Computational performance of a projection and rescaling algorithm

Javier Peña* Negar Soheili†

January 15, 2019

Abstract

This paper documents a computational implementation of a *projection and rescaling algorithm* for finding most interior solutions to the pair of feasibility problems

$$\text{find } x \in L \cap \mathbb{R}^n_+ \quad \text{and} \quad \text{find } \hat{x} \in L^\perp \cap \mathbb{R}^n_+,$$

where L denotes a linear subspace in $\mathbb{R}^n$ and $L^\perp$ denotes its orthogonal complement. The projection and rescaling algorithm is a recently developed method that combines a *basic procedure* involving only low-cost operations with a periodic *rescaling step.* We give a full description of a MATLAB implementation of this algorithm and present multiple sets of numerical experiments on synthetic problem instances with varied levels of conditioning. Our computational experiments provide promising evidence of the effectiveness of the projection and rescaling algorithm.

Our MATLAB code is publicly available. Furthermore, the simplicity of the algorithm makes a computational implementation in other environments completely straightforward.

1 Introduction

The projection and rescaling algorithm [14] is a recent polynomial-time algorithm designed for solving the polyhedral feasibility problem

$$\text{find } x \in L \cap \mathbb{R}^n_{++}, \tag{1}$$

where L denotes a linear subspace in $\mathbb{R}^n$.

The projection and rescaling algorithm works by combining two building blocks, namely a *basic procedure* and a *rescaling step* as follows. Let $P_L : \mathbb{R}^n \to L$ denote the orthogonal projection onto L. Within a bounded number of low-cost iterations, the basic procedure finds $z \in \mathbb{R}^n_{++}$ such that either

$$P_L z \in \mathbb{R}^n_{++} \tag{2}$$

or

$$\|(P_L z)^+\|_1 \leq \frac{1}{2}\|z\|_\infty, \tag{3}$$

*Tepper School of Business, Carnegie Mellon University, USA, jfp@andrew.cmu.edu
†College of Business Administration, University of Illinois at Chicago, USA, nazad@uic.edu

where $(P_L z)^+ = \max\{0, P_L z\}$. If (2) holds, then $x = P_L z \in L \cap \mathbb{R}^n_{++}$ is a solution to the original problem (1). On the other hand, if (3) holds and $z_i = \|z\|_\infty$ then for every feasible solution x to (1) we have

$$x_i \leq \frac{1}{\|z\|_\infty} \langle z, x \rangle = \frac{1}{\|z\|_\infty} \langle z, P_L x \rangle = \frac{1}{\|z\|_\infty} \langle P_L z, x \rangle \leq \frac{1}{\|z\|_\infty} \|(P_L z)_+\|_1 \cdot \|x\|_\infty \leq \frac{1}{2} \|x\|_\infty.$$

In other words, if (3) holds and $z_i = \|z\|_\infty$ then all solutions x to (1) have small i-th component. The rescaling step takes $D := I + e_i e_i^{\mathrm{T}}$ and transforms problem (1) into the following equivalent rescaled problem:

$$\text{find } x \in D(L) \cap \mathbb{R}^n_{++}. \tag{4}$$

Observe that the solutions to the rescaled problem (4) are in one-to-one correspondence with the solutions to (1) via doubling of the i-th component.

As it is easy to see and detailed in [14], when D is as above, the rescaled problem (4) is better conditioned than (1) in the following sense. If $L \cap \mathbb{R}^n_{++} \neq \emptyset$ then $\delta(D(L) \cap \mathbb{R}^n_{++}) = 2\delta(L \cap \mathbb{R}^n_{++})$ where $\delta(L \cap \mathbb{R}^n_{++})$ is the following *condition measure* of the problem (1):

$$\delta(L \cap \mathbb{R}^n_{++}) := \max \left\{ \prod_{j=1}^n x_j : x \in L \cap \mathbb{R}^n_{++}, \|x\|_\infty = 1 \right\}. \tag{5}$$

By convention $\delta(L \cap \mathbb{R}^n_{++}) = -\infty$ when $L \cap \mathbb{R}^n_{++} = \emptyset$. Observe that $\delta(L \cap \mathbb{R}^n_{++}) \leq 1$ is a measure of the *most interior* solution to (1). As detailed in [14], it follows that when $L \cap \mathbb{R}^n_{++} \neq \emptyset$, the projection and rescaling algorithm finds a solution to (1) in at most $\log_2(1/\delta(L \cap \mathbb{R}^n_{++}))$ rounds of basic procedure and rescaling step. Furthermore, each round of basic procedure and rescaling step requires a number of elementary operations that is bounded by a low-degree polynomial (quadratic or cubic) on n.

The above projection and rescaling algorithm was originally proposed by Chubanov [4, 5] and is in the same spirit as other rescaling methods in [1, 8, 13]. In addition to [14], a number of articles [6, 7, 9, 10, 11, 12, 15] have proposed new algorithmic developments by extending the projection and rescaling templates introduced in [1, 4, 5, 13]. However, despite their interesting theoretical guarantees, there has been limited work on the computational effectiveness of the projection and rescaling algorithm as well as other methods based on rescaling. As far as we know, only the articles by Li et al. [11] and by Roos [15] report numerical results on implementations of some variants of Chubanov's projection and rescaling algorithm.

This paper documents a MATLAB implementation of an enhanced version of the projection and rescaling algorithm from [14]. Our work differs from [11, 15] in several ways. Unlike the algorithms in [11, 15], our main algorithm solves both feasibility problems $L \cap \mathbb{R}^n_+$ and $L^\perp \cap \mathbb{R}^n_+$ in a symmetric fashion. We also perform and report a significantly larger set of computational experiments in higher level of detail. We compare, via numerous experiments, the performance of several possible schemes for the basic procedure. We provide full descriptions of the algorithms that we implement. The MATLAB code for our implementation is publicly available at the following website:

`http://www.andrew.cmu.edu/user/jfp/epra.html`

All of the numerical experiments reported in this paper can be easily replicated and verified via the above code. Furthermore, since our MATLAB code is a verbatim implementation of the algorithms described in the sequel, it is straightforward to replicate our implementation in other numerical computing environments such as R, python, or Julia.

