Fast Projection onto the Simplex and the ℓ_1 Ball

Laurent Condat

v1, Aug. 15, 2014

Abstract

A new algorithm is proposed to project, exactly and in finite time, a vector of arbitrary size onto a simplex or a ℓ_1-norm ball. The algorithm is demonstrated to be faster than existing methods. In addition, a wrong statement in a paper by Duchi et al. is corrected and an adversary sequence for Michelot's algorithm is exhibited, showing that it has quadratic complexity in the worst case.

I. Introduction

The projection of a vector onto the simplex or the ℓ_1 ball appears in imaging problems, such as segmentation [1] and multispectral unmixing [2], in portfolio optimization [3], and in many applications of statistics, operations research and machine learning [4]–[6]. Given an integer $N \geq 1$, a sequence (vector) $\mathbf{y} = (y_n)_{n=1}^N \in \mathbb{R}^N$ and a real $a > 0$, we aim at computing $\mathcal{P}_\Delta(\mathbf{y}) := \arg\min_{\mathbf{x}\in\Delta} \|\mathbf{x}-\mathbf{y}\|$ or $\mathcal{P}_\mathcal{B}(\mathbf{y}) := \arg\min_{\mathbf{x}\in\mathcal{B}} \|\mathbf{x}-\mathbf{y}\|$, where the norm is the Euclidean norm, the *simplex* $\Delta \subset \mathbb{R}^N$ is defined as the set of sequences whose elements are nonnegative and sum up to a[1]:

$$\Delta := \left\{(x_1,\dots,x_N) \in \mathbb{R}^N \;\middle|\; \textstyle\sum_{n=1}^N x_n = a \text{ and } x_n \geq 0,\ \forall n = 1,\dots,N\right\} \tag{1}$$

and the ℓ_1 ball, a.k.a. cross-polytope, $\mathcal{B} \subset \mathbb{R}^N$ is defined as

$$\mathcal{B} := \left\{(x_1,\dots,x_N) \in \mathbb{R}^N \;\middle|\; \textstyle\sum_{n=1}^N |x_n| \leq a\right\}.$$

These two projections are well defined and unique, since Δ and $\mathcal{B}$ are closed and convex sets. In this paper, we focus on algorithms to perform these projections exactly and in finite time. In Sect. II, we review the methods of the literature. In Sect. III, we propose a new algorithm and we show in Sect. IV that it is faster than the existing methods.

II. Review of Prior Work

We first recall a well known property, which allows to project onto the ℓ_1 ball, as soon as one can project onto the simplex:

Proposition 1 (see, e.g., [4, Lemma 3])

$$\mathcal{P}_\mathcal{B}(\mathbf{y}) = \begin{cases} \mathbf{y}, \text{ if } \sum_{n=1}^N |y_n| \leq a, \\ \big(\mathrm{sgn}(y_1)x_1,\dots,\mathrm{sgn}(y_N)x_N\big), \text{ else,} \end{cases} \tag{2}$$

where $\mathbf{x} = \mathcal{P}_\Delta(|\mathbf{y}|)$, $|\mathbf{y}| = (|y_1|,\dots,|y_N|)$, and sgn is the signum function: if $t > 0$, $\mathrm{sgn}(t) = 1$, if $t < 0$, $\mathrm{sgn}(t) = -1$, and $\mathrm{sgn}(0) = 0$.

Thus, in the following, we focus on projecting onto the simplex only, and we denote by $\mathbf{x} := \mathcal{P}_\Delta(\mathbf{y})$ the projected sequence[2]. An important property of the projection $\mathcal{P}_\Delta$, which can be derived from the corresponding Karush-Kuhn-Tucker optimality conditions, is the following:

L. Condat is with the Dept. of Images and Signals of GIPSA-lab, University of Grenoble-Alpes, in Grenoble, France. Contact: see http://www.gipsa-lab.fr/∼laurent.condat/.

This work has been partially supported by the LabEx PERSYVAL-Lab (ANR-11-LABX-0025).

[1]For $a = 1$, Δ is called the unit, or canonical, or standard, or probability simplex.

[2]Conversely, we can remark that one can project onto the simplex using projection onto the ℓ_1 ball. Indeed, $\mathcal{P}_\Delta(\mathbf{y}) = \mathcal{P}_\Delta(\mathbf{y}+c)$, for every $c \in \mathbb{R}$, and $\mathcal{P}_\Delta(\mathbf{y}) = \mathcal{P}_\mathcal{B}(\mathbf{y})$ if the elements of $\mathbf{y}$ are nonnegative and $\|\mathbf{y}\|_1 \geq a$. Thus, whatever $\mathbf{y}$, we have $\mathcal{P}_\Delta(\mathbf{y}) = \mathcal{P}_\mathcal{B}(\mathbf{y} - y_{\min} + a/N)$, where $y_{\min}$ is the smallest element of $\mathbf{y}$.

