tag{$\mathcal{C}1b$}$$

Note that the dimension of $a_{\mathcal{K}}$ and $b_{\mathcal{K}}$ is the cardinality of $\mathcal{K}$. Let

$$c_{\mathcal{K}} = P(\bar{\mathcal{K}}, \mathcal{K})a_{\mathcal{K}} - \ell(\bar{\mathcal{K}}), \text{ and } d_{\mathcal{K}} = P(\bar{\mathcal{K}}, \mathcal{K})b_{\mathcal{K}} - e.$$

We can rewrite ($\mathcal{C}2$) as

$$c_{\mathcal{K}} - \lambda d_{\mathcal{K}} \geq 0. \tag{$\mathcal{C}2b$}$$

Note that the dimension of $c_{\mathcal{K}}$ and $d_{\mathcal{K}}$ is the cardinality of $\bar{\mathcal{K}}$. Moreover, Equations ($\mathcal{C}1b$) and ($\mathcal{C}2b$) are linear in λ: Given $\mathcal{K}$, we can easily compute $a_{\mathcal{K}}, b_{\mathcal{K}}, c_{\mathcal{K}}, d_{\mathcal{K}}$.

Appendix 1.2: Algorithm description

The goal of the homotopy algorithm is to compute breakpoints and their corresponding supports. It starts with an empty support, corresponding to the zero vector for any $\lambda \geq \lambda_{max}$, and iteratively adds or removes entries to the support while decreasing the value of λ.

The first step to build the regularization path is to find the first breakpoint $\lambda_{\max}$, that is, the minimum value of λ such that the solution is the zero vector. If the optimal solution is $x = 0$, that is, $\mathcal{K} = \emptyset$ and $\bar{\mathcal{K}} = \{1, 2, \ldots, n\}$, then from Eq. 13 we have $\lambda \geq \max_i \ell_i$. Therefore, the first breakpoint is

$$\lambda_{\max} = \max_i \ell_i = \max_i (A^\top b)_i = \max_i A(:, i)^T b.$$

The index $i_1 = \arg\max_i \ell_i$ is the first to enter the support[7], thus for $\lambda_2 \leq \lambda < \lambda_{\max}$ we have $\mathcal{K} = \{i_1\}$.

From a given support $\mathcal{K}_j$, the next breakpoint $\lambda_{j+1} \leq \lambda_j$ is the largest value of λ that violates one of the conditions ($\mathcal{C}1$) or ($\mathcal{C}2$). If ($\mathcal{C}1$) is violated then a variable will leave the support, that is, a positive entry will become zero. Denoting k^* the index of this entry, we have $\mathcal{K}_{j+1} = \mathcal{K}_j \setminus \{k^*\}$. If ($\mathcal{C}2$) is violated then a variable will enter the support, that is, a zero entry will become positive, $\mathcal{K}_{j+1} = \mathcal{K}_j \cup \{k^*\}$.

Let us consider ($\mathcal{C}1$). We have $a_{\mathcal{K}}(k) - \lambda b_{\mathcal{K}}(k) \geq 0$ for all k so

$$\lambda_{j+1} \geq \max_{\{k | b_{\mathcal{K}}(k) < 0\}} \frac{a_{\mathcal{K}}(k)}{b_{\mathcal{K}}(k)}.$$

Similarly, for ($\mathcal{C}2$) we have $c_{\mathcal{K}}(k) - \lambda d_{\mathcal{K}}(k) \geq 0$ for all k so

[7] If two of more columns maximize the value of ℓ_i, we have to pick one to start the homotopy. If we always pick the one with smallest index, then the algorithm behaves normally, except that at the next iteration we will have $\lambda_{j+1} = \lambda_j$. This is similar to Bland's rule for the simplex algorithm.

$$\lambda_{j+1} \geq \max_{\{k|d_{\mathcal{K}}(k)<0\}} \frac{c_{\mathcal{K}}(k)}{d_{\mathcal{K}}(k)}.$$

Therefore,

$$\lambda_{j+1} = \max \left(\underbrace{\max_{\{k|b_{\mathcal{K}}(k)<0\}} \frac{a_{\mathcal{K}}(k)}{b_{\mathcal{K}}(k)}}_{\text{Case 1}}, \underbrace{\max_{\{k|d_{\mathcal{K}}(k)<0\}} \frac{c_{\mathcal{K}}(k)}{d_{\mathcal{K}}(k)}}_{\text{Case 2}} \right).$$

The algorithm is detailed formally in Algorithm 3. Note that, inside the algorithm loop, once a support $\mathcal{K}$ is identified, getting the corresponding optimal solution is straightforward. From Eq. 15, if $a_{\mathcal{K}}$ is nonnegative, then the unbiased optimal solution of the NNLS problem is the vector x^* such that $x^*(\bar{\mathcal{K}}) = 0$ and $x^*(\mathcal{K}) = a_{\mathcal{K}}$. If $a_{\mathcal{K}}$ has negative entries, then we can compute the unbiased solution with a standard NNLS solver, with $x^*(\bar{\mathcal{K}}) = 0$ and $x^*(\mathcal{K}) = \arg\min_{x \geq 0} \|P(\mathcal{K}, \mathcal{K})x - l(\mathcal{K})\|_2^2$.

Algorithm 3: Homotopy algorithm for sparse NNLS

Input: $A \in \mathbb{R}^{m \times r}$, $b \in \mathbb{R}^m$
Output: Breakpoints λ_j, corresponding supports $\mathcal{K}_j$ and solutions x_j^*, for all j

$i_1 \leftarrow \arg\max_i \ell_i$
$K \leftarrow \{i_1\}$; $\lambda_1 = \lambda_{\max} = \ell_{i_1}$, $j \leftarrow 1$
while $\lambda_j > 0$ **do**
  $j \leftarrow j + 1$
  Compute a_K, b_K, c_K, d_K
  $\lambda_{C1} \leftarrow \max_{\{k|b_{\mathcal{K}}(k)<0\}} \frac{a_{\mathcal{K}}(k)}{b_{\mathcal{K}}(k)}$, and $k^*_{C1} \leftarrow$ corresponding argmax
  $\lambda_{C2} \leftarrow \max_{\{k|d_{\mathcal{K}}(k)<0\}} \frac{c_{\mathcal{K}}(k)}{d_K(k)}$, and $k^*_{C2} \leftarrow$ corresponding argmax
  if $\lambda_{C1} \geq \lambda_{C2}$ **then**
    $\lambda_j \leftarrow \lambda_{C1}$
    $\mathcal{K} \leftarrow \mathcal{K} \setminus \{k^*_{C1}\}$
  else if $\lambda_{C1} < \lambda_{C2}$ **then**
    $\lambda_j \leftarrow \lambda_{C2}$
    $\mathcal{K} \leftarrow \mathcal{K} \cup \{k^*_{C2}\}$
  $x_j^*(\bar{\mathcal{K}}) \leftarrow 0$
  if $a_{\mathcal{K}} > 0$ **then**
    $x_j^*(\mathcal{K}) \leftarrow a_K$
  else
    $x^*_{\mathcal{K}} = \arg\min_{x \geq 0} \|A(:, \mathcal{K})x - b(\mathcal{K})\|_2^2$