Algorithm 1, the main algorithm in our implementation, incorporates the following enhancements to the original Projection and Rescaling Algorithm in [14]:

1. Let $L^\perp$ denote the orthogonal complement of L. Algorithm 1 finds *most interior* solutions to the problems
$$\text{find } x \in L \cap \mathbb{R}^n_+, \tag{6}$$
and
$$\text{find } \hat{x} \in L^\perp \cap \mathbb{R}^n_+. \tag{7}$$
That is, Algorithm 1 terminates with points in the relative interiors of $L \cap \mathbb{R}^n_+$ and $L^\perp \cap \mathbb{R}^n_+$. In particular, if (6) is strictly feasible then Algorithm 1 finds a point in $L \cap \mathbb{R}^n_{++}$. Likewise, if (7) is strictly feasible then Algorithm 1 finds a point in $L^\perp \cap \mathbb{R}^n_{++}$. Unlike the Projection and Rescaling Algorithm in [14] and the algorithms in [4, 5, 11], Algorithm 1 requires no prior feasibility assumptions about (6) or (7).
2. We enforce an upper bound on the size of the entries of the diagonal rescaling matrices maintained throughout Algorithm 1. The upper bound achieves two major goals. First, it prevents numerical overflow. Second, it yields a natural criteria to determine when the algorithm has found points in the relative interiors of $L \cap \mathbb{R}^n_+$ and $L^\perp \cap \mathbb{R}^n_+$.
3. In contrast to the rescaling step in the original Projection and Rescaling Algorithm that rescales L only in one direction at each round, the rescaling step in Algorithm 1 performs a more aggressive rescaling along *all* directions that can improve the conditioning of the problem. This enhancement is fairly similar to a multiple direction rescaling step introduced by Lourenço et al [12]. It is also similar in spirit to an idea proposed by Roos [15] to obtain sharper rescaling via a modified basic procedure.

The first two enhancements above enable Algorithm 1 to apply without the kind of feasibility assumption required by the original Projection and Rescaling Algorithm, namely that $L \cap \mathbb{R}^n_{++} \neq \emptyset$ or $L^\perp \cap \mathbb{R}^n_{++} \neq \emptyset$ and without concerns about numerical overflow due to excessively large rescaling. On the flip side, the correct termination of Algorithm 1 readily follows from the results in [14] only when one of conditions $L \cap \mathbb{R}^n_{++} \neq \emptyset$ or $L^\perp \cap \mathbb{R}^n_{++} \neq \emptyset$ holds and U is sufficiently large. Although our numerical experiments demonstrate that Algorithm 1 correctly terminates in the majority of the cases, a formal proof of correct termination in the case when both $L \cap \mathbb{R}^n_{++} = \emptyset$ and $L^\perp \cap \mathbb{R}^n_{++} = \emptyset$ is not known yet. The natural conjecture is that Algorithm 1 correctly terminates when U is sufficiently large. We will tackle this interesting theoretical question in some future work.

The basic procedure is the main building block of Algorithm 1. We make a separate comparison of the performance of the following four different schemes for the basic procedure proposed in [14]: perceptron, von Neumann, von Neumann with away-steps, and smooth perceptron schemes. These four schemes are described in Algorithm 2 through Algorithm 5 below. According to the theoretical results established in [14], the first three of these schemes have similar convergence rates while Algorithm 5 (the smooth perceptron scheme) has a faster convergence rate but each main iteration of this scheme is computationally more expensive. Section 3.2 describes various numerical experiments that compare the performance of the four schemes. The experiments consistently demonstrate that indeed Algorithm 5 has the best performance by a wide margin. Therefore, we use Algorithm 5 as the basic procedure within Algorithm 1. Section 3.3 describes results on various numerical experiments that test the performance of Algorithm 1. Our results demonstrate the significant advantage of using aggressive rescaling and confirm a similar observation by Chubanov [5, Section 4.2]. They also provide promising evidence that Algorithm 1 can solve instances of moderate size.

The two main sections of the paper are organized as follows. In Section 2 we describe our enhanced version of the projection and rescaling algorithm. This section also recalls four different schemes for the basic procedure proposed in [14]. In Section 3 we present several sets of numerical experiments. To generate interesting problem instances, we devise a procedure to generate problem instances with arbitrary level of conditioning. We perform several numerical experiments to compare the different schemes for the basic procedure. We also perform a number of experiments to test the effectiveness of the enhanced projection and rescaling algorithm.

2 Enhanced projection and rescaling algorithm

2.1 Main algorithm

Algorithm 1 below describes an enhanced version of the Projection and Rescaling Algorithm from [14]. The algorithm relies on the following characterization of the relative interiors of $L \cap \mathbb{R}^n_+$ and $L^\perp \cap \mathbb{R}^n_+$ for a linear subspace $L \subseteq \mathbb{R}^n$. The characterization in Proposition 1 is a consequence of the classical Goldman-Tucker partition theorem as detailed in [3].

