# A Filtered Bucket-Clustering Method for Projection onto the Simplex and the $\ell_1$ Ball

Guillaume PEREZ, Michel BARLAUD, Lionel FILLATRE, Jean-Charles RÉGIN

Université Côte d'Azur, CNRS, I3S, 06900 Sophia Antipolis, France
guillaume.perez06@gmail.com, barlaud@i3s.unice.fr, fillatre@i3s.unice.fr, jcregin@gmail.com

**Résumé –** Nous proposons une nouvelle méthode pour le calcul de la projection d'un vecteur de taille arbitraire sur le simplex ou la boule $\ell_1$. Notre méthode fusionne deux principes. Le premier est une recherche originale de la projection utilisant un algorithme de Bucket. Le second est un filtrage, au vol, des valeurs du vecteur qui ne peuvent pas appartenir à la projection. La combinaison de ces deux principes offre un algorithme simple et efficace.

**Abstract –** We propose in this paper a new method processing the projection of an arbitrary size vector onto the simplex or the $\ell_1$ ball. Our method merges two principles. The first one is an original search of the projection using a bucket algorithm. The second one is a filtering, on the fly, of the values that cannot be part of the projection. The combination of these two principles offers a simple and efficient algorithm.

## 1 Introduction

Looking for sparsity appears in many applications. Minimizing the number of non-zero components of a given vector is generally a very difficult problem. Hence the common solution is to minimize the $\ell_1$ norm of the vector [9, 4]. To this purpose, it is crucially important to have a simple algorithm to project a vector onto the $\ell_1$ ball. Given a vector $y = (y_1, y_2, \ldots, y_d) \in \mathbb{R}^d$ and a real $a > 0$, we aim at computing its projection $P_{\mathcal{B}_a}(y)$ onto the $\ell_1$ ball $\mathcal{B}_a$ of radius $a$:

$$\mathcal{B}_a = \left\{ x \in \mathbb{R}^d | \ \|x\|_1 \leqslant a \right\}, \tag{1}$$

where $\|x\|_1 = \sum_{i=1}^d |x_i|$. The projection $P_{\mathcal{B}_a}(y)$ is defined by

$$P_{\mathcal{B}_a}(y) = \arg\min_{x \in \mathcal{B}_a} \|x - y\|_2 \tag{2}$$

where $\|x\|_2$ is the Euclidean norm. As shown in [5] and revisited in [2], the projection onto the $\ell_1$ ball can be derived from the projection onto the simplex $\Delta_a$:

$$\Delta_a = \left\{ x \in \mathbb{R}^d \,|\, \sum_{i=1}^d x_i = a \text{ and } x_i \geqslant 0, \forall i = 1, \ldots, d \right\}. \tag{3}$$

Let the sign function $\text{sign}(v)$ defined as $\text{sign}(v) = 1$ if $v > 0$, $\text{sign}(v) = -1$ if $v < 0$ and $\text{sign}(v) = 0$ otherwise, for any real value $v \in \mathbb{R}$. The projection of $y$ onto the $\ell_1$ ball is given by the following formula:

$$P_{\mathcal{B}_a}(y) = \begin{cases} y & \text{if } \ y \in \mathcal{B}_a, \\ (\text{sign}(y_1)z_1, \ldots, \text{sign}(y_d)z_d) & \text{otherwise,} \end{cases} \tag{4}$$

where $z = P_{\Delta_a}(|y|)$ with $|y| = (|y_1|, |y_2|..., |y_d|)$ is the projection of $|y|$ onto $\Delta_a$. The fast computation of the projection $x = P_{\Delta_a}(y)$ for any vector $y$ is of utmost importance to deal with sparse vector surrogates. An important property has been established to compute this projection. It was shown [2] that there exists a unique $\tau = \tau_y \in \mathbb{R}$ such that

$$x_i = \max\{y_i - \tau, 0\}, \forall i = 1, \ldots, d. \tag{5}$$

The projection is almost equivalent to a thresholding operation. The main difficulty is to compute quickly the threshold $\tau_y$ for any vector $y$. Let $y_{(i)}$ be the $i$th largest value of $y$ such that $y_{(d)} \leqslant y_{(d-1)} \leqslant \ldots \leqslant y_{(1)}$. It is interesting to note that (5) involves that $\sum_{i=i}^d \max\{y_i - \tau, 0\} = a$. Let $S^*$ be the support of $x$, i.e., $S^* = \{i | x_i > 0\}$. Then,