Proposition 2 (see, e.g., [7]) There exists a unique $\tau \in \mathbb{R}$ such that

$$x_n = \max\{y_n - \tau, 0\}, \quad \forall n = 1, \ldots, N. \tag{3}$$

Therefore, the whole difficulty of the operation is to find the value of τ. Then, the projection itself simply amounts to applying the thresholding operation (3). So, we must find τ such that $\sum_{n=1}^{N} \max\{y_n - \tau, 0\} = a$. Only the largest elements of $\mathbf{y}$, which are larger than τ and are not set to zero during the projection, contribute to this sum. So, if we knew the index set $\mathcal{I} = \{n \mid x_n > 0\}$, since $\sum_{n=1}^{N} x_n = \sum_{n\in\mathcal{I}} x_n = \sum_{n\in\mathcal{I}}(y_n - \tau) = a$, we would have $\tau = (\sum_{n\in\mathcal{I}} y_n - a)/|\mathcal{I}|$. Algorithm 1, given in Fig. 1, which is based on sorting the elements of $\mathbf{y}$ in decreasing order, naturally follows from these considerations. This algorithm is explicitly given in [7] and was rediscovered many times later. Depending on the choice of the sort algorithm, the worst case complexity of Algorithm 1 is $O(N^2)$ or $O(N \log N)$ [8].

An improvement of Algorithm 1 was proposed in [5], by noticing that it is not useful to sort $\mathbf{y}$ completely, since only its largest elements are involved in the determination of τ. Thus, a heap structure can be used: a heap $\mathbf{v} = (v_1, \ldots, v_N)$ is a partially sorted sequence, such that its first element v_1 is the largest and it is fast to re-arrange $(v_2, \ldots, v_N)$ into a heap, with complexity $O(\log N)$. The complexity of arranging the elements of $\mathbf{y}$ into a heap is $O(N)$. This yields Algorithm 2, given in Fig. 1, whose complexity is $O(N + K \log N)$, where $K = |\mathcal{I}|$ is the number of nonzero elements in the solution $\mathbf{x}$.

Another way of improving Algorithm 1 is based on the following observation: it is generally considered that the fastest sorting algorithm is `quicksort`, which uses partitioning with respect to an element called pivot [8]; the pivot is chosen, and the sequence to sort is split into the two subsequences of elements smaller and larger than the pivot, which are then sorted recursively. But we can notice that τ does not depend on the ordering of the elements y_n; it depends only on the sum of the largest elements. Thus, many operations in `quicksort` are not useful for our aim and can be omitted. Indeed, let us consider, at the first iteration of the algorithm, partitioning $\mathbf{y}$ with respect to some pivot value $\rho \in [y_{\min}, y_{\max}]$, where $y_{\min}$ and $y_{\max}$ are the minimum and maximum elements of $\mathbf{y}$: we define the subsequences $\mathbf{y}_{\text{low}}$ and $\mathbf{y}_{\text{high}}$ of elements of $\mathbf{y}$ smaller and larger than ρ, respectively, and we set $S := \sum_{y\in\mathbf{y}_{\text{high}}} y - a$. Then, if $S/|\mathbf{y}_{\text{high}}| \geq \rho$, we have $\tau \geq \rho$, so that we can discard the elements of $\mathbf{y}_{\text{low}}$ and continue with $\mathbf{y}_{\text{high}}$ to determine τ. On the other hand, if $S/|\mathbf{y}_{\text{high}}| \leq \rho$, we have $\tau \leq \rho$. Thus, we can discard the elements of $\mathbf{y}_{\text{high}}$, keeping only the values S and $|\mathbf{y}_{\text{high}}|$ in memory, and continue with $\mathbf{y}_{\text{low}}$ to determine τ such that $\sum_{y\in\mathbf{y}_{\text{low}}} \max\{y - \tau, 0\} + S - |\mathbf{y}_{\text{high}}|\tau = 0$. This yields Algorithm 3, given in Fig. 1. We refer to the review paper [9] for references and discussions about this class of algorithms, which was popularized recently by the highly cited paper of Duchi et al. [4]. Before discussing the choice of the pivot, we highlight a major drawback of the algorithm given in [4], whose only difference with Algorithm 3 is that, at step 2.4., the elements of $\mathbf{v}$ equal to the pivot, except one, are left in $\mathbf{v}$ instead of being discarded. The pivot is chosen at random in $\mathbf{v}$. The worst case expected complexity (averaged over all choices of the pivot) of this algorithm is not $O(N)$ as claimed in [4], but $O(N^2)$. Indeed, expected linear time is guaranteed only if the elements of $\mathbf{y}$ are *distinct*. Since projection onto a simplex is often one operation among others in an iterative algorithm converging to a fixed point, and since sparsity of the solution is often a desirable property, it is likely that, in practice, the projection algorithm is fed with sequences in the simplex or close to it, thus containing many elements at zero. For instance, when applying the algorithm of [4] to $\mathbf{y} = (0, \ldots, 0, 1)$, the complexity is $O(N^2)$: the algorithm iterates over sequences of size N, $N - 1$, and so on until 1 is picked as pivot. This issue is corrected with Algorithm 3, which has $O(N)$ expected complexity, when the pivot is chosen at random in $\mathbf{v}$. Now, concerning the choice of the pivot, this is the same much-discussed problem as for the sort algorithm `quicksort` and the selection algorithm `quickselect`, which are based on partitioning as well [8]. The choice depends on whether one wants a deterministic algorithm or one is ready to make use of a random number generator and accept a fluctuating running time. It also depends on whether one is ready to accept the worst case $O(N^2)$, which may come randomly with low probability or may be deliberately triggered by someone having knowledge of the implementation and feeding the algorithm with a contrived sequence $\mathbf{y}$, creating a security risk. Choosing the pivot at random in the list $\mathbf{v}$ gives expected complexity $O(N)$, with some variance, but worst case complexity $O(N^2)$. If the pivot is the median of $\mathbf{v}$, the complexity becomes $O(N)$, which is optimal, but a linear time median finding routine, typically based on the median of medians [11], is cumbersome to implement and slow. According to [9], a good compromise, which we adopt in Sect. IV, is to take the median