Appendix 1.3: Computational cost

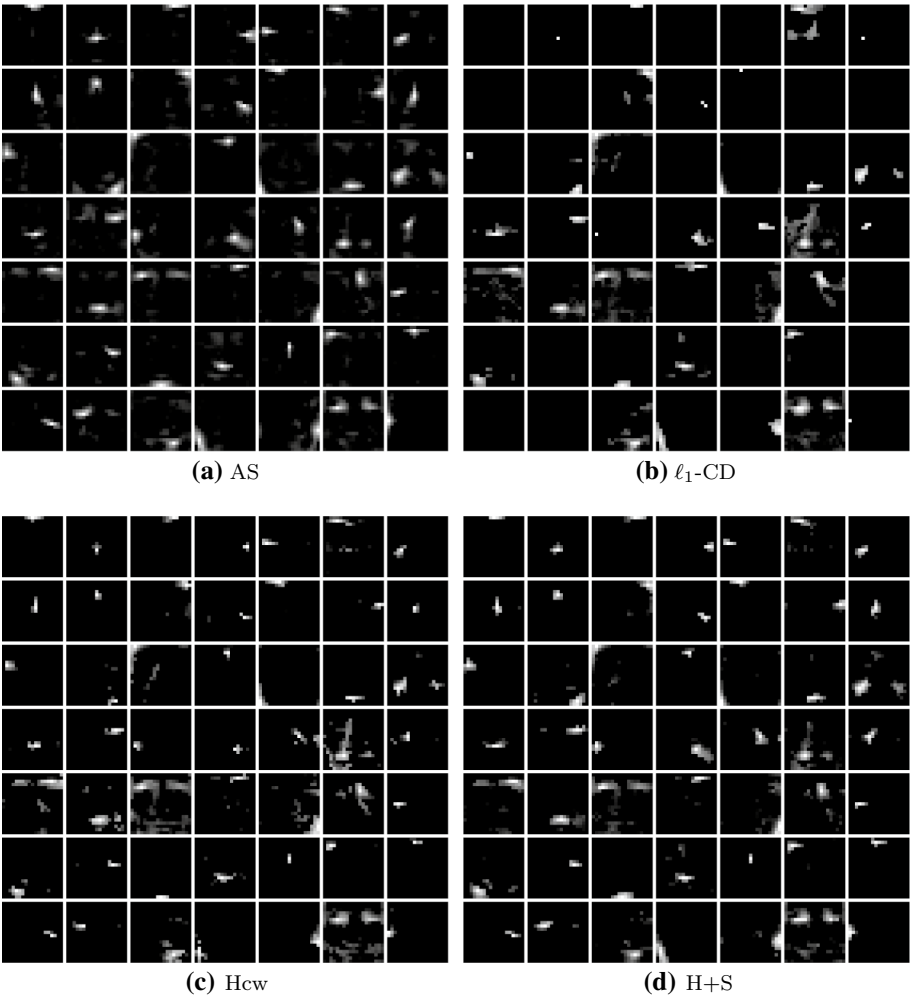
As explained by Kim et al. (2013), the time complexity of one iteration of the homotopy algorithm is the same as one iteration of the standard active-set algorithm (Lawson and Hanson 1995), that is, $\mathcal{O}(r^3)$ operations. It is dominated by the computation of $a_{\mathcal{K}}$, that entails solving a linear system in at most r variables. The number of iterations equals the number of breakpoints, which is in practice similar to those of the active-set. In the worst case, active-set methods might require an exponential number of iterations, up to $\mathcal{O}(2^r)$, as the simplex algorithm for linear programming. However, in practice, we have observed that it typically requires much less iterations, of the order of $\mathcal{O}(r)$. In particular, when P^{-1} is diagonally dominant, we have $b_{\mathcal{K}} > 0$, so $(\mathcal{C}1)$ is never violated. As a result, when λ decreases, no positive entry of x becomes zero. We only add entries to the support, so the homotopy algorithm will be done in at most r iterations. In practice, even when this condition is not met, we have observed that adding entries to the support happens far more often than removing entries.

As the homotopy algorithm solves a series of ℓ_1-penalized NNLS problems, there exist conditions under which it is guaranteed to recover the correct supports, that is, the supports of the solutions of the corresponding ℓ_0-constrained NNLS problems; see Itoh et al. (2017) and the references therein.

Appendix 2: Additional experimental results

In this document, we provide the abundance maps resulting from our experiments that could not be included in the paper:

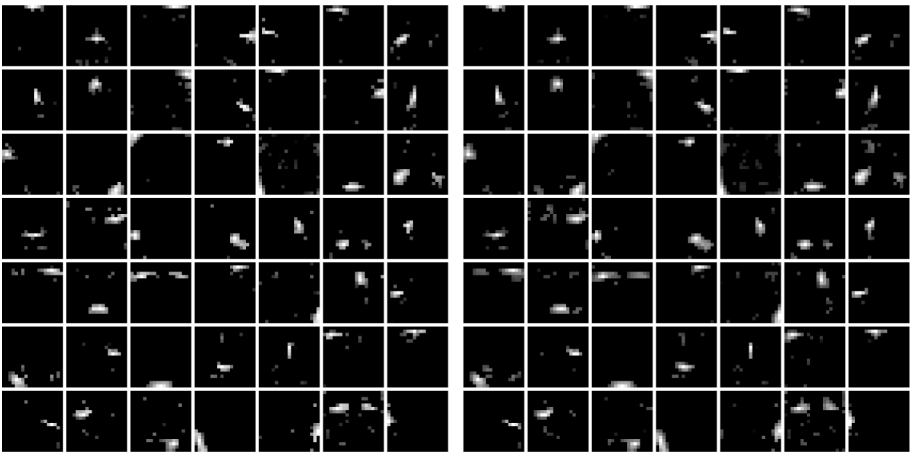
- CBCL in Fig. 10,
- Frey in Fig. 11,
- Kuls in Fig. 12,
- Jasper in Fig. 13 and 14,
- Samson in Fig. 15,
- Urban in Fig. 16, and
- Cuprite in Fig. 17.