Proposition 1 *Let $L \subseteq \mathbb{R}^n$ be a linear subspace. Then there exists a unique partion $B \cup N = \{1, \ldots, n\}$ such that*

$$\mathrm{ri}\,(L \cap \mathbb{R}^n_+) = \{x \in L \cap \mathbb{R}^n_+ : x_i > 0 \text{ for all } i \in B\},$$

and

$$\mathrm{ri}\,(L^\perp \cap \mathbb{R}^n_+) = \{\hat{x} \in L^\perp \cap \mathbb{R}^n_+ : \hat{x}_i > 0 \text{ for all } i \in N\}.$$

In particular, $x \in \mathrm{ri}\,(L \cap \mathbb{R}^n_+)$ and $\hat{x} \in \mathrm{ri}\,(L^\perp \cap \mathbb{R}^n_+)$ if and only if $x \in L$, $\hat{x} \in L^\perp$, and

$$x_B > 0,\ x_N = 0 \quad \text{and} \quad \hat{x}_N > 0,\ \hat{x}_B = 0. \tag{8}$$

Observe that $B = \{1, \ldots, n\}$ and $N = \emptyset$ when $L \cap \mathbb{R}^n_{++} \neq \emptyset$. Similarly, $B = \emptyset$ and $N = \{1, \ldots, n\}$ when $L^\perp \cap \mathbb{R}^n_{++} \neq \emptyset$. In both of these cases we shall say that the partition (B, N) is *trivial*. We shall say that the partition (B, N) is *non-trivial* otherwise, that is, when $B \neq \emptyset$ and $N \neq \emptyset$.

Each main iteration of Algorithm 1 applies the following steps. First, apply the basic procedure to $D(L) \cap \mathbb{R}^n_+$ and $\hat{D}(L^\perp) \cap \mathbb{R}^n_+$ for some diagonal rescaling matrices D and $\hat{D}$. Next, identify a potential partition (B, N) and terminate if the basic procedures yield $x \in L$ and $\hat{x} \in L^\perp$ satisfying (8). Otherwise, update the rescaling matrices D and $\hat{D}$ and proceed to the next main iteration: Apply the basic procedure to $D(L) \cap \mathbb{R}^n_+$ and $\hat{D}(L^\perp) \cap \mathbb{R}^n_+$, etc.

To prevent numerical overflow, Algorithm 1 caps the entries of the rescaling matrices D and $\hat{D}$ by some pre-specified upper bound U. This upper bound naturally determines a numerical threshold to verify if the algorithm has found solutions in the relative interiors of $L \cap \mathbb{R}^n_+$ and $L^\perp \cap \mathbb{R}^n_+$. More precisely, Algorithm 1 will terminate with points $x \in L$ and $\hat{x} \in L^\perp$ satisfying the following approximation of (8):

$$x_B > 0,\ \|x_N\|_\infty \leq \frac{1}{U}\|x\|_\infty \quad \text{and} \quad \hat{x}_N > 0,\ \|\hat{x}_B\|_\infty \leq \frac{1}{U}\|\hat{x}\|_\infty.$$

Algorithm 1 Enhanced Projection and Rescaling Algorithm (EPRA)

1 (**Initialization**)
Let $D := I$ and $\hat{D} := I$.
Let $P := P_L$ and $\hat{P} := P_{L^\perp}$.
Let $U > 0$ be a pre-specified upper bound on the rescaling matrices D and $\hat{D}$.
2 (**Basic Procedure**)
Find $z \gneqq 0$ such that either $Pz > 0$ or $\|(Pz)^+\|_1 \leq \frac{1}{2}\|z\|_\infty$.
Find $\hat{z} \gneqq 0$ such that either $\hat{P}\hat{z} > 0$ or $\|(\hat{P}\hat{z})^+\|_1 \leq \frac{1}{2}\|\hat{z}\|_\infty$.
3 (**Partition identification**)
Let $x := D^{-1}Pz$ and $\hat{x} := \hat{D}^{-1}\hat{P}\hat{z}$.
Let $B := \{i : |\hat{x}_i| < \frac{1}{U}\|\hat{x}\|_\infty\}$ and $N := \{i : |x_i| < \frac{1}{U}\|x\|_\infty\}$.
4 **if** (B, N) partitions $\{1, \ldots, n\}$ **then**
HALT and **return** $x \in \mathrm{ri}\,(L \cap \mathbb{R}^n_+)$, $\hat{x} \in \mathrm{ri}\,(L^\perp \cap \mathbb{R}^n_+)$ **end if**
5 (**Rescaling step**)
Let $e := (z/\|(Pz)^+\|_1 - 1)^+$, $D := \min\left((I + \mathrm{diag}(e))D, U\right)$ and $P := P_{D(L)}$.
Let $\hat{e} := \left(\hat{z}/\|(\hat{P}\hat{z})^+\|_1 - 1\right)^+$, $\hat{D} := \min\left((I + \mathrm{diag}(\hat{e}))\hat{D}, U\right)$ and $\hat{P} := P_{\hat{D}(L^\perp)}$.
Go back to step 2.

2.2 Basic procedure

Let $P : \mathbb{R}^n \to \mathbb{R}^n$ be the orthogonal projection onto a linear subspace of $\mathbb{R}^n$ and $\epsilon \in (0, 1)$. The goal of the basic procedure is to find a non-zero $z \in \mathbb{R}^n_+$ such that either $Pz > 0$ or $\|(Pz)^+\|_1 \leq \epsilon\|z\|_\infty$. We choose $\epsilon = 1/2$ when the basic procedure is used within Algorithm 1. We next recall the four schemes for the basic procedure proposed in [14]. Algorithm 2 describes the simplest of these schemes, namely the perceptron scheme. In the algorithms below Δ_{n-1} denote the standard simplex in $\mathbb{R}^n$,that is,

$$\Delta_{n-1} = \{x \in \mathbb{R}^n_+ : \|x\|_1 = 1\}.$$

Algorithm 2 Perceptron Scheme

1 Pick $z_0 \in \Delta_{n-1}$, and $t := 0$.
2 **while** $Pz_t \not> 0$ and $\|(Pz_t)^+\|_1 > \epsilon\|z_t\|_\infty$ **do**
Pick $u \in \Delta_{n-1}$ such that $\langle u, Pz_t\rangle \leq 0$.
Let $z_{t+1} := \left(1 - \frac{1}{t+1}\right) z_t + \frac{1}{t+1}u$.
$t := t + 1$.
3 **end while**

Algorithm 3 describes the second basic procedure scheme, namely the von Neumann scheme. This scheme is a greedy variant of the perceptron scheme. This algorithm relies on the following mapping

$$u(v) := \operatorname*{argmin}_{u \in \Delta_{n-1}} \langle u, v\rangle .$$

At each iteration, Algorithm 3 chooses z_{t+1} as the convex combination of z_t and $u(Pz_t)$ that minimizes $\|Pz_{t+1}\|_2$.