Algorithm 1 (using sorting) [7]
1. Sort $\mathbf{y}$ into $\mathbf{u}$: $u_1 \geq \cdots \geq u_N$.
2. Set $K := \max_{1 \leq k \leq N} \{k \mid (\sum_{r=1}^{k} u_r - a)/k < u_k\}$.
3. Set $\tau := (\sum_{k=1}^{K} u_k - a)/K$.
4. For $n = 1, \ldots, N$, set $x_n := \max\{y_n - \tau, 0\}$.

Algorithm 2 (using a heap) [5]
1. Build a heap $\mathbf{v}$ from $\mathbf{y}$.
2. For $k = 1, \ldots, N$, do
   - 2.1. Set $u_k := v_1$ (largest element of the heap).
   - 2.2. If $(\sum_{r=1}^{k} u_r - a)/k \geq u_k$, exit the loop.
   - 2.3. Set $K := k$.
   - 2.4. Remove v_1 from $\mathbf{v}$ and re-heapify $\mathbf{v}$.
3. Set $\tau := (\sum_{k=1}^{K} u_k - a)/K$.
4. For $n = 1, \ldots, N$, set $x_n := \max\{y_n - \tau, 0\}$.

Algorithm 3 (using partitioning) [9].
1. Set $\mathbf{v} := \mathbf{y}$, $K := 0$, $S := -a$.
2. While $\mathbf{v}$ is not empty, do
   - 2.1. Choose a pivot ρ in the convex hull of $\mathbf{v}$.
   - 2.2. Construct the subsequences $\mathbf{y}_{\text{high}} := (y \in \mathbf{v} \mid y > \rho)$, $\mathbf{y}_{\text{low}} := (y \in \mathbf{v} \mid y < \rho)$, and set M as the number of elements equal to ρ in $\mathbf{v}$.
   - 2.3. If $(S + M\rho + \sum_{y \in \mathbf{y}_{\text{high}}} y)/(K + M + |\mathbf{y}_{\text{high}}|) < \rho$, set $S := S + M\rho + \sum_{y \in \mathbf{y}_{\text{high}}} y$, $K := K + M + |\mathbf{y}_{\text{high}}|$, $\mathbf{v} := \mathbf{y}_{\text{low}}$.
   - 2.4. Else, set $\mathbf{v} := \mathbf{y}_{\text{high}}$.
3. Set $\tau := S/K$.
4. For $n = 1, \ldots, N$, set $x_n := \max\{y_n - \tau, 0\}$.

Algorithm 4 (active set method of Michelot) [10].
1. Set $\mathbf{v} := \mathbf{y}$, $\rho := (\sum_{n=1}^{N} y_n - a)/N$.
2. Do, while $|\mathbf{v}|$ changes,
   - 2.1. Replace $\mathbf{v}$ by its subsequence $(y \in \mathbf{v} \mid y > \rho)$.
   - 2.2. Set $\rho := (\sum_{y \in \mathbf{v}} y - a)/|\mathbf{v}|$.
3. Set $\tau := \rho$, $K = |\mathbf{v}|$.
4. For $n = 1, \ldots, N$, set $x_n := \max\{y_n - \tau, 0\}$.