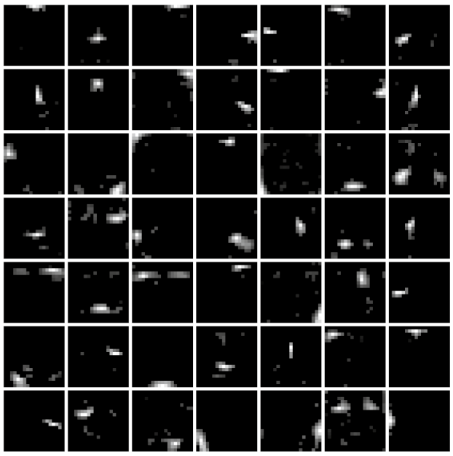
(a) AS **(b)** ℓ_1-CD

(c) Hcw **(d)** H+S

Fig. 10 Abundance maps (that is, reshaped rows of X) from the unmixing of the faces dataset CBCL by different algorithms



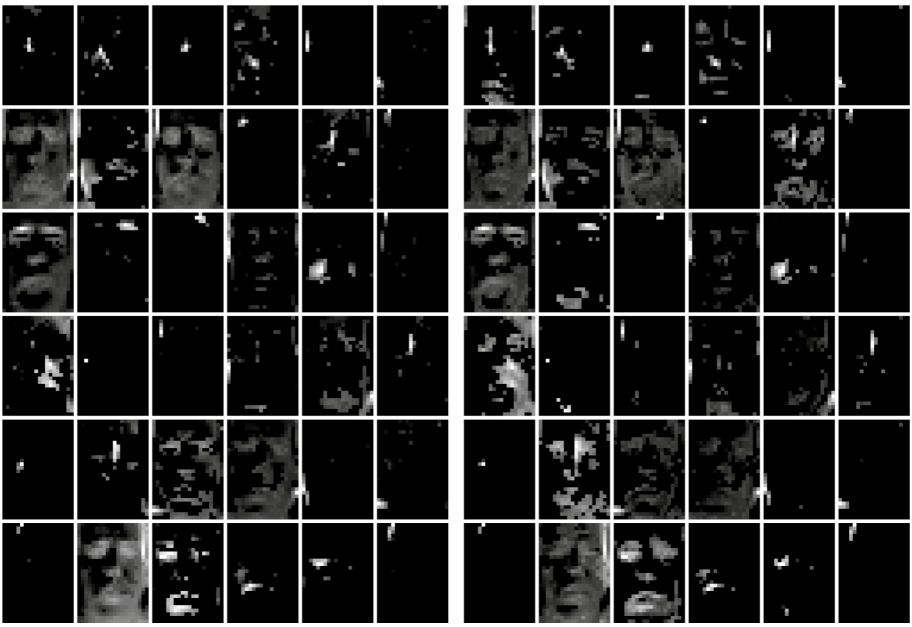
(a) OGcw **(b)** OGg



(c) OG+S

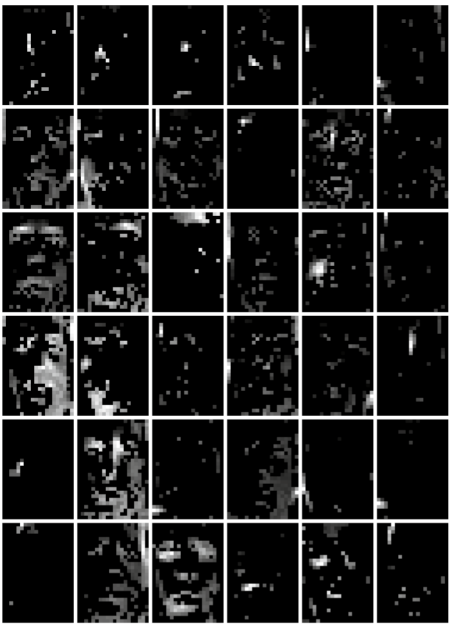
Fig. 10 (continued)

(a) AS **(b)** ℓ_1-CD

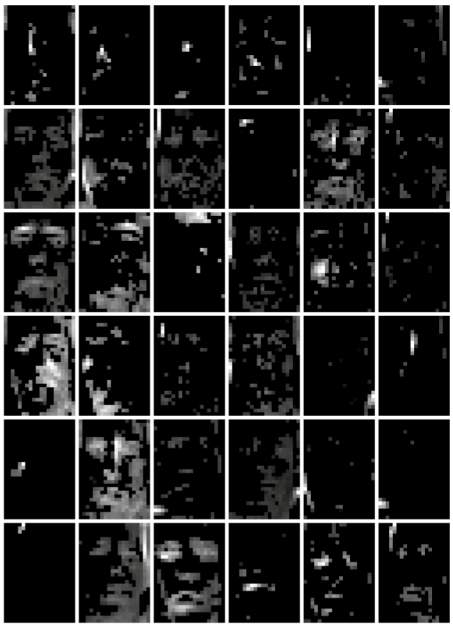


(c) Hcw **(d)** H+S

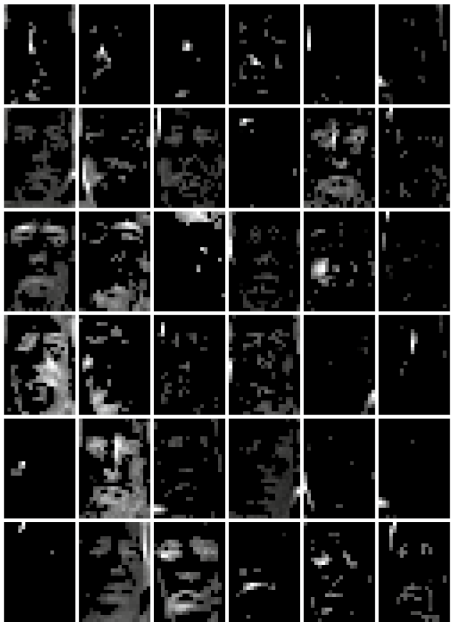
Fig. 11 Abundance maps (that is, reshaped rows of X) from the unmixing of the faces dataset Frey by different algorithms



(a) OGcw



(b) OGg



(c) OG+S

Fig. 11 (continued)

(a) AS

(b) ℓ_1-CD

(c) Hcw

(d) H+S

Fig. 12 Abundance maps (that is, reshaped rows of X) from the unmixing of the faces dataset Kuls by different algorithms