$$a = \sum_{i=1}^d x_i = \sum_{i \in S^*} x_i = \sum_{i \in S^*} (y_i - \tau).$$

It follows that $\tau_y = (\sum_{i \in S^*} y_i - a)/|S^*|$ where $|S^*|$ is the number of elements of $S^*$. The following property allows us to compute the threshold $\tau_y$. Let

$$\varrho_j(y) = \left( \sum_{i=1}^j y_{(i)} - a \right) / j \tag{6}$$

for any $j = 1, \ldots, d$. Then, it was shown that $\tau_y = \varrho_{K_y}(y)$ where

$$K_y = \max\{k \in \{1, \ldots, d\} \mid \varrho_k(y) < y_{(k)}\}. \tag{7}$$

Looking for $K_y$, or equivalently $y_{(\tau_y)}$, allows us to find immediately the threshold $\tau_y$. The most famous algorithm to compute the projection, which has been presented in [6], is based on (7). It consists on sorting the values and then finding the first value satisfying (7). A possible implementation is given in Algorithm 1. The worst case complexity of this algorithm is $O(d \log d)$. Several other methods have been proposed [2, 7, 8, 10], outperforming this simple approach.

**Algorithm 1** Sort based algorithm [6]

**Data**: $y, a$
$u \leftarrow sort(y)$
$K \leftarrow \max_{1\leqslant k\leqslant d}\{k|(\sum_{r=1}^{k} u_r - a)/k < u_k\}$
$\tau \leftarrow (\sum_{r=1}^{K} u_r - a)/K$
**for** $i \in 1..|y|$ **do**
  $x_i \leftarrow \max(y_i - \tau, 0)$

# 2 Bucket Partitioning

This section describes the bucket-based algorithm which achieves a better worst case complexity than existing algorithms.

## 2.1 Theoretical Principle

The bucket-based method is based on the existence of $K_y$ in (7). The main idea is to split recursively the vector $y$ into a hierarchical family of $B \geqslant 2$ ordered sub-vectors $\tilde{y}_b^k$ with $b = 1, \ldots, B$ and $k = 1, \ldots, \bar{k}$. The number of recursive splitting, also called the number of levels, is $\bar{k}$. It may depend on $y$ contrary to $B$ which is constant. The sub-vectors are ordered in the sense that all elements of $\tilde{y}_b^k$ are smaller than the ones of $\tilde{y}_{b+1}^k$ for all $b = 1, \ldots, B-1$. Each sub-vector $\tilde{y}_b^k$ is called a bucketed vector or simply a bucket. The goal is to find the bucket which contains $y_{(K_y)}$. We will show that only one bucket at level $k$ is relevant because only this single bucket is necessary to identify the bucket which actually contains $y_{(K_y)}$ where $K_y$ is defined in (7). Hence, all the buckets $\tilde{y}_b^{k+1}$ at level $k+1$ are contained in a single bucket $\tilde{y}_{b_k}^k$ where $b_k$ is the identification number of the bucket at level $k$ which needs a deeper analysis. By convention, we assume that $\tilde{y}_{b_0}^0 = y$ and $b_0 = 1$.

Let us define the process to create and analyze the hierarchical family of buckets. For any level $k+1 \geqslant 1$, let us consider the interval $I^{k+1}$ defined by

$$I^{k+1} = [\min \tilde{y}_{b_k}^k, \max \tilde{y}_{b_k}^k] \tag{8}$$

where $\min \tilde{y}_b^k$, resp. $\max \tilde{y}_b^k$, denotes the minimum element, resp. maximum element, of bucket $\tilde{y}_b^k$. Let us consider a partition of $I^{k+1}$ into $B$ ordered sub-intervals $I_1^{k+1}, \ldots, I_B^{k+1}$. Let $h^{k+1} : I^{k+1} \mapsto \{1, \ldots, B\}$ be the bucketing function such that $h^{k+1}(v) = b$ when the real value $v$ belongs to $I_b^{k+1}$. The bucket $\tilde{y}_{b_k}^k$ is then splitted into $B$ ordered sub-vectors $\tilde{y}_b^{k+1}$ such that

(i) $\tilde{y}_b^{k+1} = (y_i)_{i\in S_b^{k+1}}$,

(ii) $S_b^{k+1} = \{i \in S_{b_k}^k \mid h^{k+1}(y_i) = b\}$,

(iii) $\max \tilde{y}_b^{k+1} \leqslant \min \tilde{y}_{b+1}^{k+1}$ for all $b = 1, \ldots, B-1$,

with the convention $S_{b_0}^0 = \{1, \ldots, d\}$. We get $\left|S_B^k\right| \geqslant 1$ at any level $k \geqslant 1$ because of the definition of $I^{k+1}$.