Algorithm 3 Von Neumann Scheme

Pick $z_0 \in \Delta_{n-1}$, and $t := 0$.
while $Pz_t \not> 0$ and $\|(Pz_t)^+\|_1 > \epsilon\|z_t\|_\infty$ **do**
Let $u = u(Pz_t)$ and $z_{t+1} := z_t + \theta(u - z_t)$ where

$$\theta_t = \operatorname*{argmin}_{\theta\in[0,1]} \|P(z_t + \theta(u - z_t))\|_2^2 = \frac{\|Pz_t\|_2^2 - \langle u, Pz_t\rangle}{\|Pz_t\|_2^2 + \|Pu\|_2^2 - 2\langle u, Pz_t\rangle}.$$

$t := t + 1$.
end while

Algorithm 4 describes the third basic procedure scheme, namely the von Neumann with away steps scheme, which in turn is a variant of the von Neumann scheme. Algorithm 4 relies on the following construction. Define the *support* of a current iterate z as $S(z) := \{i \in \{1, \ldots, n\} : z_i > 0\}$. At each main iteration Algorithm 4 chooses between two different kinds of steps: *regular* steps as in Algorithm 3 and *away steps* that decrease the weight on a component of z belonging to $S(z)$. The away steps are computed via the mapping

$$v(z) := \operatorname*{argmax}_{\substack{v\in\Delta_{n-1}\\ S(v)\subseteq S(z)}} \langle v, Pz\rangle .$$

Algorithm 4 Von Neumann with Away Steps Scheme

Pick $z_0 \in \Delta_{n-1}$, and $t := 0$.
while $Pz_t \not> 0$ and $\|(Pz_t)^+\|_1 > \epsilon\|z_t\|_\infty$ **do**
Let $u = u(Pz_t)$ and $v = v(z_t)$.
if $\|Pz_t\|^2 - \langle u, Pz_t\rangle > \langle v, Pz_t\rangle - \|Pz_t\|^2$ **then** (regular step)
$a := u - z_t$; $\theta_{\max} = 1$,
else (away step)
$a := z_t - v$; $\theta_{\max} = \frac{\langle v,z\rangle}{1-\langle v,z\rangle}$.
endif
Let $z_{t+1} := z_t + \theta a$ where

$$\theta = \operatorname*{argmin}_{\theta\in[0,\theta_{\max}]} \|P(z_t + \theta a)\|^2 = \min\left\{\theta_{\max}, -\frac{\langle z_t, Pa\rangle}{\|Pa\|^2}\right\}$$

$t := t + 1$.
end while

Algorithm 5 describes the fourth basic procedure scheme, namely the smooth perceptron scheme, which in turn is a variant of the the perceptron scheme that relies on the following smooth version of the mapping $u(\cdot)$. Let $\bar{u} \in \Delta_{n-1}$ be fixed. For $\mu > 0$ let

$$u_\mu(v) := \operatorname*{argmin}_{u\in\Delta_{n-1}} \left\{\langle u, v\rangle + \frac{\mu}{2}\|u - \bar{u}\|^2\right\}.$$

Algorithm 5 Smooth Perceptron Scheme

Let $u_0 := \bar{u}$; $\mu_0 = 2$; $z_0 := u_{\mu_0}(Pu_0)$; and $t := 0$.
while $Pz_t \not> 0$ and $\|(Pz_t)^+\|_1 > \epsilon\|z_t\|_\infty$ **do**
  $\theta_t := \frac{2}{t+3}$
  $u_{t+1} := (1-\theta_t)u_t + \theta_t z_t + \theta_t^2 u_{\mu_t}(Pu_t)$
  $\mu_{t+1} := (1-\theta_t)\mu_t$
  $z_{t+1} := (1-\theta_t)z_t + \theta_t u_{\mu_{t+1}}(Pu_{t+1})$
  $t := t+1$.
end while

3 Numerical experiments

This section describes various sets of numerical experiments that test the Enhanced Projection and Rescaling Algorithm described as Algorithm 1 above. We also performed numerical experiments to compare the four schemes for the basic procedure, namely Algorithm 2 through Algorithm 5 on suitably generated instances.

3.1 Schemes to construct challenging instances

We should note that except for the case when the dimension of the subspace L is about half the dimension of the ambient space $\mathbb{R}^n$, a naive approach to generate random instances yields results of limited interest. More precisely, suppose $L \subseteq \mathbb{R}^n$ is a random subspace generated via $L = \ker(A)$ where the entries of $A \in \mathbb{R}^{m\times n}$ are independently drawn from a standard normal distribution. From a classical result on coverage processes by Wendel [16, Equation (1)] it follows that

$$\mathbb{P}(L^\perp \cap \mathbb{R}^n_{++} \neq \emptyset) = 2^{1-n}\sum_{k=0}^{m-1}\binom{n-1}{k} \quad \text{and} \quad \mathbb{P}(L \cap \mathbb{R}^n_{++} \neq \emptyset) = 2^{1-n}\sum_{k=m}^{n-1}\binom{n-1}{k}. \quad (9)$$

