EXACT SPARSE NONNEGATIVE LEAST SQUARES

Nicolas Nadisic⋆ *Arnaud Vandaele*⋆ *Nicolas Gillis*⋆ *Jeremy E. Cohen*†

⋆ University of Mons, Belgium.
†CNRS, Université de Rennes, Inria, IRISA, Rennes, France.

ABSTRACT

We propose a novel approach to solve exactly the sparse nonnegative least squares problem, under hard ℓ_0 sparsity constraints. This approach is based on a dedicated branch-and-bound algorithm. This simple strategy is able to compute the optimal solution even in complicated cases such as noisy or ill-conditioned data, where traditional approaches fail. We also show that our algorithm scales well, despite the combinatorial nature of the problem. We illustrate the advantages of the proposed technique on synthetic data sets, as well as a real-world hyperspectral image.

***Index Terms*—** nonnegative least squares, sparse coding, branch-and-bound.

1. INTRODUCTION

Nonnegative least squares (NNLS) problems of the form

$$\min ||Ax - b||_2^2 \text{ such that } x \geq 0, \tag{1}$$

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, and $b \in \mathbb{R}^m$, are a variant of least squares (LS) problems where the solution is required to be entry-wise nonnegative. Nonnegativity is useful for models where data are additive combinations of meaningful components [1]. For example, in hyperspectral images, the spectra of pixels are nonnegative linear combinations of the pure components they contain [2]. NNLS is also one of the principal tools used in the so-called "alternating approaches" to solve nonnegative matrix factorization [3]. As a constraint, nonnegativity is known to naturally induce sparsity, that is, produce solutions with few non-zero components; see for example [4] and the references therein. Sparse solutions express data points as combinations of only a few atoms, thus improving the interpretability of the decomposition. For instance, in the task of identifying the materials present in the pixels of a hyperspectral image, sparsity means a pixel will be composed of only a few materials. However, there is no guarantee on the sparsity of the solution to a general NNLS problem, while controlling the sparsity level of solutions is important in many applications, such as hyperspectral imaging [5] and sparse NMF [6]. Hence comes the need to design sparsity-enhancing techniques.

The sparsity of a vector x is typically measured by its ℓ_0-"norm", $\|x\|_0 = |\{i : x_i \neq 0\}|$. It is equal to the number of nonzero components in x. However, as this "norm" is non-convex and non-smooth, it is hard to enforce ℓ_0 constraints. One way to overcome this issue is to use the ℓ_1-norm as a convex surrogate, via the well-known LASSO approach [7]. This approach implies a regularization parameter (usually denoted λ) that can be hard to tune, especially if one requires a specific level of sparsity. This regularization also introduces a bias: the optimized problem is different, therefore the solution and its support may change. Thus, in some cases, it is preferable to directly solve an optimization problem with ℓ_0 constraints, without the use of a convex surrogate. In this setting, we propose a technique for solving the sparse NNLS problem with an explicit level of sparsity via a hard constraint on the ℓ_0-"norm" of the solution.

Problem: Given $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $k \in \mathbb{N}$, we wish to solve exactly the following sparse NNLS problem

$$x^* = \operatorname{argmin} \|Ax - b\|_2^2 \text{ such that } \|x\|_0 \leq k \text{ and } x \geq 0. \tag{2}$$

This problem is also referred to as nonnegative sparse coding [8]. Because of the discrete nature of the ℓ_0-"norm", Problem (2) is combinatorial, with $\binom{n}{k}$ possible supports for x. A brute-force algorithm would solve a NNLS subproblem for each possible support and select the solution with minimal error. This is the approach used by [9] in the context of sparse NMF. However, the number of possible supports grows exponentially with n and k. To avoid computing all of them, the solution we propose can prune the search space using a *branch-and-bound* strategy. With this technique, the problem is still solved exactly, but far fewer NNLS subproblems are solved, leading to reasonable computing times even for large n and k.

In Section 2, we discuss competing algorithmic approaches to the nonnegative sparse coding problem. In Section 3, we present a novel branch-and-bound algorithm. Finally, in Section 4, we illustrate the effectiveness of the approach on synthetic data sets and on a practical application of hyperspectral unmixing.

