Math. Program., Ser. A

Fast Projection onto the Simplex and the ℓ_1 Ball

Laurent Condat

Final author's version. The final publication is available at Springer via http://dx.doi.org/10.1007/s10107-015-0946-6

Abstract A new algorithm is proposed to project, exactly and in finite time, a vector of arbitrary size onto a simplex or an ℓ_1-norm ball. It can be viewed as a Gauss-Seidel-like variant of Michelot's variable fixing algorithm; that is, the threshold used to fix the variables is updated after each element is read, instead of waiting for a full reading pass over the list of non-fixed elements. This algorithm is empirically demonstrated to be faster than existing methods.

Keywords Simplex · ℓ_1-norm ball · Large-scale optimization

Mathematics Subject Classification (2010) 49M30, 65C60, 65K05, 90C25.

1 Introduction

The projection of a vector onto the simplex or the ℓ_1 ball appears in imaging problems, such as segmentation [15] and multispectral unmixing [2], in portfolio optimization [5], and in many applications of statistics, operations research and machine learning [7,20,3,18,19]. 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

$$\mathscr{P}_\Delta(\mathbf{y}) := \operatorname*{arg\,min}_{\mathbf{x}\in\Delta} \|\mathbf{x}-\mathbf{y}\| \tag{1}$$

or

$$\mathscr{P}_\mathscr{B}(\mathbf{y}) := \operatorname*{arg\,min}_{\mathbf{x}\in\mathscr{B}} \|\mathbf{x}-\mathbf{y}\|, \tag{2}$$

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 (for $a = 1$, Δ is called the unit, or canonical, or standard, or probability simplex):

$$\Delta := \big\{(x_1,\dots,x_N) \in \mathbb{R}^N \;\big|\; \textstyle\sum_{n=1}^N x_n = a \\ \text{and } \; x_n \geq 0, \; \forall n = 1,\dots,N\big\} \tag{3}$$

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

L. Condat is with Univ. Grenoble Alpes, GIPSA-Lab, F-38000 Grenoble, France.
Contact: see http://www.gipsa-lab.fr/∼laurent.condat/

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

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

These two projections are well defined and unique, since Δ and $\mathscr{B}$ are closed and convex sets. In this paper, we focus on algorithms to perform these projections exactly and in finite time. In Sect. 2, we review the methods of the literature. In Sect. 3, we propose a new algorithm and we show in Sect. 4 that it is faster than the existing methods.

2 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 2.1 (see, e.g., [7, Lemma 3])

$$\mathscr{P}_{\mathscr{B}}(\mathbf{y}) = \begin{cases} \mathbf{y}, \text{ if } \sum_{n=1}^N |y_n| \leq a, \\ \big(\mathrm{sgn}(y_1)x_1,\dots,\mathrm{sgn}(y_N)x_N\big), \text{ else,} \end{cases} \tag{4}$$

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

Remark 2.1 Conversely to Proposition 2.1, one can project onto the simplex using projection onto the ℓ_1 ball. Indeed, $\mathscr{P}_{\Delta}(\mathbf{y}) = \mathscr{P}_{\Delta}(\mathbf{y}+c)$, for every $c \in \mathbb{R}$, and $\mathscr{P}_{\Delta}(\mathbf{y}) = \mathscr{P}_{\mathscr{B}}(\mathbf{y})$ if the elements of $\mathbf{y}$ are nonnegative and $\|\mathbf{y}\|_1 \geq a$. Thus, whatever $\mathbf{y}$, we have $\mathscr{P}_{\Delta}(\mathbf{y}) = \mathscr{P}_{\mathscr{B}}(\mathbf{y} - y_{\min} + a/N)$, where $y_{\min}$ is the smallest element of $\mathbf{y}$.

So, by virtue of Proposition 2.1, we focus in the following on projecting onto the simplex only, and we denote by

$$\mathbf{x} := \mathscr{P}_{\Delta}(\mathbf{y}) \tag{5}$$

the projected sequence. An important property of the projection $\mathscr{P}_{\Delta}$, which can be derived from the corresponding Karush-Kuhn-Tucker optimality conditions, is the following:

Proposition 2.2 (see, e.g., [10]) *There exists a unique* $\tau \in \mathbb{R}$ *such that*

$$x_n = \max\{y_n - \tau, 0\}, \quad \forall n = 1,\dots,N. \tag{6}$$

Therefore, the whole difficulty of the operation $\mathscr{P}_{\Delta}$ is to find the value of τ. Then, the projection itself simply amounts to applying the thresholding operation (6). 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 $\mathscr{I} = \{n \mid x_n > 0\}$, since $\sum_{n=1}^N x_n = \sum_{n\in\mathscr{I}} x_n = \sum_{n\in\mathscr{I}} (y_n - \tau) = a$, we would have $\tau = (\sum_{n\in\mathscr{I}} y_n - a)/|\mathscr{I}|$. Algorithm 1, given in

Algorithm 1 (using sorting) [10]
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) [20]
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) [12].
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) [17].
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 $\mathbf{x} = (x_n)_{n=1}^N = \mathscr{P}_\Delta(\mathbf{y})$. Incidentally, the number K at the end of the algorithms is the number of nonzero elements in $\mathbf{x}$.