Fig. 1. Several algorithms to project onto the simplex Δ. The input data consists in $N \geq 1$, $\mathbf{y} \in \mathbb{R}^N$, $a > 0$, and the ouput is the sequence $\mathbf{x} = (x_n)_{n=1}^N = \mathcal{P}_\Delta(\mathbf{y})$. Incidentally, the number K at the end of the algorithms is the number of nonzero elements in $\mathbf{x}$.

of $\mathbf{v}$ as pivot, but to find it using the efficient algorithm of [12], whose expected complexity (not worst case) in terms of comparisons is $3N/2 + o(N)$.

A different algorithm has been proposed by Michelot [10]. It is reproduced[3] as Algorithm 4 in Fig. 1. It can be viewed as a version of Algorithm 3, where the pivot ρ would alway be known to be a lower bound of τ, so that the step 2.4. is always executed. Indeed, for every subsequence $\mathbf{v}$ of $\mathbf{y}$, by setting $\rho = (\sum_{y \in \mathbf{v}} - a)/|\mathbf{v}|$, we have $a = \sum_{y \in \mathbf{v}} (y - \rho) \leq \sum_{y \in \mathbf{v}} \max\{y - \rho, 0\} \leq \sum_{n=1}^{N} \max\{y_n - \rho, 0\}$. Therefore, $\rho \leq \tau$. Consequently, if $y \leq \rho$, we know that $\max\{y - \tau, 0\} = 0$ and we can discard the inactive variable y, which does not contribute

[3]Actually, Algorithm 4 is an improvement of Michelot's algorithm, with the test "$>$" instead of "$\geq$" at step 2.1. This modification has been proposed in [13, Sect. 5.7]. Algorithm 4 is also the same algorithm as in [14].

Proposed Algorithm

1. Set $\mathbf{v} := (y_1)$, $\tilde{\mathbf{v}}$ as an empty list, $\rho := y_1 - a$.
2. For $n = 2, \ldots, N$, do
   - If $y_n > \rho$,
     - 2.1. Set $\rho := \rho + (y_n - \rho)/(|\mathbf{v}| + 1)$.
     - 2.2. If $\rho > y_n - a$, add y_n to $\mathbf{v}$.
     - 2.3. Else, add $\mathbf{v}$ to $\tilde{\mathbf{v}}$, set $\mathbf{v} := (y_n)$, $\rho := y_n - a$.
3. If $\tilde{\mathbf{v}}$ is not empty, for every element y of $\tilde{\mathbf{v}}$, do
   - 3.1. If $y > \rho$, add y to $\mathbf{v}$ and set $\rho := \rho + (y - \rho)/|\mathbf{v}|$.
4. Do, while $|\mathbf{v}|$ changes,
   - For every element y of $\mathbf{v}$, do
     - 4.1. If $y \leq \rho$, remove y from $\mathbf{v}$ and set $\rho := \rho + (\rho - y)/|\mathbf{v}|$.
5. Set $\tau := \rho$, $K = |\mathbf{v}|$.
6. For $n = 1, \ldots, N$, set $x_n := \max\{y_n - \tau, 0\}$.

Fig. 2. Proposed algorithm to project onto the simplex Δ. The input data consists in $N \geq 1$, $\mathbf{y} \in \mathbb{R}^N$, $a > 0$, and the ouput is the sequence $\mathbf{x} = (x_n)_{n=1}^N = \mathcal{P}_\Delta(\mathbf{y})$. Incidentally, the number K at the end of the algorithm is the number of nonzero elements in $\mathbf{x}$.

to the determination of τ. By alternating between the calculation of $\rho = (\sum_{y \in \mathbf{v}} y - a)/|\mathbf{v}|$ and the update of the sequence $\mathbf{v}$ by discarding its elements smaller or equal to ρ, the algorithm enters a steady state after a finite number of iterations (consisting of steps 2.1. and 2.2.), with $\rho = \tau$. Algorithm 4 has several advantages: it is deterministic, very simple to implement, and independent of the ordering of the elements in $\mathbf{y}$. Its complexity is observed linear in practice [13]. Yet, its worst case complexity is $O(N^2)$; this corresponds to the case where only one element is discarded from $\mathbf{v}$ at step 2.1. of every iteration. Let us exhibit, for the first time to the author's knowledge, an example of such a worst case. The construction of this sequence $\mathbf{y}$, of size $N \geq 4$, is recursive and as follows. Let ε be a small positive real. We set $y_1 := a$, $y_2 := -\varepsilon$ and, for every $n = 3, \ldots, N$, $y_n := (n-1)y_{n-1} - \varepsilon - \sum_{r=2}^{n-2} y_r$. Then it can be shown easily by induction that, at the end of the i-th iteration, for $1 \leq i \leq N-3$, we have $\mathbf{v} = (y_1, \ldots, y_{N-i})$, $\rho = (\sum_{y \in \mathbf{v}} y - a)/|\mathbf{v}| = (\sum_{n=1}^{N-i} y_n - a)/(N-i) = y_{N-i-1} - \varepsilon/(N-i)$, and $y_1 > \cdots > y_{N-i-1} > \rho > y_{N-i} > \cdots > y_N$. The algorithm terminates after N iterations, with $\mathbf{v} = (a)$ and $\tau = 0$. Thus, the complexity is $O(\sum_{n=1}^N n) = O(N^2)$. We note, however, that such a pathological sequence, with elements growing exponentially, is completely unlikely to be met in practical applications.