2. RELATED WORK

Sparse coding is a well-studied problem, with many heuristic approaches available for solving large-scale problems [10]. One of the most popular, the ℓ_1-regularized LASSO, solves the problem $\min_{x \geq 0} ||Ax - b||_2^2 + \lambda ||x||_1$ for some $\lambda > 0$. In this method, sparsity is encouraged through the regularization parameter λ. Selecting a value for this parameter that achieves a given level of sparsity is not straightforward. Moreover, existing theoretical guarantees for support recovery such as the Exact Recovery Condition [11] are quite restrictive for some uses of sparse NNLS, in particular when used as a routine for solving sparse NMF [9]. However, it benefits from the powerful properties of convex optimization, and efficient convex optimization algorithms exist that make use of optimality conditions to perform screening or design a stopping criterion [12].

Greedy algorithms are another popular approach. Techniques of this type start with an empty support, and then select the atoms one by one to enrich the support, until the desired sparsity, k, is reached. The atom selection is done in a *greedy* way, selecting at each iteration the atom that maximizes the decrease of the residual error.

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.

Orthogonal variants of these algorithms ensure an atom will not be selected more than once. Popular examples are orthogonal matching pursuit (OMP) [13] and orthogonal least squares (OLS) [14]. Nonnegative variants of these algorithms have been proposed; see [15] and the references therein. They aim to solve (2) heuristically, but theoretical recovery guarantees are similarly limited.

Bienstock proposed a branch-and-cut algorithm to solve a cardinality-constrained quadratic program (CCQP), a problem similar to (2) with different constraints, using continuous relaxation at every iteration [16]. Bertsimas and Shioda extended it by proposing a new way to solve the continuous relaxation problems [17]. Mhenni et. al generalized this CCQP and proposed a novel branch-and-bound algorithm to solve exactly the continuous relaxation problems using dedicated optimization methods [18, 19].

3. THE ARBORESCENT ALGORITHM

We propose a novel technique to solve the sparse NNLS problem exactly via a branch-and-bound algorithm called `arborescent`[1]. In this algorithm, the possible patterns of zeros (that is, the possible supports) in the solution vector are enumerated on a tree, as shown in Figure 1. Each node represents an over-support $\mathcal{K}$ of x. The entries of x indexed by $\mathcal{K}$ are those that are not constrained to be 0. The cardinality of $\mathcal{K}$ is equal to the current tree depth. Exploring a node means solving the following NNLS subproblem

$$f^*(\mathcal{K}) = \min ||A(:,\mathcal{K})x(\mathcal{K}) - b||_2^2 \text{ such that } x(\mathcal{K}) \geq 0, \qquad (3)$$

where $x(\mathcal{K})$ is the subvector of x whose entries are specified by the subset $\mathcal{K} \subseteq \{1, 2, \ldots, n\}$. The value $f^*(\mathcal{K})$ is the *error* associated with the node corresponding to $\mathcal{K}$. In `arborescent`, an *active-set* algorithm [20] is used as the NNLS solver. The key property of active-set methods is that they solve the NNLS problem exactly, without a need for parameter tuning. Note that active-set methods allow for a *warm start*, that is, the algorithm can be initialized at the current node with the solution from a previous node. This significantly speeds up the iterative process since it starts close to the optimal solution.

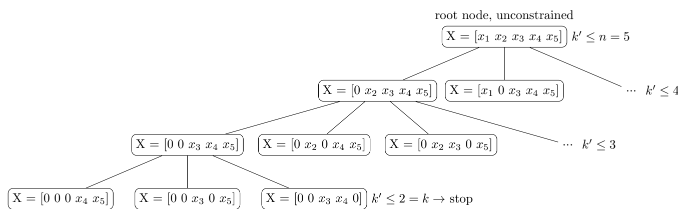

Fig. 1. Ex. of the `arborescent` search tree, for $n = 5$ and $k = 2$.

Branch. The root of the tree represents the unconstrained problem. No component of x is constrained to zero, that is, $\mathcal{K} = \{1, 2, \ldots, n\}$. At this root node, the support of x is required to have cardinality at most $k' = n$ (unconstrained). From this root we generate one child node for each component of x, where the respective component is not in the support (and thus constrained to zero). The cardinality of the support of x in these children nodes is therefore at most $k' = n - 1$. The tree is constructed recursively, by constraining one additional component to be equal to zero per node. When the tree has reached the depth were the cardinality of the support set is $k' = k$, the desired sparsity is obtained and the leaf nodes at this depth represent feasible solutions to (2).