Let $C_{b_0+1}^0 = 0$ and, for any $k > 0$,

$$C_b^{k+1} = \begin{cases} C_{b_k+1}^k + \sum_{b'\geqslant b} \sum_{i\in S_{b'}^{k+1}} y_i & \text{if } b_k < B, \\ C_{b_{k-1}+1}^{k-1} + \sum_{b'\geqslant b} \sum_{i\in S_{b'}^{k+1}} y_i & \text{if } b_k = B, \end{cases} \tag{9}$$

be the cumulative sum of buckets $\tilde{y}_b^{k+1}$ to $\tilde{y}_B^{k+1}$, including also the cumulative sum of the buckets kept at previous levels 1, 2, …, $k$ which have been not discarded. Thus, by definition, $C_b^{k+1}$ is the cumulative sum of all $y_{(i)}$ for $i \leqslant N_b^{k+1}$ where

$$N_b^{k+1} = \begin{cases} N_{b_k+1}^k + \sum_{b'\geqslant b} \left|S_{b'}^{k+1}\right| & \text{if } b_k < B, \\ N_{b_{k-1}+1}^k + \sum_{b'\geqslant b} \left|S_{b'}^{k+1}\right| & \text{if } b_k = B, \end{cases} \tag{10}$$

is the number of elements in the family of buckets $\tilde{y}_b^{k+1}, \ldots, \tilde{y}_B^{k+1}$, including also the number of elements not discarded at previous levels $1, 2, \ldots, k$. We adopt the convention $N_{b_0+1}^0 = 0$. It follows from (9) and (10) that

$$\varrho_{N_b^{k+1}}(y) = (C_b^{k+1} - a)/N_b^{k+1}. \tag{11}$$

By definition of the buckets, it follows that

$$\min \tilde{y}_{b:B}^{k+1} = y_{(N_b^{k+1})} \tag{12}$$

where $\tilde{y}_{b:B}^{k+1}$ is the vector obtained from the concatenation of buckets $\tilde{y}_b^{k+1}, \ldots, \tilde{y}_B^{k+1}$. If $\varrho_{N_B^{k+1}}(y) \geqslant \min \tilde{y}_B^{k+1} = y_{(N_B^{k+1})}$, then $K_y < N_B^{k+1}$ according to (7). Hence, we can discard all the remaining buckets $\tilde{y}_b^{k+1}$ for $b < B$ and continue the bucket-based exploration of $\tilde{y}_B^{k+1}$ to approximate $K_y$ more accurately. Otherwise, we know that $K_y \geqslant N_B^{k+1}$, thus we continue the analysis of the remaining buckets $\tilde{y}_b^{k+1}$. Let $b_{k+1}$ be the largest $b$ such that

$$\varrho_{N_b^{k+1}}(y) \geqslant \min \tilde{y}_{b:B}^{k+1}. \tag{13}$$

When $k = 0$, the index $b_1$ may not exist if $y \in \Delta_a$: the process is stopped. Otherwise, if $b_{k+1}$ exists, then $K_y < N_{b_{k+1}}^{k+1}$. Hence, we can discard all the remaining buckets $\tilde{y}_b^{k+1}$ for $b < b_{k+1}$ and continue the analysis with $\tilde{y}_{b_{k+1}}^{k+1}, \ldots, \tilde{y}_B^{k+1}$. However, by definition of $b_{k+1}$, we know that

$$\varrho_{N_{b_{k+1}+1}^{k+1}}(y) < \min \tilde{y}_{b_{k+1}+1:B}^{k+1}. \tag{14}$$

Hence, $K_y$ necessarily satisfies $N_{b_{k+1}+1}^{k+1} \leqslant K_y < N_{b_{k+1}}^{k+1}$. To compute $K_y$, it is then sufficient to explore only $\tilde{y}_{b_{k+1}}^{k+1}$.