Yet another way to look at the problem is to view the search of τ as a root finding problem [14], [15]. Let us define the function $f : \rho \mapsto \sum_{n=1}^N \max\{y_n - \rho, 0\} - a$. We look for τ such that $f(\tau) = 0$, so τ is a root of f. f has the following properties: it is convex; it is piecewise linear with breakpoints at the y_n; it is strictly decreasing on $(-\infty, y_{\max}]$ and $f(\rho) = -a$ for every $\rho \in [y_{\max}, +\infty)$. So, for any $\rho \in \mathbb{R}$, if $f(\rho) < 0$, then $\rho > \tau$ and if $f(\rho) > 0$, then $\rho < \tau$. Moreover, $f(y_{\min} - a/N) \geq 0$, $f(y_{\max} - a/N) \leq 0$ and $f(y_{\max} - a) \geq 0$, so that $\tau \in [\max\{y_{\max} - a, y_{\min} - a/N\}, y_{\max} - a/N]$. Thus, Algorithm 3 may be interpreted as a bracketing method and Algorithm 4 as a Newton method [16, Proposition 1] to find the root τ. The method of [15] combines features from the bisection method and the secant method. However, the proof of [15] that the bisection method has complexity $O(N)$ is not valid: for a fixed, arbitrarily small, value $\delta > 0$, the number of elements y_n in an interval of size δ bracketing τ may be as large as N, so that the number of bisection steps, each of complexity $O(N)$, may be arbitrarily large. Finally, we note that projection onto the simplex is a particular case of the more general *continuous quadratic knapsack problem*; most methods proposed to solve this problem are based on the principles discussed in this section [9], [13] and we refer to the survey paper [6] for a complete annotated list of references.

III. Proposed Algorithm

Using the principles seen in the previous section, we are in position to explain the proposed algorithm, given in Fig. 2, which can be viewed as a Gauss–Seidel-like variation of Algorithm 4. Indeed, the lower bound ρ of τ can be updated not only after a complete scan of the active sequence $\mathbf{v}$, but after *every* element of $\mathbf{v}$ is read. Let us first

TABLE I
COMPLEXITY OF THE ALGORITHMS TO PROJECT ONTO THE SIMPLEX, WITH RESPECT TO THE LENGTH N OF THE DATA AND NUMBER K OF NONZERO ELEMENTS IN THE SOLUTION. FOR ALGORITHM 1, QUICKSORT AND A RANDOM PIVOT ARE ASSUMED. FOR ALGORITHM 3 WITH THE MEDIAN PIVOT, A $O(N)$ MEDIAN FINDING ROUTINE IS ASSUMED.

	worst case complexity	expected complexity	observed in practice
Algorithm 1	$O(N^2)$	$O(N \log N)$	$O(N \log N)$
Algorithm 2	$O(N+K\log N)$	—	$O(N+K\log N)$
Alg. 3, random pivot	$O(N^2)$	$O(N)$	$O(N)$
Alg. 3, median pivot	$O(N)$	—	$O(N)$
Algorithm 4	$O(N^2)$	—	$O(N)$
Proposed Algorithm	$O(N^2)$	—	$O(N)$

describe a simplified version of the algorithm, without the step 3. and with the steps 2.1., 2.2. and 2.3. replaced by "2.1. Add y_n to $\mathbf{v}$ and set $\rho := \rho + (y_n - \rho)/|\mathbf{v}|$."

The algorithm starts with the first pass (steps 1. and 2.), which does not assume any knowledge about $\mathbf{y}$. Let us consider that we are currently reading the element y_n, for some $2 \leq n \leq N$ and that we have already read the previous elements y_r, $r = 1, \ldots, n-1$. We have a subsequence $\mathbf{v}$ of $(y_r)_{r=1}^{n-1}$ of all the elements potentially larger than τ and we maintain the variable $\rho = (\sum_{y \in \mathbf{v}} y - a)/|\mathbf{v}|$. We know that $\rho \leq \tau$. Hence, if $y_n \leq \rho$, then $y_n \leq \tau$. So, we can ignore this element and we do nothing. In the other case $y_n > \rho$, we add y_n to $\mathbf{v}$, since y_n is potentially larger than τ, and ρ is assigned the new value of $(\sum_{y \in \mathbf{v}} y - a)/|\mathbf{v}|$, which is strictly larger than previously. Then we continue the pass with the next element y_{n+1}. The pass is initialized with $\mathbf{v} = (y_1)$ and $\rho = y_1 - a$.