In particular, (9) implies that if n is even and $\dim(L) = n - m = n/2$ then $L \cap \mathbb{R}^n_{++} \neq \emptyset$ with probability 0.5. Furthermore, (9) implies that if $\dim(L) = n - m \gg n/2$ then $L \cap \mathbb{R}^n_{++} \neq \emptyset$ with high probability. Similarly, (9) implies that if $\dim(L) = n - m \ll n/2$ then $L^\perp \cap \mathbb{R}^n_{++} \neq \emptyset$ with high probability. The identity (9) also suggests that when $L \subseteq \mathbb{R}^n$ is a random subspace and $\dim(L)$ is far enough from $n/2$ then with high probability $\max\{\delta(L\cap\mathbb{R}^n_{++}), \delta(L^\perp\cap\mathbb{R}^n_{++})\}$ is bounded away from zero as there is extra room for either L or $L^\perp$ to cut deep inside $\mathbb{R}^n_{++}$. The latter fact can be rigorously stated and justified, albeit in somewhat technical terms, by using the machinery on coverage processes and probabilistic analysis of condition numbers developed by Bürgisser et al [2]. Our numerical experiments confirm that indeed most random instances L with either $\dim(L) \gg n/2$ or $\dim(L) \ll n/2$ are easily solvable without rescaling (see Table 9 in Section 3). Therefore such random instances are not particularly interesting.

We next describe schemes to generate collections of more interesting and challenging instances. First, we describe how to generate random subspaces $L \subseteq \mathbb{R}^n$ such that $L \cap \mathbb{R}^n_{++} \neq \emptyset$ with a *controlled* condition measure $\delta(L \cap \mathbb{R}^n_{++})$. We subsequently describe how to generate random subspaces $L \subseteq \mathbb{R}^n$ such that both $L\cap\mathbb{R}^n_+$ and $L^\perp\cap\mathbb{R}^n_+$ have non-trivial relative interiors.

Proposition 2 *Let $\bar{x} \in \mathbb{R}^n_{++}$ and $\bar{u} \in \mathbb{R}^n_+$ be such that $\|\bar{x}\|_\infty = 1$, $\|\bar{u}\|_1 = n$ and $\bar{u}_j = 0$ whenever $\bar{x}_j < 1$ for $j = 1, \ldots, n$. Let $A = \begin{bmatrix} a_1 & \cdots & a_m \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^{m \times n}$ be such that $a_1 = \bar{u} - \bar{X}^{-1}\mathbf{1}$ and $\langle a_j, \bar{x} \rangle = 0$ for $j = 2, \ldots, m$ where $\bar{X} \in \mathbb{R}^{n \times n}$ is a diagonal matrix with elements of $\bar{x}$ spread across the diagonal and $\mathbf{1} \in \mathbb{R}^n$ is the vector of ones. Then for $L = \ker(A) := \{x \in \mathbb{R}^n : Ax = 0\}$ we have*

$$\bar{x} = \mathsf{argmax} \left\{ \prod_{j=1}^n x_j : x \in L \cap \mathbb{R}^n_{++}, \|x\|_\infty = 1 \right\}.$$

In particular, $L \cap \mathbb{R}^n_{++} \neq \emptyset$ and $\delta(L \cap \mathbb{R}^n_{++}) = \prod_{j=1}^n \bar{x}_j$.

Proof: It suffices to show that

$$\begin{aligned} \bar{x} &= \mathop{\mathsf{argmax}}_x \left\{ \ln \left(\prod_{i=1}^n x_i \right) : x \in L \cap \mathbb{R}^n_{++}, \|x\|_\infty = 1 \right\} \\ &= \mathop{\mathsf{argmax}}_x \left\{ \ln \left(\prod_{i=1}^n x_i \right) : x \in L \cap \mathbb{R}^n_{++}, \|x\|_\infty \leq 1 \right\}. \end{aligned} \tag{10}$$

The conditions on the rows of A readily ensure that $\bar{x} \in L \cap \mathbb{R}^n_{++}$. Thus $\bar{x}$ is a feasible solution to (10). Since $\zeta = \bar{X}^{-1}\mathbf{1}$ is the gradient of the objective function in (10) at $\bar{x}$, to show that $\bar{x}$ is optimal it suffices to show that $\langle \zeta, x - \bar{x} \rangle \leq 0$ for any feasible solution to (10). Take $x \in L \cap \mathbb{R}^n_{++}$ with $\|x\|_\infty \leq 1$. Since $x \in L$, we have $\left\langle \bar{X}^{-1}\mathbf{1} - \bar{u}, x \right\rangle = \langle a_1, x \rangle = 0$. Therefore $\langle \zeta, x - \bar{x} \rangle = \left\langle \bar{X}^{-1}\mathbf{1}, x \right\rangle - n \leq \langle \bar{u}, x \rangle - \|\bar{u}\|_1 \|x\|_\infty \leq 0$. The last two steps follow from $\|\bar{u}\|_1 = n$ and Hölder's inequality respectively. ■