From the definition of $I^{k+1}$, it is easy to verify that the size of the bucket $\tilde{y}_{b_k}^k$ is strictly decreasing as a function of $k$ since the boundaries of $I^k$ are two elements of the previous bucket $\tilde{y}_{b_{k-1}}^{k-1}$ and $B \geqslant 2$. After a finite number $\bar{k}$ of iterations, $\tilde{y}_{b_{\bar{k}}}^{\bar{k}}$ contains only one value or some repetitions, say $t_0 \geqslant 1$, of the same value, say $v_0$. It is straightforward to verify that

$$\frac{C_{b_{\bar{k}}}^{\bar{k}} - a}{N_{\bar{k}}^{\bar{k}}} \geqslant v_0 \iff \frac{C_{b_{\bar{k}}}^{\bar{k}} - t_0 v_0 - a}{N_{\bar{k}}^{\bar{k}} - t_0} \geqslant v_0. \tag{15}$$

Hence, this last bucket $\tilde{y}_{b_{\bar{k}}}^{\bar{k}}$ can not contain $y_{(K_y)}$. We stop the exploration. It follows that $\tau_y$ satisfies

$$\tau_y = \begin{cases} \varrho_{N_{b_{\bar{k}}+1}^{\bar{k}}}(y) & \text{if} \quad b_{\bar{k}} < B, \\ \varrho_{N_{b_{\bar{k}-1}+1}^{\bar{k}-1}}(y) & \text{if} \quad b_{\bar{k}} = B. \end{cases} \tag{16}$$

The complexity of this algorithm essentially depends on the choice of the bucketing functions $h_k$ at any level $k$. This aspect is studied in the next subsection. A possible implementation is given in Algorithm 2.

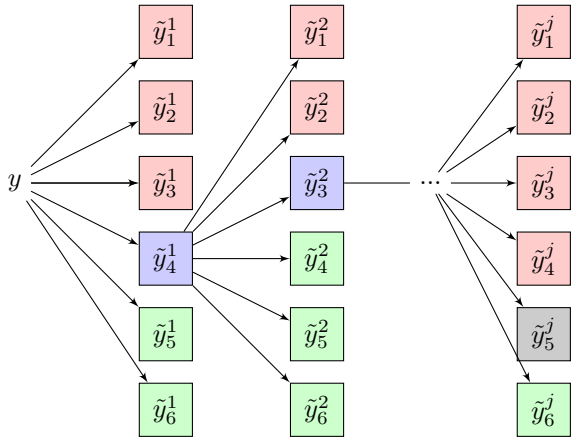


Figure 1: Principle of "hierarchical bucket filtering".

**Example** A synthetic example is given in figure 1. The vector $y$ is split in 6 buckets. Starting from the vector of the largest values $\tilde{y}_6^1$, the algorithm checks if $\varrho_{N_6^1}(y) < \min \tilde{y}_6^1$. It is thus the algorithm continues and checks $\tilde{y}_5^1$, which is also true. Then for $\tilde{y}_4^1$, the assertion is false, thus the algorithm splits the vector $\tilde{y}_4^1$. The same process is made until the last layer $\bar{k}$, where the bucket $\tilde{y}_5^{\bar{k}}$ has the property to stop the algorithm: $\tau_y = \varrho_{N_6^{\bar{k}}}(y)$.

## 2.2 Worst Case Complexity

Let $\lceil x \rceil$ be the function, defined from the real number to the integer number, returning the lowest integer greater than $x$. A first possible implementation of the function $h^{k+1}$, defined for the values in the interval $[\alpha = \min \tilde{y}_{b_k}^k, \beta = \max \tilde{y}_{b_k}^k]$ is :

$$h^{k+1}(x) = \left\lceil \frac{x - \alpha}{\beta - \alpha} * B \right\rceil \tag{17}$$

This function splits the interval $[\alpha, \beta]$ into $B$ intervals of equal length. Using this function, the worst-case complexity is $O(d^2)$, since at worst, each iteration removes only 2 elements.

That's why we propose another function, based on the encoding of numbers. Let $D$ be the number of binary digits used to encode the numbers of $y$. Consider that number $u$ is greater than number $v$ if the encoding of $u$ is lexicographically greater as the one of $v$, assumption which is true in nowadays computer using positive double precision values.