As described above, several nodes of the `arborescent` tree would correspond to the same over-support $\mathcal{K}$. For example, in Figure 1 there would be a redundant node $\{0, 0, x_3, x_4, x_5\}$ as the child of both $\{0, x_2, x_3, x_4, x_5\}$ and $\{x_1, 0, x_3, x_4, x_5\}$. To avoid this redundant computations, we first order the nodes in the root node in order of increasing magnitude of the entries of the unconstrained solution. Let x^* be the optimal solution of (1), we reorder the entries of x^* so that the $x_1^* \leq x_2^* \leq \cdots \leq x_n^*$. With this information, we can deduce if the node has already been explored.

Bound. As an additional entry of x is constrained to be zero for each child node, the error of a given node will always be greater than or equal to the error of its parent, by construction. As soon as we reached a leaf of the tree, we obtain a feasible solution whose error is therefore an upper bound for (2). Hence if a given node has an error greater than this upper bound, all the children of this node will also have a higher error, and it can be safely pruned. A key element of a branch-and-bound algorithm is the strategy used to explore the tree. A relevant exploration can quickly lead to a good admissible solution, and thus to a bound allowing the pruning of large parts of the search space. As described above, in `arborescent`, the solution x^* to the unconstrained problem is computed in the root node and the components of x^* are sorted in ascending order. Thereafter, the exploration is done depth-first, and "left-first". Among all children of a node, the node furthest left is explored first, that is, the components that are closest to zero in the unconstrained solution are constrained first. This is based on the hypothesis that if a component is close to zero in the solution of the unconstrained problem, it is more likely to be zero in the optimal solution of the constrained problem. We have observed empirically that this approach outperforms more complex strategies, including greedy node selection (that is, selecting the node with the lowest error).

Pseudocode. `arborescent` is detailed in Algorithm 1. The set P is the pool of nodes. On line 5, it is initialized with the root node, that has a full support (no component is constrained). On line 6, a node is selected in P, following the strategy described above, and removed from P on line 8. On line 9, the NNLS subproblem defined by A and b limited to the corresponding over-support $\mathcal{K}$ is solved using the initialization x. If the error is larger that the current lowest error, no child node can be optimal, and the node is pruned (line 11). Otherwise, the exploration continues. If the desired sparsity level k is not reached, a new node is generated for every component of the over-support (lines 14 and 15). If the value of k is reached, the error of the current node is compared to the lowest error found so far (line 17), and if it is lower, it replaces it (line 18).

Computational complexity. In the worst case, `arborescent` has to build and explore the whole tree, that is, to solve a number of NNLS subproblems equal to $\sum_{l=k}^{r} \binom{r}{l}$. In the best case, all nodes except the leftmost ones are pruned, so the number of NNLS subproblems to solve is $\sum_{l=k}^{r} l$. In practice, the number or nodes explored is far from the worst case; see section 4.2.

4. EXPERIMENTS

The code of `arborescent`, along with test scripts, is provided on `gitlab.com/nnadisic/sparse-nmf`. All experiments were performed on a personal computer with an i5 processor, with a clock frequency of 2.30GHz. `arborescent` is implemented in C++ with a MEX interface, and its competitors are implemented in Matlab. All algorithms are monothreaded.

[1] `arborescent` Realizes a Branch-and-bound Optimization to Require Explicit Sparsity Constraints to be Enforced in NNLS Tasks.

Algorithm 1: `arborescent`

Input: $A \in \mathbb{R}_+^{m \times n}$, $b \in \mathbb{R}_+^m$, $k \in \{1, 2, \ldots, n\}$
Output: $x_{best} = \arg\min_{x \geq 0} ||Ax - b||_2^2$ s.t. $||x||_0 \leq k$

```
1  Init x_0 ← NNLS(A, b)
2  Sort the elements in x_0 in increasing order
3  Init K_0 ← {1, ..., n}
4  Init best_error ← +∞
5  Init P ← {(K_0, x_0)}
6  while P ≠ ∅ do
7      (K, x_parent) = P.select()
8      P ← P \ {(K, x_parent)}
9      (error, x) ← NNLS(A(:, K), b, x_parent(K))
10     if error > best_error then
11         prune (do nothing)
12     else
13         if size(K) > k then
14             foreach i ∈ K do
15                 P ← P ∪ {(K \ {i}, x) }
16         else
17             if error < best_error then
18                 best_error ← error
19                 x_best ← x
20 return x_best
```