Proposition 2 readily suggests a scheme to generate subspaces $L \subseteq \mathbb{R}^n$ such that the condition measure $\delta(L \cap \mathbb{R}^n_{++})$ is positive but as small as we wish: pick $\bar{x} \in \mathbb{R}^n_{++}$ with $\|\bar{x}\|_\infty = 1$ and generate $\bar{u} \in \mathbb{R}^n_+, A \in \mathbb{R}^{m \times n}$, and $L = \ker(A)$ as in Proposition 2. We next explain how Proposition 2 can be further leveraged to generate $L \subseteq \mathbb{R}^n$ so that both $L \cap \mathbb{R}^n_+$ and $L^\perp \cap \mathbb{R}^n_+$ have non-trivial relative interiors. Suppose (B, N) is a partition of $\{1, \ldots, n\}$ and

$$A = \begin{bmatrix} A_{BB} & A_{NB} \\ 0 & A_{NN} \end{bmatrix} \tag{11}$$

is such that $L_B = \ker(A_{BB}) \subseteq \mathbb{R}^B$ and $L_N = \mathrm{Im}(A_{NN}^{\mathrm{T}}) \subseteq \mathbb{R}^N$ satisfy $L_B \cap \mathbb{R}^B_{++} \neq \emptyset$ and $L_N \cap \mathbb{R}^N_{++} \neq \emptyset$. If A_{NN} is full row-rank then it readily follows that the subspaces $L = \ker(A)$ and $L^\perp = \mathrm{Im}(A^{\mathrm{T}})$ satisfy

$$\mathrm{ri}\,(L \cap \mathbb{R}^n_+) = \{x \in L \cap \mathbb{R}^n_+ : x_i > 0 \text{ for all } i \in B\}$$

and

$$\mathrm{ri}\,(L^\perp \cap \mathbb{R}^n_+) = \{\hat{x} \in L^\perp \cap \mathbb{R}^n_+ : \hat{x}_i > 0 \text{ for all } i \in N\}.$$

Hence we can generate subspaces $L \subseteq \mathbb{R}^n$ such that both $L \cap \mathbb{R}^n_+$ and $L^\perp \cap \mathbb{R}^n_+$ have non-trivial relative interiors by proceeding as follows. First, choose a partition (B, N) of $\{1, \ldots, n\}$. Next, use the construction suggested by Proposition 2 to generate full row-rank matrices A_{BB}, A_{NN} such that $\ker(A_{BB}) \cap \mathbb{R}^B_{++} \neq \emptyset$ and $\mathrm{Im}(A_{NN}^{\mathrm{T}}) \cap \mathbb{R}^N_{++} \neq \emptyset$. Finally let $L = \ker(A)$ where A is of the form (11) for some A_{NB} of appropriate size.

3.2 Comparison of basic procedure schemes

The computational experiments summarized in this section compare the performance of the four schemes for the basic procedure, namely Algorithm 2 through Algorithm 5. We implemented these algorithms in MATLAB and ran them on collections of instances defined by $L = \ker(A)$, for $A \in \mathbb{R}^{m\times n}$. We used the QR-factorization to obtain the orthogonal projection mappings $P = P_L$ and $\hat{P} = P_{L^\perp}$.

We performed two main sets of experiments. The first set of experiments contains instances $L = \ker(A)$ where the entries of $A \in \mathbb{R}^{m\times n}$ are independently drawn from a standard normal distribution and $m = n/2$ for $n = 200, 500, 1000, 2000$. When m significantly differs from $n/2$, random instances generated in this way are uninteresting as they can easily be solved by any of the four schemes. The second set of experiments contains more challenging instances $L = \ker(A)$ where $A \in \mathbb{R}^{m\times n}$ is generated via the procedure suggested by Proposition 2 for $n = 1000$, $m = 100, 200, 800, 900$. More precisely, we generated $\bar{x} \in \mathbb{R}^n_{++}$ as follows. First, we set a random chunk of its entries uniformly at random between 0 and 0.001. Second, we set remaining entries uniformly at random distributed between 0 and 1. Third, we scaled the entries of $\bar{x}$ to obtain $\|\bar{x}\|_\infty = 1$. Once we generated $\bar{x}$ in this fashion, we generated $A \in \mathbb{R}^{m\times n}$ as in Proposition 2.

Table 1 through Table 4 summarize the results on various sets of experiments. Each row corresponds to a set of 1000 instances. To keep the number of iterations and CPU time manageable, we enforced an upper bound of 10000 iterations for all four schemes. The first two columns in each table indicate the size of $A \in \mathbb{R}^{m\times n}$. The other columns display three numbers for each of the four schemes: the average number of iterations, the average CPU time, and the success rate on the batch of 1000 instances of size m by n. The success rate is the proportion of instances on which the scheme terminates normally before reaching the upper bound of 10000 iterations.

Table 1 and Table 2 display the results for the first set of experiments when $m = n/2$ and $A \in \mathbb{R}^{m\times n}$ is randomly generated without any control on the conditioning of $L \cap \mathbb{R}^n_{++}$. Table 3 and Table 4 display similar summaries for the second set of experiments where we generate $A \in \mathbb{R}^{m\times n}$ so that $L \cap \mathbb{R}^n_{++}$ has a controlled condition measure via the procedure suggested by Proposition 2. The tables summarize results for two values of ϵ: $\epsilon = 10^{-1}$ (large), and $\epsilon = 10^{-4}$ (small).

Table 1: Results for naive random instances, large ϵ ($\epsilon = 10^{-1}$), and 10000 iteration limit

m	n	perceptron	VN	VNA	smooth
100	200	(6956.28, 0.27, 0.74)	(5070.41, 0.26, 0.69)	(3021.73, 0.23, 0.95)	(27.04, 0.03, 1)
250	500	(9963.91, 0.85, 0.02)	(9207.1, 0.26, 0.2)	(8737.9, 0.23, 0.38)	(43.88, 0.13, 1)
500	1000	(10000, 8.67, 0)	(9981.29, 8.84, 0.01)	(9992.46, 14.3, 0.01)	(58.50, 0.42, 1)
1000	2000	(10000, 34.72, 0)	(10000, 35.54, 0)	(10000, 67.24, 0)	(80.21, 1.42, 1)