(a) OGcw

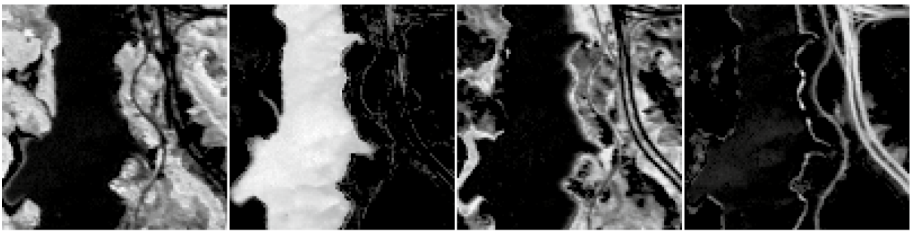
(b) OGg

(c) OG+S

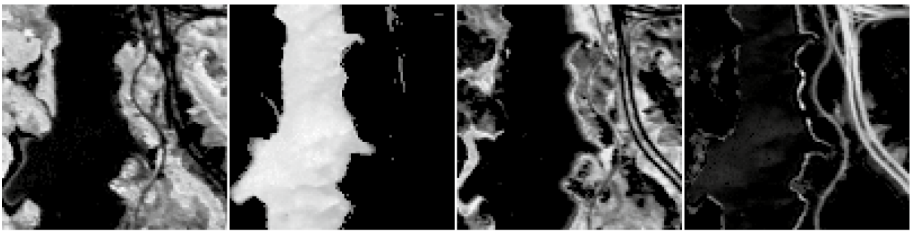
(d) ARBOcw

(e) ARBO+S

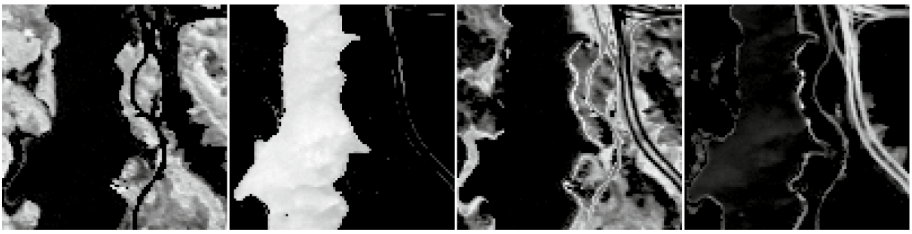
Fig. 12 (continued)



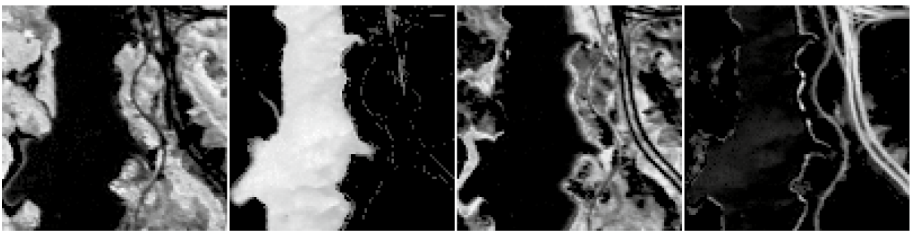
(a) AS



(b) ℓ_1-CD

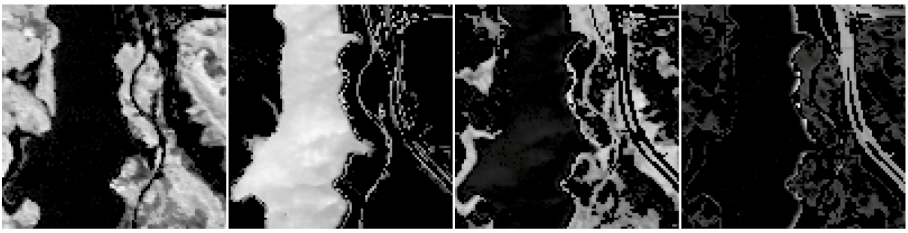


(c) Hcw

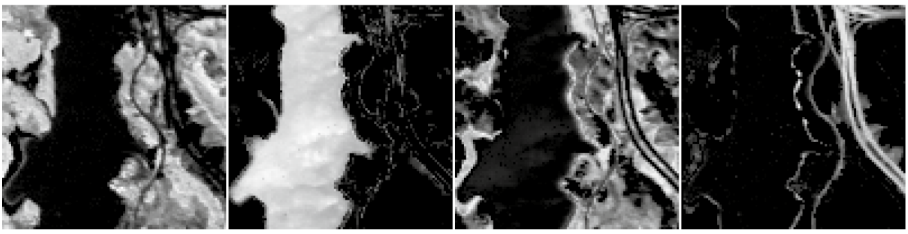


(d) H+S

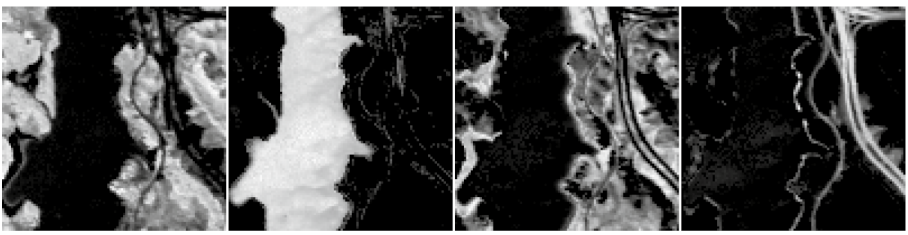
Fig. 13 Abundance maps (that is, reshaped rows of X) from the unmixing of the hyperspectral image Jasper by different algorithms