At the beginning of all the subsequent passes (step 4.), we have the subsequence $\mathbf{v}$ of all the elements of $\mathbf{y}$ potentially larger than τ. The difference with the beginning of the first pass is that we have calculated the value $\rho = (\sum_{y \in \mathbf{v}} y - a)/|\mathbf{v}|$; we will use it to remove elements from $\mathbf{v}$ sequentially. Let us consider that we are reading the element $v \in \mathbf{v}$. If $v > \rho$, we do nothing. Else, $v \leq \tau$, so we remove this element from $\mathbf{v}$. Consequently, ρ is assigned the new value of $(\sum_{y \in \mathbf{v}} y - a)/|\mathbf{v}|$, which is strictly larger than previously. Then we continue the pass with the next element of $\mathbf{v}$.

The proof of correctness of this algorithm is straightforward. At the end of every pass, either at least one element has been removed from $\mathbf{v}$, or $\mathbf{v}$ and ρ remain the same as after the previous pass. In the latter case, the elements of $\mathbf{y}$ which are not present in $\mathbf{v}$ are smaller than ρ and the elements in $\mathbf{v}$ (which are the $|\mathbf{v}|$ largest elements of $\mathbf{y}$) are larger than ρ, so that $a = \sum_{y \in \mathbf{v}} (y - \rho) = \sum_{n=1}^{N} \max\{y_n - \rho, 0\}$. Thus, from Proposition 2, $\rho = \tau$.

The first pass of the proposed algorithm, as given in Fig. 2, contains a refinement with respect to the algorithm just described. Every pass aims at calculating the best possible lower bound ρ of τ. And for every n, $y_n - a \leq \tau$. So, when reading y_n, if $y_n - a$ is larger than the current value of ρ, we set $\rho := y_n - a$. But then a cleanup pass (step 3.) is necessary after the first pass, to restore the invariant properties that $\rho = (\sum_{y \in \mathbf{v}} y - a)/|\mathbf{v}|$ and that $\mathbf{v}$ contains all the elements of $\mathbf{y}$ larger than ρ.

We end this section with a few comments on the complexity of the proposed algorithm. It is always faster than Algorithm 4, since after every pass, more elements of $\mathbf{v}$ are removed. Contrary to Algorithm 4, its complexity depends on the ordering of the elements in $\mathbf{y}$. In the most favorable case where $\mathbf{y}$ is sorted in decreasing order, the complexity is $O(N)$, since at the end of the first pass, which behaves like step 2. of Algorithm 1, we have $\rho = \tau$. The complexity is also $O(N)$ if $\mathbf{y}$ is sorted in increasing order, since $\rho = \tau$ after the second pass. However, the worst case complexity is $O(N^2)$: if $\mathbf{y}$ is the adversary sequence for Algorithm 4 shown in the previous section, with a parameter ε sufficiently small, then $\tilde{\mathbf{y}} = (y_{N-1}, a/2, a/2, y_2, \ldots, y_{N-2})$ is an adversary sequence for the proposed algorithm.

IV. COMPARISON OF ALGORITHMS

The complexity of the algorithms is summarized in Tab. I. In Tab. II, for one example, the number of elements not yet characterized as active or inactive is shown, after every pass of the iterative algorithms. This demonstrates the efficiency of the selection process of the proposed algorithm. All the algorithms were implemented in C and quite optimized. The code is freely available on the website of the author. The code was run on a Apple Macbook pro laptop with a 2.3Ghz Intel Core i7 CPU. The computation times for several experiments, reported in Tab. III,

TABLE II

NUMBER $|\mathbf{v}|$ OF ELEMENTS IN THE SEQUENCE $\mathbf{v}$ AFTER EVERY PASS OF THE ALGORITHMS, FOR ONE EXAMPLE WITH $N = 10^6$; $\mathbf{y}$ HAS I.I.D. GAUSSIAN RANDOM ELEMENTS OF MEAN $1/N$ AND STD. DEV. 0.1.