Consider the numbers $u = 1.6250$ and $v = 0.9375$, if we consider the digits, then $u$ is lexicographically greater than $v$. Today's computers store the number by first storing the *exponent* of the first non zero bit of the number, often using a bias for reaching negative exponent. Then, starting from the next bit on the right of the first non zero bit, the $k$ next digits are stored. These vector of $k$ bits is usually called the fraction. For negative numbers, a bit is set to 1, but we consider here only positive numbers.

Consider an exponent given using 3 bits and 4 bits for the fraction. We use here a bias of 3. Consider the numbers:

- 011-1010. First the exponent is equal to 3 minus the bias, thus 0. The represented number is $2^0 + 2^{-1} + 2^{-3} = 1 + 0.5 + 0.125 = 1.625$
- 010-1110. First the exponent is equal to 2 minus the bias, thus -1. The represented number is $2^{-1} + 2^{-2} + 2^{-3} + 2^{-4} = 0.5 + 0.25 + 0.125 + 0.0625 = 0.9375$

Using this binary encoding, the number are still lexicographically ordered.

An import remark is that, using encoded numbers over $D$ bits, the number of different numbers is finite. Moreover, let $b$ be the base of the encoding, the maximum number of comparison needed to compare two numbers $u$ and $v$ is $\lceil \log_b(D) \rceil$.

In order to define an efficient function $h$, we relax the equation (8) and thus the property ensuring that at each iteration, the number of elements of the bucket is decreasing. But we ensure that the maximal number of hierarchical iterations is finite. Here it will be $\lceil \log_b(D) \rceil$.

Let the function $E_b^k$, defined for numbers encoded over $D$ digits, be the function returning the $k$th digit in base $b$. The function $h$ is now defined as follows:

$$h^{k+1}(x) = E_b^{k+1}(x) \tag{18}$$

In today's computers, double precision values are often encoded over 64 bits. Using a base $b = 256$, thus defined over 8 binary digits (a Byte), the maximum depth of the algorithm is $log_{2^8}(2^{64}) = 8$. Moreover, the $h$ function is strongly computationally less expensive than ones using divide and multiply operations. Note that the number of buckets is equal to the base too.

Since the maximum number of operation at each iteration is bound by the number of values of $y$. The complexity of applying this methods in classical real implementation in computer is $O(d)$. Note that using a Byte for comparing number is often done while implementing efficient sorting method such as the Radix sort [3].

In the general case, using $D$ binary digits, and a a base $b$, the complexity is $O((d + b) * \log_b(D))$.

**Filtering** One of the main advantage of the method proposed by [2] is to filter, while iterating over the values, the values that are zeros in the projection. To do so, a lower bound $\tau'$ of $\tau$ is maintained and allows to remove values $y_i < \tau'$. Thus each time a value $y_i$ is considered, the algorithm adds it in its next bucket if and only if $y_i \geqslant \tau'$.

With a vector generated using Gaussian random variables, only few indexes in the projection will be non zeros. Such indexes can be fastly find with this filtering method. Let $Bucket^p$ be the bucket projection algorithm and $Bucket_f^p$ be its version having the additional filtering.

We propose for the lower bound $\tau'$, to start each time with the $\varrho$ value of the next bucket. Then for each value processed, if the value is dominated by $\tau'$, then the value is removed. Otherwise the value is used to update $\tau'$.

**Algorithm 2** $Bucket^p$

**Data**: $y, a$
$\tilde{y}^0_{b_0} \leftarrow y$
$C^0_{b_0} \leftarrow -a$
$N^0_{b_0} \leftarrow 0$
1 **for** $k \in 1..\lceil \log_b(D) \rceil$ **do**
  **for** $b \in 1..B$ **do**
    $S^k_b \leftarrow \{i \in S^{k-1}_{b_{k-1}} \mid h^{k-1}(y_i) = b\}$
    $\tilde{y}^k_b \leftarrow (y_i)_{i \in S^k_b}$
2  **for** $b \in B..1$ **do**
    $b_k \leftarrow b$
    **if** $\varrho_{N^k_{b+1}}(y) > max(\tilde{y}^k_b)$ **then**
      break loop **1**
    **if** $\varrho_{N^k_b}(y) \geqslant min(\tilde{y}^k_b)$ **then**
      break loop **2**
$\tau \leftarrow \varrho_{N^k_{b_k}}(y)$
**for** $i \in 1..|y|$ **do**
  $x_i \leftarrow \max(y_i - \tau, 0)$