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 [10] and was rediscovered many times later. Depending on the choice of the sorting algorithm, the worst case complexity of Algorithm 1 is $O(N^2)$ or $O(N \log N)$ [1].

An improvement of Algorithm 1 was proposed in [20], 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 $K = |\mathscr{I}|$ 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 [1]; 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 [12] for references and discussions about this class of algorithms, which was popularized recently by the highly cited paper of Duchi et al. [7]. Before discussing the choice of the pivot, we highlight a major drawback of the algorithm given in [7], 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 [7], 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 [7] 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 our Algorithm 3, which has $O(N)$ expected complexity when the pivot is chosen at random in $\mathbf{v}$, thanks to our careful handling of the elements equal to the pivot.

Now, concerning the choice of the pivot, this is the same much-discussed problem as for the sorting algorithm `quicksort` and the selection algorithm `quickselect`, which are based on partitioning as well [1]. The choice depends on whether 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 [4], is cumbersome to implement and slow. According to [12], a good compromise, which we adopt in Sect. 4, is to take the median of $\mathbf{v}$ as pivot, but to find it

using the efficient algorithm of [11], whose expected complexity (not worst case) in terms of comparisons is $3N/2 + o(N)$.

A different algorithm, of the variable-fixing type [13,18,19], has been proposed by Michelot [17]. It is reproduced[1] 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 element y, which does not contribute 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. Examples of such pathological sequences can be easily constructed; one example is given in [6, end of Section 3].

Yet another way to look at the problem is to view the search of τ as a root finding problem [16,9]. 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 [6, Proposition 1] to find the root τ. The method of [16] combines features from the bisection method and the secant method. However, the proof of [16] 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 [12,13] and we refer to the survey papers [18,19] for a complete annotated list of references.

3 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

[1] 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 [9].

Proposed Algorithm

1. Set $\mathbf{v} := (y_1)$, $\tilde{\mathbf{v}}$ as an empty list, $\rho := y_1 - a$.
2. For $n = 2, \dots, N$, 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 $n = 1, \dots, N$, 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 $\mathbf{x} = (x_n)_{n=1}^N = \mathscr{P}_\Delta(\mathbf{y})$. Incidentally, the number K at the end of the algorithm is the number of nonzero elements in $\mathbf{x}$.

variation of Algorithm 4. Indeed, the lower bound ρ of τ can be updated not only after a complete scan of the sequence $\mathbf{v}$, but after *every* element of $\mathbf{v}$ is read. Let us first 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, $r = 1, \dots, n-1$. 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 $y \in \mathbf{v}$. If $y > \rho$, we do nothing. Else, $y \leq \tau$, 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

Table 1 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 median finding routine with worst case complexity $O(N)$ (and not $O(N^2)$ like in the implementation evaluated in Section 4) 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	$O(N^2)$	—	$O(N)$

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 $O(N^2)$; like for Algorithm 4, an adversary sequence can be easily constructed.

4 Comparison of the algorithms

The complexity of the algorithms is summarized in Tab. 1. In Tab. 2, 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. Attention was paid to the numerical robustness as well, with all the averaged quantities like $(\sum_{y\in\mathbf{v}} y - a)/|\mathbf{v}|$ computed using the Welford–Knuth running mean algorithm [14]. The code is freely available on the website of the author. Note that we implemented the algorithms to maximize the execution speed, without taking care of the memory usage. For instance, we implemented Algorithm 3 with two auxiliary buffers of size N, to store the elements lower and greater than the pivot. Using only one buffer of size N would be possible, performing partitioning using swaps to put the elements lower and greater than the pivot in the first and second part of the buffer, respectively (see the description of Program 6 in [1] for details). This would slow down the algorithm, however. The proposed algorithm only requires one memory buffer of size N to store the sequence $\mathbf{v}$, which is updated in place. As a parenthesis, we remark that one could easily modify Algorithm 4 and the proposed algorithm, so that they do not require any auxiliary memory buffer, except the one to store $\mathbf{y}$, replaced by $\mathbf{x}$ in place at the end

Table 2 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		

of the algorithm; this would require reading the entire sequence $\mathbf{y}$ at every pass of the algorithm.