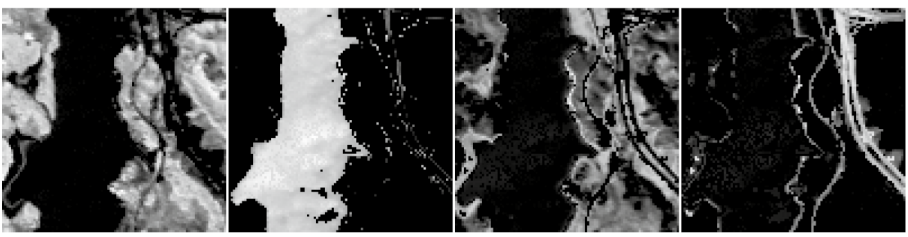
(a) OGcw



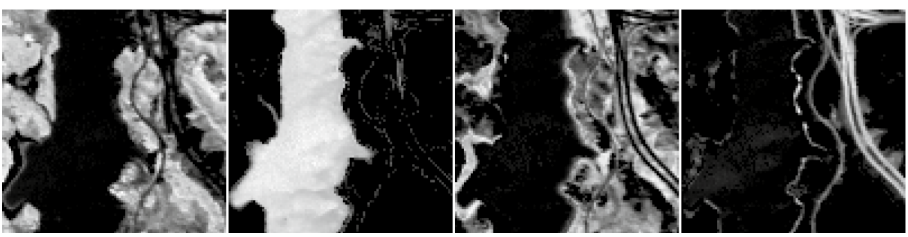
(b) OGg



(c) OG+S

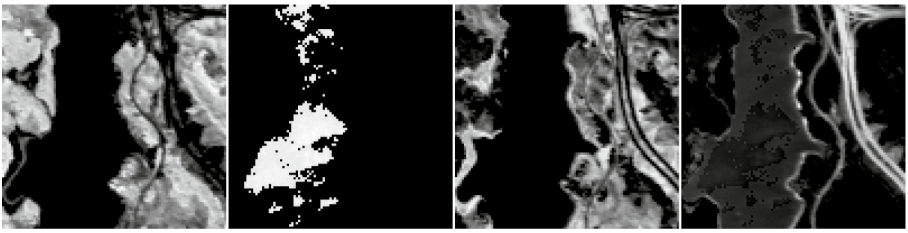


(d) ARBOcw

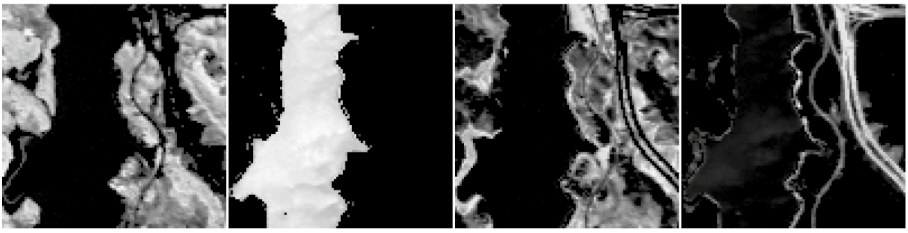


(e) ARBO+S

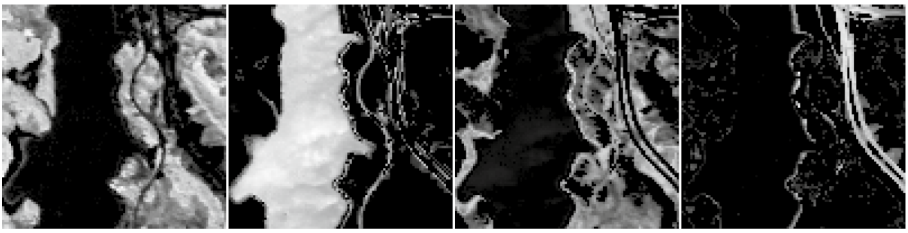
Fig. 13 (continued)



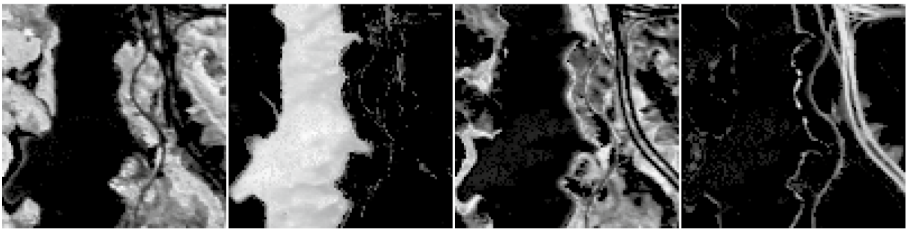
(a) ℓ_1-CD



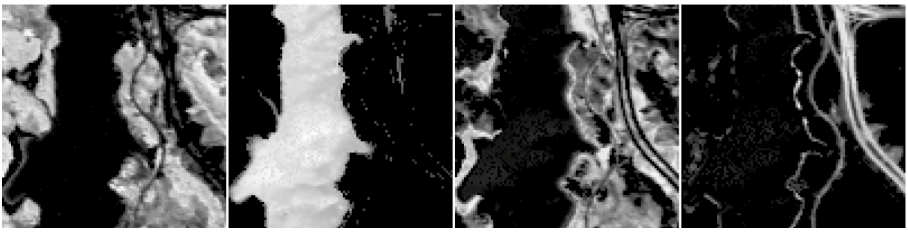
(b) H+S



(c) OGg

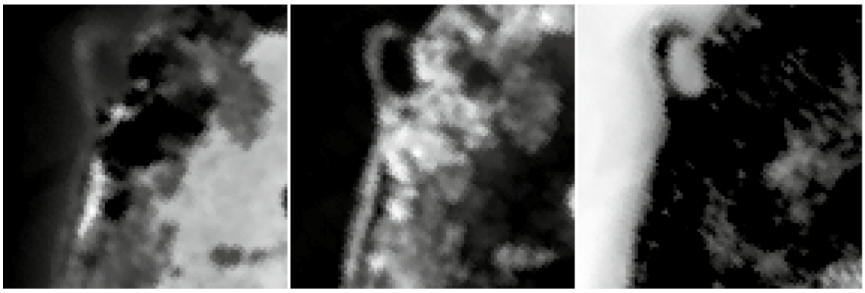


(d) OG+S

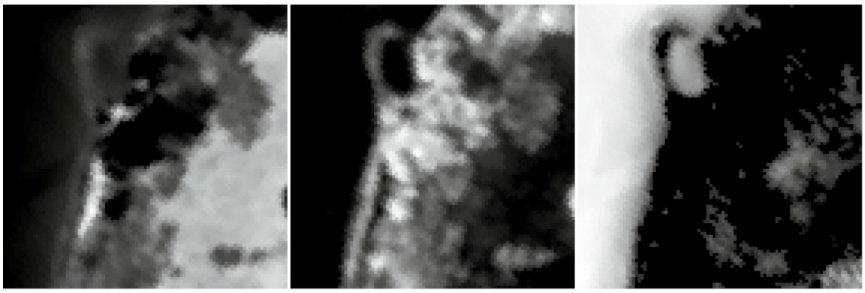


(e) ARBO+S

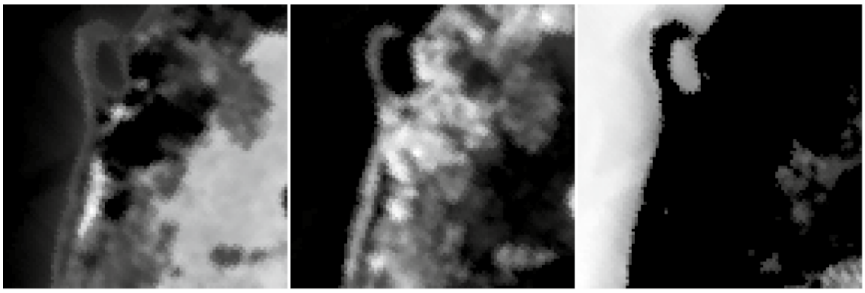
Fig. 14 Abundance maps (that is, reshaped rows of X) from the unmixing of the hyperspectral image Jasper by different algorithms, with $k = q/n = 1.8$