Table 2: Results for naive random instances, small ϵ ($\epsilon = 10^{-4}$), and 10000 iteration limit

m	n	perceptron	VN	VNA	smooth
100	200	(8236.2, 0.33, 0.35)	(5395.1, 0.28, 0.67)	(5861, 0.45, 0.66)	(123.6, 0.14, 1)
250	500	(9981.9, 0.94, 0.01)	(9258.1, 0.99, 0.2)	(9518.5, 1.64, 0.16)	(231.8, 0.64, 1)
500	1000	(10000, 8.28, 0)	(9973.7, 8.39, 0.01)	(9988.5, 13.57, 0.01)	(337.64, 2.15, 1)
1000	2000	(10000, 35.61, 0)	(10000, 36.34, 0)	(10000, 68.71, 0)	(465.94, 7.87, 1)

Table 3: Results for controlled condition instances, large ϵ ($\epsilon = 10^{-1}$), and 10000 iteration limit

m	n	perceptron	VN	VNA	smooth
100	1000	(9134.38, 0.88, 0.32)	(8519.12, 8.48, 0.25)	(3329.84, 6.27, 1)	(130.77, 0.30, 1)
200	1000	(9649.15, 8.02, 0.21)	(8645.91, 7.35, 0.26)	(5005.64, 8.12, 0.98)	(140.21, 0.27, 1)
800	1000	(3383.55, 2.86, 0.87)	(9798.34, 8.33, 0.03)	(6566.22, 10.61, 0.7)	(220.53, 0.42, 1)
900	1000	(2156.34, 1.92, 0.99)	(9842.71, 8.71, 0.03)	(1429.66, 2.43, 1)	(198.58, 0.39, 1)

Table 4: Results for controlled condition instances, small ϵ ($\epsilon = 10^{-4}$), and 10000 iteration limit

m	n	perceptron	VN	VNA	smooth
100	1000	(9961.9, 8.26, 0)	(9926.9, 16.06, 0.01)	(9934.6, 16.01, 0.01)	(9463.5, 17.65, 0.16)
200	1000	(9952.6, 8.74, 0.01)	(9942.1, 16.95, 0.01)	(9950.1, 16.95, 0.01)	(9579.9, 19.16, 0.13)
800	1000	(9964.52, 8.94, 0)	(9961.16, 17.15, 0)	(9967.6, 17.15, 0)	(8557.5, 17.05, 0.75)
900	1000	(9939.5, 8.80, 0.01)	(9915.7, 16.94, 0.01)	(9939.1, 16.94, 0.01)	(7537.2, 14.85, 0.97)

When ϵ is large (Table 1 and Table 3), the algorithms often stop when the condition $\|(Pz)^+\|_1 \leq \epsilon\|z\|_\infty$ is satisfied. Not surprisingly, when ϵ is small (Table 2 and Table 4), the basic procedures more often stop when $Pz > 0$ and require a larger number of iterations and longer CPU time. Also as expected, when the instances become larger, they become more challenging and more iterations are needed to find a feasible solution.

Our numerical experiments for large ϵ demonstrate that the smooth perceptron scheme is faster both in number of iterations and in terms of CPU time than any of the other three schemes. The experiments also suggest that when enforcing the 10000 iteration limit the perceptron, von Neumann, and von Neumann with away steps are comparable in terms of number of iterations and CPU time. Given the evidence in favor of the smooth perceptron scheme, we use this method within the Enhanced Projection and Rescaling Algorithm.

We note that the numerical experiments for small ϵ in Table 4 confirm that the scheme for generating challenging instances indeed yields instances that are difficult to solve for all schemes and thus provide an interesting testbed for the Enhanced Projection and Rescaling Algorithm.

The low success rates in some of the entries in Table 1 through Table 4 reveal that for many instances the upper limit of 10000 iterations is reached by the perceptron, von Neumann, and von Neumann with away steps schemes. Thus for additional robustness check, we also performed some extra sets of experiments without any limit on the number of iterations. The results are summarized in Table 5 and Table 6. We ran fewer instances and used $\epsilon = 10^{-1}$ to keep the experiments manageable. (Some schemes run for over several million iterations in some instances.) The last four columns of Table 5 and Table 6 report only the average number of iterations and average CPU times since all instances are run until successful termination without iteration limit. Each row corresponds to a set of 100 instances except for the last row for $m = 1000$, $n = 2000$. In this case we only ran 20 instances due to time limitations. Without iteration limit, some of these instances take multiple hours of CPU time.

The results in these two tables further confirm that the smooth perceptron scheme is faster both in number of iterations and in terms of CPU time than any of the other three schemes. Furthermore, the additional experiments suggest that without iteration limit the von Neumann scheme usually requires the highest number of iterations.

Table 5: Results for naive random instances, large ϵ ($\epsilon = 10^{-1}$), and no iteration limit

m	n	perceptron	VN	VNA	smooth
100	200	(8780.80, 0.31)	(24453.42, 1.11)	(3054.96, 0.22)	(66.7, 0.008)
250	500	(62807.14, 11.8)	(565958.64, 112.7)	(22853.18, 8.13)	(122.18, 0.06)
500	1000	(267348.4, 227.73)	(2301999.2, 2017.9)	(91897.2, 151.3)	(177.0,0.34)
1000	2000	(856072.0, 2739.47)	(717508.8, 2331.06)	(162726.9, 1010.66)	(226.6, 1.6)

Table 6: Results for controlled condition instances, large ϵ ($\epsilon = 10^{-1}$), and no iteration limit

m	n	perceptron	VN	VNA	smooth
100	1000	(14982.34, 15.64)	(47942.98, 51.68)	(3585.23, 6.97)	(127.93, 0.29)
200	1000	(16037.40, 16.67)	(57652.07, 62.43)	(5152.81, 9.87)	(146.49, 0.33)
800	1000	(9157.77, 8.8)	(892550.84, 905.88)	(7243.26, 13.11)	(220.8, 0.47)
900	1000	(1975.82, 1.93)	(692402.25, 695.27)	(1410.76, 2.46)	(199.8, 0.43)

3.3 Performance of the Enhanced Projection and Rescaling Algorithm

This section describes the performance of Algorithm 1 on two main sets of problem instances. The first set contains instances of $L = \ker(A)$ for $A \in \mathbb{R}^{m\times n}$ with $L \cap \mathbb{R}^n_{++} \neq \emptyset$ generated via the approach based on Proposition 2 as described in Section 3.1. The second set of instances $L = \ker(A)$ is also generated via a similar approach but ensuring that both $\text{ri}\,(L \cap \mathbb{R}^n_+) \neq \{0\}$ and $\text{ri}\,(L^\perp \cap \mathbb{R}^n_+) \neq \{0\}$. Most of these instances are sufficiently challenging that they cannot be solved by the basic procedure (via the smooth perceptron scheme) without rescaling.