4.1. Comparison on Synthetic Datasets

We compare `arborescent` to four algorithms:

• A ℓ_1-penalized coordinate descent, implemented by modifying the Matlab code from [21]. This method uses a dynamic tuning of the regularization parameter λ to reach the desired sparsity. After a first run, the support of the solution is restricted to the k highest values, and a NNLS solver is run only for the values within the support so that the solution is guaranteed to be k-sparse while the bias of the ℓ_1 approximation is removed. We refer to this method as L1-CD.

• The interior-point method SDPT3 (version 4.0) [22, 23] with CVX as a modeling system [24, 25] that solves the ℓ_1-penalized problem. The regularization parameter λ is chosen as the final value of λ obtained by the previous method presented above, and the same post-processing is used. We refer to this method as CVX.

• Nonnegative OMP (NNOMP) and Nonnegative OLS (NNOLS) [15] for which the Matlab codes are provided by the authors.

Synthetic test cases are built by generating a random matrix $A \in \mathbb{R}_+^{m \times n}$ and a random k-sparse vector $x_{true} \in \mathbb{R}_+^n$, computing $b = Ax_{true}$, and trying to find again x_{true} with A, b and k as parameters of the sparse NNLS algorithm. We consider four cases: well-conditioned A and noiseless b, well-conditioned A and noisy b, ill-conditioned A and noiseless b, ill-conditioned A and noisy b. For well-conditioned A, each entry of A is generated using the uniform distribution in [0,1] (`rand(m,n)` in Matlab). For ill-conditioned A, we proceed in the same way, then compute the SVD of $A = U\Sigma V^T$, replace Σ by values between 10^{-6} and 1 equally spaced in a logscale (`logspace(-6,0,n)` in Matlab), and finally take $A = U\Sigma V^T$. For b, we also use the uniform distribution in [0,1]. For the noise e added to b, we generate a vector where each entry is generated using the normal distribution of mean 0 and variance 1 (`randn(m,1)` in Matlab), then rescale $e \leftarrow 0.05 \frac{e}{||e||_2} ||b||_2$ so that $||e||_2 = 0.05||b||_2$ (the noise level is 5%). We generate such data sets for three values of m: 1000, 100, and 20, with fixed $n = 20$ and $k = 10$, for a total of 12 test settings. For each setting, 100 data sets are randomly generated, and processed by the 5 algorithms. For each algorithm, we measure the relative error $\frac{||Ax-b||_2}{||b||_2}$ averaged over these 100 runs, as well as the average computational time, and the number of *successes* (meaning the number of times the support of x_{true} is recovered).

Tables 1 to 3 present the results of the experiments. As expected, `arborescent` always has an error of zero for noiseless data as it solves the problem exactly. This is particularly interesting when A is ill-conditioned, because then the competitors generally fail to identify the support. For noisy data, it always outperforms the competitors in terms of error, and of number of successes. It has a large majority of successes in all but the most difficult cases, when $m = 20$ and b is noisy. However, the superior performance of `arborescent` comes at the cost of an increase in running time.

4.2. Scalability of `arborescent`

To test the scalability of our solution, we run it with the same data model as in Section 4.1, in the well-conditioned and noiseless case, with the following parameters: $m = 1000$, $n = \{10, 12, ..., 60\}$ and $k = \frac{n}{2}$. Again 100 datasets are generated for each pair $\{n, k\}$, and processed by `arborescent`. The results are presented in table 4. We see that, despite the exponentially increasing size of the combinatorial problem (in the worst case scenario, `arborescent` would have to explore $\sum_{l=k}^{n} \binom{n}{l}$ nodes; note that $\binom{60}{30} > 10^{17}$), most of the search space is pruned, and only a few nodes are explored. The running times does not increase exponentially, allowing the use of our solution in medium-scale problems. Note the computational time is not always monotonic as n increases (for example, $n = 52$). The reason is that the NNLS is sometimes significantly more difficult to solve (hence more nodes need to be explored) which increases the computational time. This happens very rarely.