Pass number	Algorithm 3 random pivot	Algorithm 3 median pivot	Algorithm 4	Proposed Algorithm
1	575147	499999	499787	8145
2	85926	249999	212149	1622
3	15811	124999	85994	359
4	2049	62499	33840	107
5	2013	31249	13189	56
6	997	15624	5123	53
7	709	7811	1993	53
8	435	3905	785	
9	26	1952	306	
10	14	975	133	
11	12	487	71	
12	7	243	55	
13	5	121	53	
14	2	60	53	
15	0	30		
16		15		
17		7		
18		3		
19		1		
20		0		

TABLE III

COMPUTATION TIMES IN SECONDS FOR DIFFERENT EXPERIMENTS, AVERAGED OVER 10^2 (FOR $N = 10^6$) AND 10^4 (FOR $N = 10^3$ AND $N = 20$) REALIZATIONS (THE NUMBER IN PARENTHESES IS THE STD. DEV.). EXPERIMENTS 1 AND 2 CORRESPOND TO THE y_n BEING I.I.D. RANDOM GAUSSIAN NUMBERS OF MEAN a/N AND STD. DEV. 1 AND 10^{-3}, RESPECTIVELY. EXPERIMENT 3 CORRESPONDS TO THE y_n BEING I.I.D. RANDOM GAUSSIAN NUMBERS OF MEAN 0 AND STD. DEV. 10^{-3}, EXCEPT ONE ELEMENT AT A RANDOM POSITION, WHICH IS A RANDOM GAUSSIAN NUMBER OF MEAN a AND STD. DEV. 10^{-3}. EXPERIMENT 4 CORRESPONDS TO THE y_n BEING 0, EXCEPT ONE ELEMENT EQUAL TO a AT A RANDOM POSITION. SO, $\mathbf{y}$ IS NOT ON THE SIMPLEX FOR EXPERIMENTS 1–3, AND ON THE SIMPLEX FOR EXPERIMENT 4. IN ALL CASES, $a = 1$ AND THE BEST TIME IS IN BOLD.

	Experiment 1			Experiment 2		
	$N=10^6$ ($K\approx 6$)	$N=10^3$ ($K\approx 4$)	$N=20$ ($K\approx 3$)	$N=10^6$ ($K\approx 3282$)	$N=10^3$ ($K\approx 816$)	$N=20$ ($K\approx 20$)
Algorithm 1	1.1e-1 (8e-4)	7.4e-5 (5e-6)	1.4e-6 (6e-7)	1.1e-1 (1e0)	8.0e-5 (5e-6)	1.6e-6 (7e-7)
Algorithm 2	1.1e-2 (6e-4)	9.1e-6 (1e-6)	6.6e-7 (7e-7)	1.2e-2 (9e-5)	5.6e-5 (4e0)	9.7e-7 (7e-7)
Alg. of Duchi et al.	1.0e-2 (5e-3)	1.1e-5 (6e-6)	8.8e-7 (8e-7)	1.3e-2 (7e-3)	1.6e-5 (5e-6)	7.7e-7 (7e-7)
Alg. 3, random pivot	1.0e-2 (5e-3)	1.1e-5 (6e-6)	9.5e-7 (7e-7)	1.3e-2 (6e-3)	1.6e-5 (4e-6)	8.2e-7 (7e-7)
Alg. 3, median pivot	2.1e-2 (4e-4)	2.7e-5 (3e-6)	9.9e-7 (7e-7)	2.1e-2 (8e-4)	2.5e-5 (2e-6)	1.0e-6 (7e-7)
Algorithm 4	1.8e-2 (7e-4)	1.8e-5 (6e-6)	8.2e-7 (7e-7)	1.8e-2 (2e-4)	3.2e-5 (3e-6)	6.3e-7 (7e-7)
Proposed Algorithm	**1.8e-3** (2e-5)	**1.8e-6** (7e-7)	**5.5e-7** (7e-7)	**3.7e-3** (5e-5)	**1.5e-5** (1e-6)	**6.1e-7** (7e-7)