(a) AS



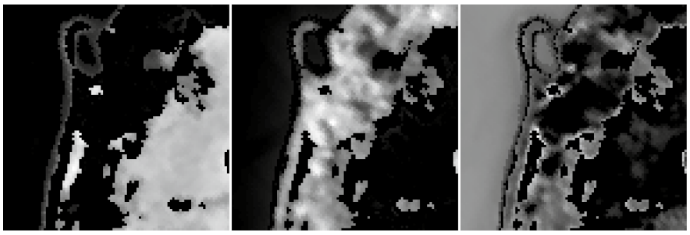
(b) ℓ_1-CD



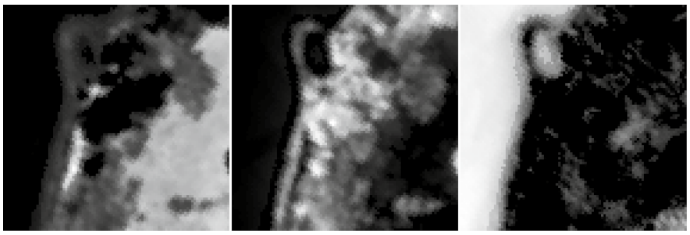
(c) Hcw

(d) H+S

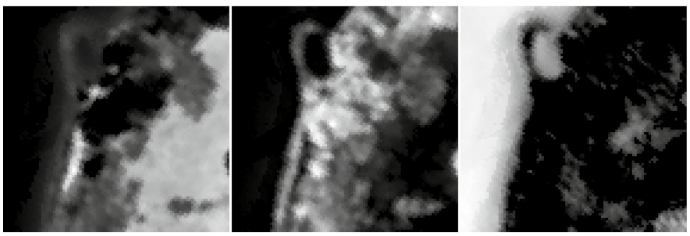
Fig. 15 Abundance maps (that is, reshaped rows of X) from the unmixing of the hyperspectral image Samson by different algorithms



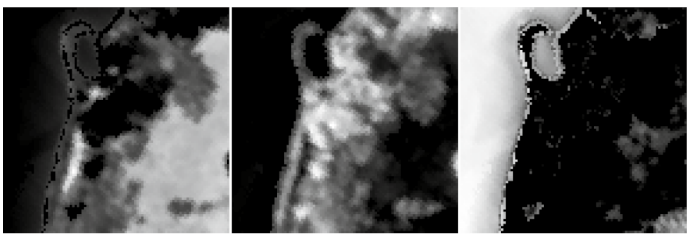
(a) OGcw



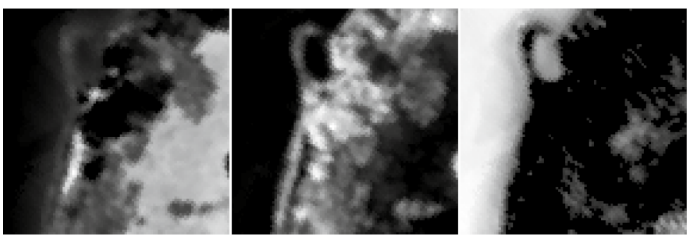
(b) OGg



(c) OG+S



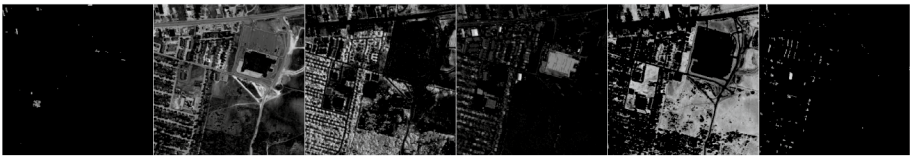
(d) ARBOcw



(e) ARBO+S

Fig. 15 (continued)

(a) AS

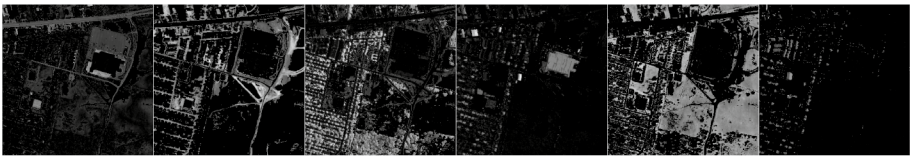


(b) ℓ_1-CD

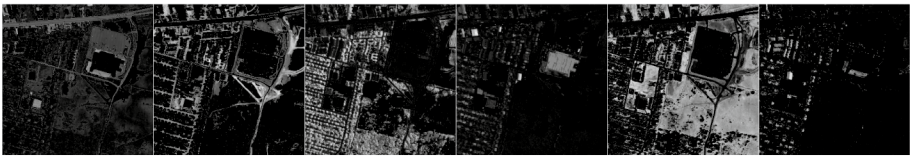
(c) Hcw

(d) H+S

Fig. 16 Abundance maps (that is, reshaped rows of X) from the unmixing of the hyperspectral image Urban by different algorithms