4.3. Application on a Hyperspectral Image

In this section, `arborescent` is used to identify the materials present in the pixels of a hyperspectral image. The input data is the well-known Cuprite image [26]. It features $250 \times 191 = 47750$ pixels in $m = 188$ denoised spectral bands. We refer to this 188×47750 matrix as M. For the dictionary D, we use the output of the successive projection algorithm algorithm [27] that selects a subset of the columns of M. The number of materials is $n = 12$, so D is a 188×12 matrix, and we want to find a 12×47750 matrix, V, representing the abundances of materials in each pixel such that $M \approx DV$. Every column of this matrix represents a pixel, and an independent NNLS subproblem: V is computed by solving 47750 NNLS problems. As an additional constraint, we require the solution to be column-wise k-sparse with $k = 5$. We solve the problem first with a NNLS algorithm with no sparsity constraints (we refer to it as NNLS), then with `arborescent` and with L1-CD, using the same post-processing and settings as section 4.1. For each algorithm we report the running time, the relative reconstruction error $\frac{||M-DV||_F}{||M||_F}$, and the average column-wise sparsity. Table 5 shows the results of the three considered algorithms. Although NNLS naturally produces a relatively sparse solution, it is higher than the sparsity target. L1-CD, on the contrary, produces a solution sparser than required (showing the difficulty to tune the parameter), at the cost of a high error (from 1.74% for NNLS to 4.21% for L1-CD). `arborescent` produces a solution with the required sparsity, and at the same time a low error, very close to the NNLS one (1.78%). Figure 2 shows the resulting material abundances for `arborescent`.

	Well-cond A, Noiseless b			Well-cond A, Noisy b			Ill-cond A, Noiseless b			Ill-cond A, Noisy b		
	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.
L1-CD	0	6.83	100	5.01	3.13	97	3.50	8.61	16	6.28	4.12	9
CVX	0	796.90	100	4.96	634.62	100	0.08	609.45	96	4.98	571.72	98
NNOMP	0	5.60	100	4.96	3.73	100	2.79	4.14	3	5.83	3.63	3
NNOLS	0	4.67	100	4.96	3.36	100	1.95	4.21	23	5.50	3.13	22
arbo.	0	43.09	100	4.96	1369.30	100	0	29.80	100	4.96	1223.20	100

Table 1. Results for $m = 1000$. Relative error is in percent. Time is in milliseconds. Succ. is the number of successes, that is, the number of times the algorithm recovered the support of x_{true}.

	Well-cond A, Noiseless b			Well-cond A, Noisy b			Ill-cond A, Noiseless b			Ill-cond A, Noisy b		
	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.
L1-CD	0.27	1.93	93	4.97	2.48	84	3.65	2.22	11	6.02	1.97	7
CVX	0	536.95	100	4.73	502.00	100	0.02	508.60	98	4.84	489.68	58
NNOMP	0.48	2.83	89	4.98	2.79	83	3.09	2.83	3	5.50	2.80	2
NNOLS	0.20	2.52	95	4.88	2.62	91	2.15	2.55	14	5.11	2.55	18
arbo.	0	46.32	100	4.73	1145.10	100	0	29.39	100	4.71	1304.40	63

Table 2. Results for $m = 100$. Measures are defined as in table 1.

	Well-cond A, Noiseless b			Well-cond A, Noisy b			Ill-cond A, Noiseless b			Ill-cond A, Noisy b		
	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.	Rel. Err.	Time	Succ.
L1-CD	2.88	1.39	23	4.29	1.31	15	3.41	1.44	5	4.66	1.53	1
CVX	0.00	532.23	100	3.26	495.62	23	0.95	548.41	39	3.07	509.10	7
NNOMP	3.02	2.85	12	3.85	2.74	6	2.07	2.96	2	3.57	2.88	0
NNOLS	2.57	2.59	18	3.73	2.54	10	1.48	3.02	12	3.26	3.08	1
arbo.	0	46.84	100	3.09	1472.20	30	0	34.20	100	2.83	1283.70	11

Table 3. Results for $m = 20$. Measures are defined as in table 1.

n	Time (ms.)	NNE	n	Time (ms.)	NNE
10	3.6504	9.24	36	678.09	60.62
12	6.3706	11.02	38	667.68	48.9
14	9.3337	15.41	40	1034.3	63.56
16	16.386	18.16	42	3134.9	166.41
18	24.837	23.15	44	2070.5	97.17
20	42.344	29.37	46	2610.7	97.94
22	83.267	41.06	48	8236.9	249.19
24	97.774	35.82	50	1936.8	52.14
26	137.17	38.1	52	39119	900.74
28	277.33	59.06	54	6209.8	132.3
30	268.09	48.33	56	9173.5	161.73
32	392.77	54.57	58	8752.4	146.14
34	601.7	67.74	60	14149	182.91

Table 4. Results of the scalability test of `arborescent`. Time is the average over 100 instances (for $k = n/2$, $m = 1000$). NNE is the average number of nodes explored in the tree.