	Experiment 3			Experiment 4		
	$N=10^6$ ($K\approx 21$)	$N=10^3$ ($K\approx 9$)	$N=20$ ($K\approx 4$)	$N=10^6$ ($K=1$)	$N=10^3$ ($K=1$)	$N=20$ ($K=1$)
Algorithm 1	1.1e-1 (2e-3)	7.4e-5 (5e-6)	1.4e-6 (6e-7)	2.9e-2 (5e-4)	1.8e-5 (2e-6)	8.8e-7 (7e-7)
Algorithm 2	1.1e-2 (3e-4)	9.6e-6 (1e-6)	7.0e-7 (7e-7)	3.1e-3 (2e-4)	**2.1e-6** (8e-7)	5.5e-7 (6e-7)
Alg. of Duchi et al.	1.0e-2 (6e-3)	1.2e-5 (6e-6)	8.9e-7 (7e-7)	1.1e+2 (4e+2) (!)	2.1e-5 (2e-4)	1.0e-6 (9e-7)
Alg. 3, random pivot	1.0e-2 (5e-3)	1.1e-5 (6e-6)	9.6e-7 (7e-7)	**2.8e-3** (1e-4)	2.4e-6 (7e-7)	**5.2e-7** (6e-7)
Alg. 3, median pivot	2.1e-2 (4e-4)	2.7e-5 (2e-6)	9.9e-7 (7e-7)	1.4e-2 (5e-4)	1.3e-5 (2e-6)	8.0e-7 (7e-7)
Algorithm 4	1.8e-2 (2e-4)	1.8e-5 (2e-6)	7.9e-7 (7e-7)	1.6e-2 (2e-3)	1.8e-5 (1e-6)	7.6e-7 (7e-7)
Proposed Algorithm	**3.5e-3** (2e-4)	**1.1e-5** (6e-6)	**6.9e-7** (7e-7)	7.4e-3 (3e-3)	6.9e-6 (3e-6)	6.3e-7 (7e-7)

show that the proposed algorithm performs the best, except in the particular case, of limited practical interest, where $\mathbf{y}$ is maximally sparse and exactly on the simplex.

V. CONCLUSION

We have provided a synthetic overview of the available algorithms to project onto the simplex or the ℓ_1 ball and we have proposed a new and faster algorithm. Future work will be focused on more complicated optimization problems, like minimizing a cost function over this kind of polyhedral sets.

REFERENCES

[1] J. Lellmann, J. H. Kappes, J. Yuan, F. Becker, and C. Schnörr, "Convex multi-class image labeling by simplex-constrained total variation," in *Proc. of Scale Space and Variational Methods in Computer Vision (SSVM)*, vol. 5567, 2009, pp. 150–162.

[2] J. M. Bioucas-Dias, A. Plaza, N. Dobigeon, M. Parente, Q. Du, P. Gader, and J. Chanussot, "Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches," *IEEE J. Sel. Topics Appl. Earth Observations Remote Sens.*, vol. 5, no. 2, pp. 354–379, 2012.

[3] J. Brodie, I. Daubechies, C. D. Mol, D. Giannone, and I. Loris, "Sparse and stable Markowitz portfolios," *Proc. Nat. Acad. Sci.*, vol. 106, no. 30, pp. 12 267–12 272, 2009.

[4] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra, "Efficient projections onto the ℓ_1-ball for learning in high dimensions," in *Proc. of the 25th Int. Conf. on Machine learning (ICML)*, 2008.

[5] E. van den Berg and M. P. Friedlander, "Probing the Pareto frontier for basis pursuit solutions," *SIAM J. Sci. Comput.*, vol. 31, pp. 890–912, Nov. 2008.

[6] M. Patriksson, "A survey on the continuous nonlinear resource allocation problem," *European Journal of Operational Research*, vol. 185, pp. 1–46, 2008.

[7] M. Held, P. Wolfe, and H. Crowder, "Validation of subgradient optimization," *Mathematical Programming*, vol. 6, pp. 62–88, 1974.

[8] J. L. Bentley and M. D. McIlroy, "Engineering a sort function," *Software—Practice & Experience*, vol. 23, no. 11, pp. 1249–1265, Nov. 1993.

[9] K. C. Kiwiel, "Breakpoint searching algorithms for the continuous quadratic knapsack problem," *Math. Program., Ser. A*, vol. 112, pp. 473–491, 2008.

[10] C. Michelot, "A finite algorithm for finding the projection of a point onto the canonical simplex of $\mathbb{R}^n$," *J. Optim. Theory Appl.*, vol. 50, no. 1, pp. 195–200, Jul. 1986.

[11] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan, "Time bounds for selection," *J. Computer and System Sciences*, vol. 7, no. 4, pp. 448–461, Aug. 1973.

[12] K. C. Kiwiel, "On Floyd and Rivest's SELECT algorithm," *Theoretical Computer Science*, vol. 347, pp. 214–238, 2005.

[13] ——, "Variable fixing algorithms for the continuous quadratic knapsack problem," *J. Optim. Theory Appl.*, vol. 136, pp. 445–458, 2008.

[14] P. Gong, K. Gai, and C. Zhang, "Efficient Euclidean projections via piecewise root finding and its application in gradient projection," *Neurocomputing*, vol. 74, pp. 2754–2766, 2011.

[15] J. Liu and J. Ye, "Efficient Euclidean projections in linear time," in *Proc. of the 26th Int. Conf. Machine Learning (ICML)*, 2009.

[16] R. Cominetti, W. F. Mascarenhas, and P. J. S. Silva, "A Newton's method for the continuous quadratic knapsack problem," *Math. Prog. Comp.*, vol. 6, pp. 151–169, 2014.