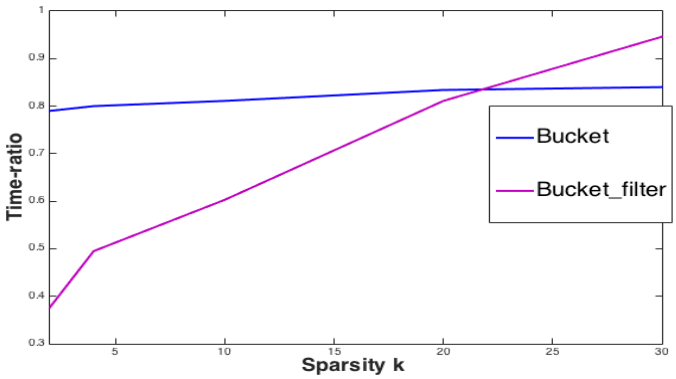


Figure 2: red: ratio between our $Bucket^p_f$ algorithm and the $Pivot_f$ algorithm, blue: ratio between our $Bucket^p$ algorithm and $Pivot_f$ algorithm. The $x$-axis is a function of the vector sparsity k.

# 3 Experimental evaluation

In order to simulate sparse data vectors generated by the splitting algorithm. See [1] for more details. We set a ratio $d/k$ of components to zeros of a random Gaussian vector (dimension $d = 1000000$). Fig. 2 shows that our hierarchical bucket filtering always outperforms the $Pivot_f$ method [2]. Moreover the improvement of our simple hierarchical bucket method is larger for sparse vectors. As a conclusion $bucket$-based method are well suited for projection on the $\ell_1$ ball inside splitting algorithms.

**Gaussian** When we use a normal distribution without assumption on the number of zeros, the running times of $Bucket^p_f$ and $Pivot_f$ are comparable and outperform all the other methods.

By profiling, we have remarked that for both algorithms more than half of the time is spent on the first iteration over $y$, and almost the other half is spent building the vector $x$, the remaining part of the algorithm is completely flooded in the global run time.

# 4 Conclusion

This paper propose a new projection algorithm based on a bucket decomposition, that allows a finer grain splitting of the values. An improvement based of the filtering principles is also given. Finally, thanks to the experimental evaluation, we have shown that our algorithms perform better in practice on sparse vectors.

# References


[1] M. Barlaud, W. Belhajali, P. L. Combettes, and L. Fillatre, "Classification and regression using an outer approximation projection-gradient method," in *To appear IEEE Signal Processing*.

[2] L. Condat, "Fast projection onto the simplex and the l1 ball," *Mathematical Programming Series A*, vol. 158, no. 1, pp. 575–585, 2016.

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press Cambridge, 2001, vol. 6.

[4] D. L. Donoho and M. Elad, "Optimally sparse representation in general (nonorthogonal) dictionaries via l minimization," *Proceedings of the National Academy of Sciences*, vol. 100, no. 5, pp. 2197–2202, 2003.

[5] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra, "Efficient projections onto the l 1-ball for learning in high dimensions," in *Proceedings of the 25th international conference on Machine learning*. ACM, 2008, pp. 272–279.

[6] M. Held, P. Wolfe, and H. P. Crowder, "Validation of subgradient optimization," *Mathematical programming*, vol. 6, no. 1, pp. 62–88, 1974.

[7] K. C. Kiwiel, "Breakpoint searching algorithms for the continuous quadratic knapsack problem," *Mathematical Programming*, vol. 112, no. 2, pp. 473–491, 2008.

[8] C. Michelot, "A finite algorithm for finding the projection of a point onto the canonical simplex of n," *Journal of Optimization Theory and Applications*, vol. 50, no. 1, pp. 195–200, 1986.

[9] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society. Series B (Methodological)*, pp. 267–288, 1996.

[10] E. Van Den Berg and M. P. Friedlander, "Probing the pareto frontier for basis pursuit solutions," *SIAM Journal on Scientific Computing*, vol. 31, no. 2, pp. 890–912, 2008.