	NNLS	L1-CD	arbo
Time (s.)	15.81	22.35	1053
Rel. Error (%)	1.74	4.21	1.78
Sparsity	6.61	4.34	4.77

Table 5. Results of the regression on Cuprite image.

5. CONCLUSION

We proposed `arborescent`, a dedicated branch-and-bound algorithm to tackle the k-sparse nonnegative least squares problem exactly. It works in very general settings, where existing approaches such as LASSO or greedy algorithms fail to identify the support of the optimal solution. The branch-and-bound technique allows for substantial pruning of the search space, so this combinatorial problem can be solved exactly with a drastic reduction of computation time over brute-force methods. The combination of relative speed, scalability, and an exact solution provide a broadly applicable technique for medium-scale sparse nonnegative least squares problems.

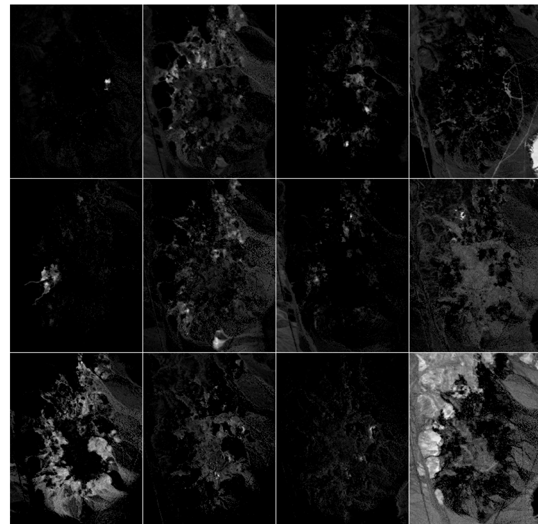

Fig. 2. Material abundances in the Cuprite image. These correspond to true materials; for example the first row corresponds to Muscovite, Dumortierite, Alunite and Montmorillonite; see [28]. (Note that spatial coherence is strong while this is not explicitly enforced as the sparse NNLS are solved independently for each pixel. This indicates that the solution is meaningful.)

6. REFERENCES

[1] Daniel D. Lee and H. Sebastian Seung, "Unsupervised learning by convex and conic coding," in *Advances in neural information processing systems*, 1997, pp. 515–521.

[2] José M. Bioucas-Dias, Antonio Plaza, Nicolas Dobigeon, Mario Parente, Qian Du, Paul Gader, and Jocelyn Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 5, no. 2, pp. 354–379, 2012.

[3] Nicolas Gillis, "The why and how of nonnegative matrix factorization," *Regularization, Optimization, Kernels, and Support Vector Machines*, vol. 12, no. 257, pp. 257–291, 2014.

[4] Simon Foucart and David Koslicki, "Sparse recovery by means of nonnegative least squares," *IEEE Signal Processing Letters*, vol. 21, no. 4, pp. 498–502, 2014.

[5] Wing-Kin Ma, José M. Bioucas-Dias, Tsung-Han Chan, Nicolas Gillis, Paul Gader, Antonio J. Plaza, ArulMurugan Ambikapathi, and Chong-Yung Chi, "A signal processing perspective on hyperspectral unmixing: Insights from remote sensing," *IEEE Signal Processing Magazine*, vol. 31, no. 1, pp. 67–81, 2013.

[6] Patrik O. Hoyer, "Non-negative matrix factorization with sparseness constraints," *Journal of Machine Learning Research*, vol. 5, no. Nov, pp. 1457–1469, 2004.

[7] Robert Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[8] Patrik O. Hoyer, "Non-negative sparse coding," in *Proceedings of the 12th IEEE Workshop On Neural Networks for Signal Processing.*, 2002, pp. 557–565.