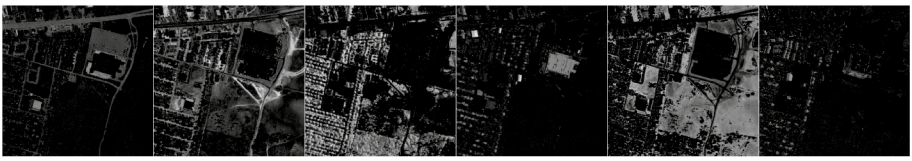


(a) OGcw

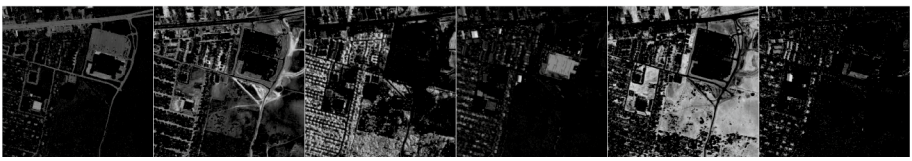


(b) OGg

(c) OG+S

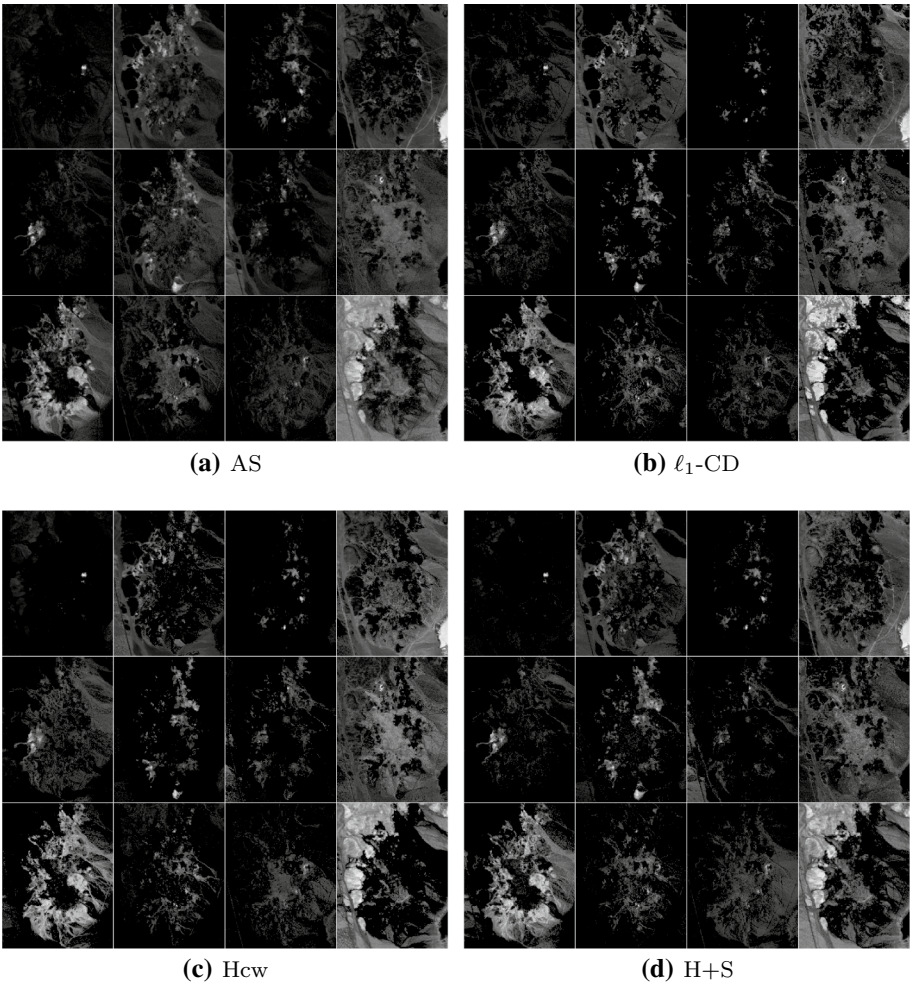


(d) ARBOcw



(e) ARBO+S

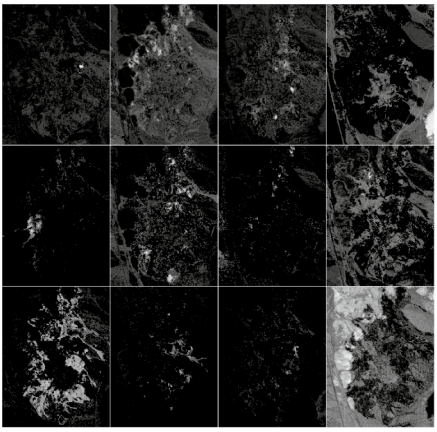
Fig. 16 (continued)



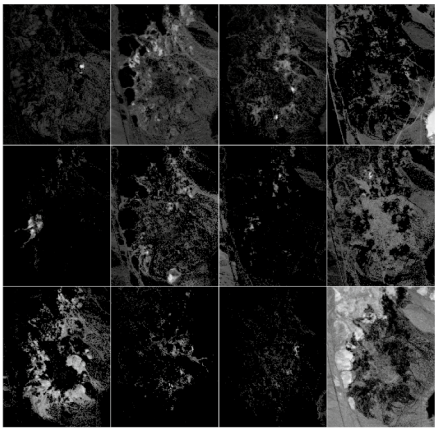
(a) AS **(b)** ℓ_1-CD

(c) Hcw **(d)** H+S

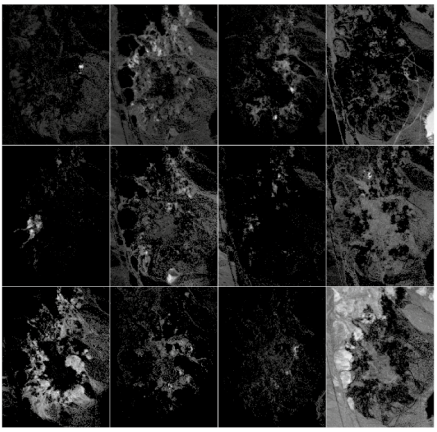
Fig. 17 Abundance maps (that is, reshaped rows of X) from the unmixing of the hyperspectral image Cuprite by different algorithms



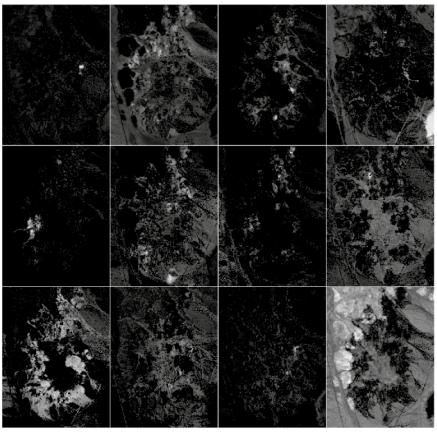
(a) OGcw



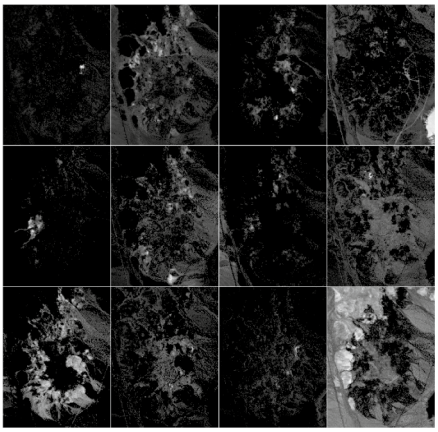
(b) OGg



(c) OG+S



(d) ARBOcw



(e) ARBO+S

Fig. 17 (continued)

Acknowledgements We thank Maxime De Wolf for his help in the implementation of the homotopy algorithm. We thank T.T. Nguyen and her co-authors for making the codes of nonnegative greedy algorithms available online under a free software license. Finally, we thank the reviewers of this paper, whose comments helped significantly to improve the paper.

Author contributions NN designed the algorithm Salmon, implemented it, performed the experiments, and wrote the majority of the article. JEC, AV, and NG all contributed significantly to the design of the algorithm and to the writing. JEC provided expertise in greedy algorithms and helped with the implementation of the matrix-wise variant of NNOMP. AV provided expertise on the homotopy algorithm and helped with its analysis and implementation. NG supervised the work, significantly improved the proof of near-optimality, and revised the manuscript.

Funding NN and NG acknowledge the support by the European Research Council (ERC starting Grant No 679515), and by the Fonds de la Recherche Scientifique - FNRS and the Fonds Wetenschappelijk Onderzoek - Vlanderen (FWO) under EOS project O005318F-RG47. NG also acknowledges the Francqui foundation. JEC acknowledges the support of the ANR Grant ANR JCJC LoRAiA ANR-20-CE23-0010.

Data availability The primary sources of the datasets used in our experiments are indicated in the main document. We also include them along with our code and test scripts in the following online repository https://gitlab.com/nnadisic/giant.jl.

Declarations

Conflict of Interest The authors have no conflict of interest or competing interest to declare.

Ethics approval Not applicable.

References

Aharon, M., Elad, M., & Bruckstein, A. M. (2005). K-SVD and its non-negative variant for dictionary design. In Wavelets XI, Int. Soc. for Optics and Photonics.

Ben Mhenni, R., Bourguignon, S., & Ninin, J. (2021). Global optimization for sparse solution of least squares problems. *Optimization Methods and Software*, 1–30.

Bioucas-Dias, J. M., Plaza, A., Dobigeon, N., Parente, M., Du, Q., Gader, P., & Chanussot, J. (2012). Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 5*(2), 354–379.