When implementing projection onto the ℓ_1 ball, according to Proposition 2.1, the naive approach consists in doing a first pass to compute $|\mathbf{y}|$ and to decide whether $\mathbf{y}$ is inside the ℓ_1 ball. With the proposed algorithm, this pass can be fused with its first pass (steps 1. and 2.), replacing y_n by $|y_n|$ everywhere. Then, after step 2., a simple test is performed: if $\rho \leq 0$, $\mathbf{y}$ is inside the ℓ_1 ball, so the algorithm sets $\mathbf{x} = \mathbf{y}$ and terminates. Else, $\mathbf{y}$ is outside the ℓ_1 ball and the algorithm continues with step 3.; the signs of the y_n are applied to the x_n at step 6.

The code was run on a Apple Macbook Pro laptop under OS 10.9.4 with a 2.3Ghz Intel Core i7 CPU and 8Go RAM. The computation times for projecting onto the simplex, reported in Tab. 3, show that the proposed algorithm performs globally the best, except in the particular case, of limited practical interest, where $\mathbf{y}$ is maximally sparse and exactly on the simplex. In Tab. 4, for projection onto the ℓ_1 ball, the proposed algorithm is compared to the improved bisection algorithm (IBIS) of Liu and Ye [16]; we used the C code of the authors (function `eplb` of their package SLEP), available online at `http://www.public.asu.edu/~jye02/Software/SLEP/`. As a result, the proposed algorithm is more than twice faster than IBIS for large N.

5 Concluding remarks

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. In some practical applications, the vector to project is of small length N and the projection

Table 3 Computation times in seconds for projecting $\mathbf{y}$ onto the unit simplex ($a = 1$), 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 is on the simplex for Experiment 4. In all cases, the best time is in bold.

	Experiment 1		
	$N = 10^6$ ($K \approx 6$)	$N = 10^3$ ($K \approx 4$)	$N = 20$ ($K \approx 3$)
Algorithm 1	1.1e-1 (8e-4)	7.4e-5 (5e-6)	1.4e-6 (6e-7)
Algorithm 2	1.1e-2 (6e-4)	9.1e-6 (1e-6)	6.6e-7 (7e-7)
Alg. of Duchi et al.	1.0e-2 (5e-3)	1.1e-5 (6e-6)	8.8e-7 (8e-7)
Alg. 3, random pivot	1.0e-2 (5e-3)	1.1e-5 (6e-6)	9.5e-7 (7e-7)
Alg. 3, median pivot	2.1e-2 (4e-4)	2.7e-5 (3e-6)	9.9e-7 (7e-7)
Algorithm 4	1.8e-2 (7e-4)	1.8e-5 (6e-6)	8.2e-7 (7e-7)
Proposed Algorithm	**1.8e-3** (2e-5)	**1.8e-6** (7e-7)	**5.5e-7** (7e-7)

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

	Experiment 3		
	$N = 10^6$ ($K \approx 21$)	$N = 10^3$ ($K \approx 9$)	$N = 20$ ($K \approx 4$)
Algorithm 1	1.1e-1 (2e-3)	7.4e-5 (5e-6)	1.4e-6 (6e-7)
Algorithm 2	1.1e-2 (3e-4)	**9.6e-6** (1e-6)	7.0e-7 (7e-7)
Alg. of Duchi et al.	1.0e-2 (6e-3)	1.2e-5 (6e-6)	8.9e-7 (7e-7)
Alg. 3, random pivot	1.0e-2 (5e-3)	1.1e-5 (6e-6)	9.6e-7 (7e-7)
Alg. 3, median pivot	2.1e-2 (4e-4)	2.7e-5 (2e-6)	9.9e-7 (7e-7)
Algorithm 4	1.8e-2 (2e-4)	1.8e-5 (2e-6)	7.9e-7 (7e-7)
Proposed Algorithm	**3.5e-3** (2e-4)	1.1e-5 (6e-6)	**6.9e-7** (7e-7)

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

is one operation among others, some of which with complexity $O(N^2)$, like dense matrix-vector products; in such case, the cost of the projection is negligible and all the projection algorithms are equally valid choices. By contrast, for large-scale problems like learning or classification over large dictionaries and in imaging, N is of order 10^6 and more; in such case, all the operations have complexity $O(N)$ or $O(N \log N)$ and a

Table 4 Computation times in seconds for projecting $\mathbf{y}$ onto the unit ℓ_1 ball ($a = 1$), 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.). The y_n are i.i.d. random Gaussian numbers of zero mean and std. dev. 0.1. $\mathbf{y}$ was always outside the ℓ_1 ball. In all cases, the best time is in bold.