[9] Jeremy E. Cohen and Nicolas Gillis, "Nonnegative Low-rank Sparse Component Analysis," in *2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 8226–8230.

[10] Simon Foucart and Holger Rauhut, *A Mathematical Introduction to Compressive Sensing*, Applied and Numerical Harmonic Analysis. Springer New York, 2013.

[11] Rmi Gribonval and Morten Nielsen, "Sparse representations in unions of bases," *IEEE transactions on Information theory*, vol. 49, no. 12, pp. 3320–3325, 2003.

[12] Laurent El Ghaoui, Vivian Viallon, and Tarek Rabbani, "Safe feature elimination in sparse supervised learning," *CoRR*, vol. abs/1009.4219, 2010.

[13] Yagyensh Chandra Pati, Ramin Rezaiifar, and Perinkulam Sambamurthy Krishnaprasad, "Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition," in *Proceedings of 27th Asilomar Conference on Signals, Systems and Computers*. IEEE, 1993, pp. 40–44.

[14] Sheng Chen, Stephen A. Billings, and Wan Luo, "Orthogonal least squares methods and their application to non-linear system identification," *International Journal of control*, vol. 50, no. 5, pp. 1873–1896, 1989.

[15] Thanh T. Nguyen, Jerome Idier, Charles Soussen, and El-Hadi Djermoune, "Non-Negative Orthogonal Greedy Algorithms," *IEEE Transactions on Signal Processing*, 2019.

[16] Daniel Bienstock, "Computational study of a family of mixed-integer quadratic programming problems," *Mathematical programming*, vol. 74, no. 2, pp. 121–140, 1996.

[17] Dimitris Bertsimas and Romy Shioda, "Algorithm for cardinality-constrained quadratic optimization," *Computational Optimization and Applications*, vol. 43, no. 1, pp. 1–22, 2009.

[18] Sébastien Bourguignon, Jordan Ninin, Hervé Carfantan, and Marcel Mongeau, "Exact sparse approximation problems via mixed-integer programming: Formulations and computational performance," *IEEE Transactions on Signal Processing*, vol. 64, no. 6, pp. 1405–1419, 2015.

[19] Ramzi Mhenni, Sébastien Bourguignon, and Jordan Ninin, "Global optimization for sparse solution of least squares problems," 2019, `https://hal.archives-ouvertes.fr/hal-02066368/`.

[20] Luís F. Portugal, Joaquim J. Judice, and Luís N. Vicente, "A comparison of block pivoting and interior-point algorithms for linear least squares problems with nonnegative variables," *Mathematics of Computation*, vol. 63, no. 208, pp. 625–643, 1994.

[21] Nicolas Gillis, "Sparse and unique nonnegative matrix factorization through data preprocessing," *Journal of Machine Learning Research*, vol. 13, no. Nov, pp. 3349–3386, 2012.

[22] Kim-Chuan Toh, Michael J. Todd, and Reha H. Tütüncü, "Sdpt3a matlab software package for semidefinite programming, version 1.3," *Optimization methods and software*, vol. 11, no. 1-4, pp. 545–581, 1999.

[23] Reha H. Tütüncü, Kim-Chuan Toh, and Michael J. Todd, "Solving semidefinite-quadratic-linear programs using SDPT3," *Mathematical programming*, vol. 95, no. 2, pp. 189–217, 2003.

[24] Inc. CVX Research, "CVX: Matlab software for disciplined convex programming, version 2.0," `http://cvxr.com/cvx`, 2012.

[25] Michael C. Grant and Stephen P. Boyd, "Graph implementations for nonsmooth convex programs," in *Recent advances in learning and control*, pp. 95–110. Springer, 2008.

[26] "USGS spectroscopy lab," `https://www.usgs.gov/labs/spec-lab`, Accessed: 2019-10-21.

[27] Nicolas Gillis and Stephen A. Vavasis, "Fast and robust recursive algorithms for separable nonnegative matrix factorization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 4, pp. 698–714, 2014.

[28] ArulMurugan Ambikapathi, Tsung-Han Chan, Wing-Kin Ma, and Chong-Yung Chi, "Chance-constrained robust minimum-volume enclosing simplex algorithm for hyperspectral unmixing," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 49, no. 11, pp. 4194–4209, 2011.