Blumensath, T., & Davies, M. E. (2009). Iterative hard thresholding for compressed sensing. *Applied and computational harmonic analysis, 27*(3), 265–274.

Bruckstein, A. M., Elad, M., & Zibulevsky, M. (2008). On the uniqueness of nonnegative sparse solutions to underdetermined systems of equations. *IEEE Transactions on Information Theory, 54*(11), 4813–4820.

Chen, S., Billings, S. A., & Luo, W. (1989). Orthogonal least squares methods and their application to non-linear system identification. *International Journal of control, 50*(5), 1873–1896.

Cichocki, A., Phan, A.H., & Caiafa, C. (2008). Flexible HALS algorithms for sparse non-negative matrix/tensor factorization. In *IEEE workshop on machine learning for signal processing*, (pp. 73–78).

Cohen, J.E., & Gillis, N. (2019). Nonnegative low-rank sparse component analysis. In *2019 IEEE international conference on acoustics, speech and signal processing*, (pp. 8226–8230).

Donoho, D. L., & Tsaig, Y. (2008). Fast solution of ℓ_1-norm minimization problems when the solution may be sparse. *IEEE Transactions on Information theory, 54*(11), 4789–4812.

Efron, B., Hastie, T., Johnstone, I., Tibshirani, R., et al. (2004). Least angle regression. *The Annals of statistics, 32*(2), 407–499.

Eggert, J., & Korner, E. (2004). Sparse coding and NMF. *IEEE International Joint Conference on Neural Networks, 4*, 2529–2533.

Foucart, S., & Koslicki, D. (2014). Sparse recovery by means of nonnegative least squares. *IEEE Signal Processing Letters, 21*(4), 498–502.

Gillis, N. (2012). Sparse and unique nonnegative matrix factorization through data preprocessing. *Journal of Machine Learning Research, 13*, 3349–3386.

Gillis, N. (2014). Successive nonnegative projection algorithm for robust nonnegative blind source separation. *SIAM Journal on Imaging Sciences, 7*(2), 1420–1450.

Gillis, N. (2020). *Nonnegative matrix factorization*. SIAM.

Hoyer, P. O. (2002). Non-negative sparse coding. In *IEEE workshop on neural networks for signal processing*, (pp 557–565).

Hoyer, P. O. (2004). Non-negative matrix factorization with sparseness constraints. *Journal of machine learning research, 5*, 1457–1469.

Itoh, Y., Duarte, M. F., & Parente, M. (2017). Perfect recovery conditions for non-negative sparse modeling. *IEEE Transactions on Signal Processing, 65*(1), 69–80.

Kim, H., & Park, H. (2007). Sparse non-negative matrix factorizations via alternating non-negativity-constrained least squares for microarray data analysis. *Bioinformatics, 23*(12), 1495–1502.

Kim, J., Ramakrishnan, N., Marwah, M., Shah, A., & Park, H. (2013). Regularization paths for sparse nonnegative least squares problems with applications to life cycle assessment tree discovery. In *IEEE 13th international conference on data mining*, (pp. 360–369).

Lawson, C.L., & Hanson, R.J. (1995). Solving least squares problems. Society for Industrial and Applied Mathematics.

Lee, D. D., & Seung, H. S. (1997). Unsupervised learning by convex and conic coding. In *Advances in neural information processing systems*, (pp. 515–521).

Lee, D. D., & Seung, H. S. (1999). Learning the parts of objects by non-negative matrix factorization. *Nature, 401*(6755), 788–791.

Ma, W. K., Bioucas-Dias, J. M., Chan, T. H., Gillis, N., Gader, P., Plaza, A. J., Ambikapathi, A., & Chi, C. Y. (2013). A signal processing perspective on hyperspectral unmixing: Insights from remote sensing. *IEEE Signal Processing Magazine, 31*(1), 67–81.

Mohimani, G. H., Babaie-Zadeh, M., & Jutten, C. (2007). Fast sparse representation based on smoothed ℓ_0 norm. In *International conference on independent component analysis and signal separation* (pp. 389–396). Springer.

Morup, M., Madsen, K. H., & Hansen, L. K. (2008). Approximate L0 constrained non-negative matrix and tensor factorization. In *2008 IEEE international symposium on circuits and systems* (pp. 1328–1331). IEEE.

Nadisic, N., Vandaele, A., Gillis, N., & Cohen, J. E. (2020). Exact sparse nonnegative least squares. In *2020 IEEE international conference on acoustics, speech and signal processing* (pp. 5395–5399).

Nadisic, N., Vandaele, A., Gillis, N., & Cohen, J. E. (2021). Exact biobjective k-sparse nonnegative least squares. In *EUSIPCO 2021-29th European signal processing conference* (pp. 2079–2083).

Nguyen, T. T., Idier, J., Soussen, C., & Djermoune, E. H. (2019). Non-negative orthogonal greedy algorithms. *IEEE Transactions on Signal Processing, 67*(21), 5643–5658.

Osborne, M. R., Presnell, B., & Turlach, B. A. (2000). A new approach to variable selection in least squares problems. *IMA Journal of Numerical Analysis, 20*(3), 389–403.

Pati, Y. C., Rezaiifar, R., & Krishnaprasad, P. S. (1993). Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Proceedings of 27th Asilomar conference on signals, systems and computers* (pp. 40–44).

Peharz, R., & Pernkopf, F. (2012). Sparse nonnegative matrix factorization with ℓ_0-constraints. *Neurocomputing, 80*, 38–46.

Portugal, L. F., Judice, J. J., & Vicente, L. N. (1994). A comparison of block pivoting and interior-point algorithms for linear least squares problems with nonnegative variables. *Mathematics of Computation, 63*(208), 625–643.

Soussen, C., Gribonval, R., Idier, J., & Herzet, C. (2013). Joint k-step analysis of orthogonal matching pursuit and orthogonal least squares. *IEEE Transactions on Information Theory, 59*(5), 3158–3174.

Stojnic, M., Parvaresh, F., & Hassibi, B. (2009). On the reconstruction of block-sparse signals with an optimal number of measurements. *IEEE Transactions on Signal Processing, 57*(8), 3075–3085.

Tibshirani, R. (1996). Regression shrinkage and selection via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological), 58*(1), 267–288.

Tropp, J. A. (2004). Greed is good: Algorithmic results for sparse approximation. *IEEE Transactions on Information theory, 50*(10), 2231–2242.

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

Tropp, J. A., Gilbert, A. C., & Strauss, M. J. (2006). Algorithms for simultaneous sparse approximation. Part I: Greedy pursuit. *Signal processing, 86*(3), 572–588.

Yaghoobi, M., Wu, D., & Davies, M. E. (2015). Fast non-negative orthogonal matching pursuit. *IEEE Signal Processing Letters, 22*(9), 1229–1233.

Zhu, F. (2017). Hyperspectral unmixing: Ground truth labeling, datasets, benchmark performances and survey. Preprint arXiv:1708.05125.