	$N = 10^6$ ($K \approx 22$)	$N = 10^3$ ($K \approx 10$)	$N = 20$ ($K \approx 5$)
Algorithm IBIS	1.3e-2 (5e-4)	1.6e-5 (1e-6)	9.8e-7 (7e-7)
Proposed Algorithm	**6.7e-3** (1e-4)	**1.0e-5** (1e-6)	**7.7e-7** (7e-7)

50x speedup of the proposed algorithm with respect to the naive sort-based algorithm does make a big difference.

We can note that the proposed algorithm can be used for matrix approximation problems, by reasoning on their eigenvalues or singular values, e.g. to project a symmetric matrix onto the spectahedron, which is the set of positive semidefinite matrices of unit trace and can be seen as a natural generalization of the unit simplex to symmetric matrices.

Future work will be focused on fast algorithms for optimization problems involving the simplex or the ℓ_1 ball. This includes inverse imaging problems regularized by a constraint on the total variation seminorm [8] and the computation of abundance maps in multispectral unmixing [2].

References

1. Bentley, J.L., McIlroy, M.D.: Engineering a sort function. Software—Practice & Experience **23**(11), 1249–1265 (1993)
2. Bioucas-Dias, J.M., Plaza, A., Dobigeon, N., Parente, M., Du, Q., Gader, P., Chanussot, J.: Hyperspectral unmixing overview: Geometrical, statistical, and sparse regression-based approaches. IEEE J. Sel. Topics Appl. Earth Observations Remote Sens. **5**(2), 354–379 (2012)
3. Blondel, M., Fujino, A., Ueda, N.: Large-scale multiclass support vector machine training via Euclidean projection onto the simplex. In: Proc. of the 22th Int. Conf. on Pattern Recognition (ICPR), pp. 1289–1294 (2014)
4. Blum, M., Floyd, R.W., Pratt, V.R., Rivest, R.L., Tarjan, R.E.: Time bounds for selection. J. Computer and System Sciences **7**(4), 448–461 (1973)
5. Brodie, J., Daubechies, I., Mol, C.D., Giannone, D., Loris, I.: Sparse and stable Markowitz portfolios. Proc. Nat. Acad. Sci. **106**(30), 12,267–12,272 (2009)
6. Cominetti, R., Mascarenhas, W.F., Silva, P.J.S.: A Newton's method for the continuous quadratic knapsack problem. Math. Prog. Comp. **6**, 151–169 (2014)
7. Duchi, J., Shalev-Shwartz, S., Singer, Y., Chandra, T.: Efficient projections onto the ℓ_1-ball for learning in high dimensions. In: Proc. of the 25th Int. Conf. on Machine learning (ICML) (2008)
8. Fadili, J., Peyré, G.: Total variation projection with first order schemes. IEEE Trans. Image Processing **20**(3), 657–669 (2011)
9. Gong, P., Gai, K., Zhang, C.: Efficient Euclidean projections via piecewise root finding and its application in gradient projection. Neurocomputing **74**, 2754–2766 (2011)
10. Held, M., Wolfe, P., Crowder, H.: Validation of subgradient optimization. Mathematical Programming **6**, 62–88 (1974)
11. Kiwiel, K.C.: On Floyd and Rivest's SELECT algorithm. Theoretical Computer Science **347**, 214–238 (2005)
12. Kiwiel, K.C.: Breakpoint searching algorithms for the continuous quadratic knapsack problem. Math. Program., Ser. A **112**, 473–491 (2008)
13. Kiwiel, K.C.: Variable fixing algorithms for the continuous quadratic knapsack problem. J. Optim. Theory Appl. **136**, 445–458 (2008)
14. Knuth, D.E.: The Art of Computer Programming, vol. 2, p. 232, 3rd edn. Addison-Wesley, Boston (1998)

15. Lellmann, J., Kappes, J.H., Yuan, J., Becker, F., Schnörr, C.: Convex multi-class image labeling by simplex-constrained total variation. In: Proc. of Scale Space and Variational Methods in Computer Vision (SSVM), vol. 5567, pp. 150–162 (2009)
16. Liu, J., Ye, J.: Efficient Euclidean projections in linear time. In: Proc. of the 26th Int. Conf. Machine Learning (ICML) (2009)
17. Michelot, C.: A finite algorithm for finding the projection of a point onto the canonical simplex of $\mathbb{R}^n$. J. Optim. Theory Appl. **50**(1), 195–200 (1986)
18. Patriksson, M.: A survey on the continuous nonlinear resource allocation problem. European Journal of Operational Research **185**, 1–46 (2008)
19. Patriksson, M., Strömberg, C.: Algorithms for the continuous nonlinear resource allocation problem—new implementations and numerical studies. European Journal of Operational Research **243**(3), 703–722 (2015)
20. van den Berg, E., Friedlander, M.P.: Probing the Pareto frontier for basis pursuit solutions. SIAM J. Sci. Comput. **31**, 890–912 (2008)