We ran Algorithm 1 with $U = 10^{10}$ in all of our experiments. Table 7 displays the results for the first set of instances $L = \ker(A)$ with $L \cap \mathbb{R}^n_{++} \neq \emptyset$. Each row corresponds to a set of 500 instances of $A \in \mathbb{R}^{m \times n}$ for m and n as indicated in the first two columns. The other three columns display the average number of rescaling iterations, average total number of basic procedure iterations, and average CPU time for each set of 500 instances. Furthermore, we note that Algorithm 1 successfully solves all instances, that is, it terminates with a point $x \in L \cap \mathbb{R}^n_{++}$. It is noteworthy that the number of rescaling iterations ranges from 9 to 15 across instances of different sizes. To further illustrate this interesting fact, Figure 1 plots the number of rescaling iterations for some sets of instances.

Table 7: Algorithm 1 on controlled condition instances with $L \cap \mathbb{R}^n_{++} \neq \emptyset$

m	n	Average # of rescaling iterations	Average total # of iterations	Average CPU time (in seconds)
100	200	9.51	712.38	0.076
250	500	11.03	1419.98	0.76
100	1000	7.97	1843.74	4.48
200	1000	9.00	1954.73	4.41
500	1000	11.92	2487.47	5.49
800	1000	13.08	4026.00	6.69
900	1000	11.63	4184.90	6.64
1000	2000	12.18	4318.48	35.76

Table 8 and Figure 2 display similar results for the second set of instances with both $\text{ri}\,(L \cap \mathbb{R}^n_+) \neq \emptyset$ and $\text{ri}\,(L^\perp \cap \mathbb{R}^n_+) \neq \emptyset$. To accommodate for a wide and flexible range of dimensions of $\text{ri}\,(L \cap \mathbb{R}^n_+) \neq \emptyset$ and $\text{ri}\,(L^\perp \cap \mathbb{R}^n_+) \neq \emptyset$, for each fixed value of n we construct $A \in \mathbb{R}^{m \times n}$ and $L = \ker(A)$ with varying values of m. For this second set of instances we also report the *success rate*, that is, the percentage of instances where the the partition (B, N) is correctly identified. The algorithm succeeds in identifying this partition for most instances. In the rare cases when this is not the case, failure occurs because either B or N are small and Algorithm 1 terminates with a point that is either in $L^\perp \cap \mathbb{R}^n_{++}$ or in $L \cap \mathbb{R}^n_{++}$ within roundoff error. The experiments show that on this second set of instances a higher number of rescaling iterations is usually necessary. This is somewhat expected as these instances include the extra difficulty of finding a non-trivial partition (B, N) of $\{1, \ldots, n\}$.

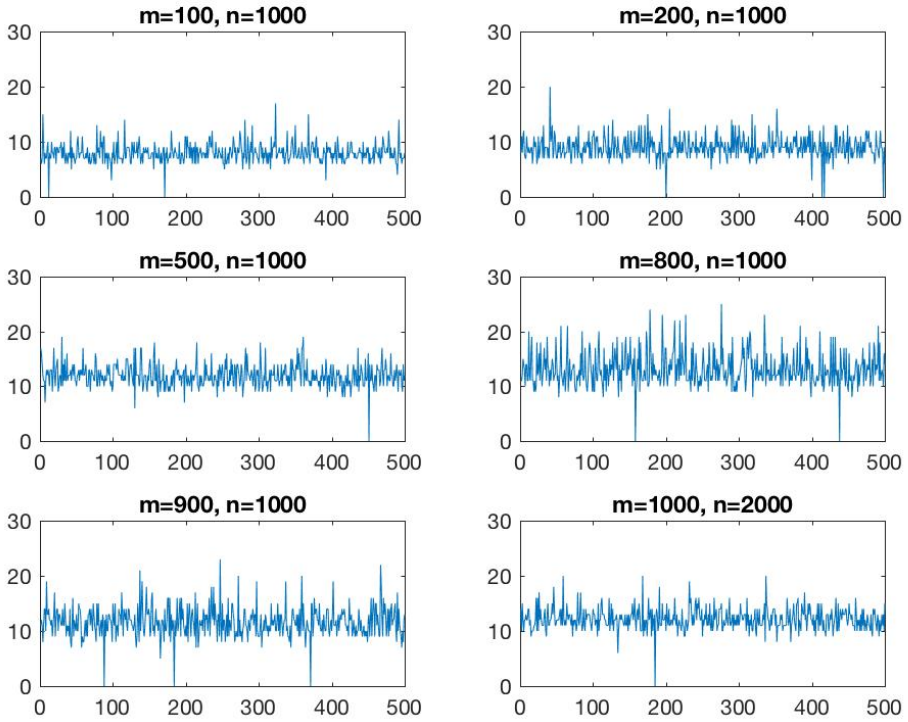


Figure 1: Number of rescaling iterations for controlled condition instances L with $L \cap \mathbb{R}^n_{++} \neq \emptyset$.

Table 8: Algorithm 1 on instances with $\text{ri}\,(L \cap \mathbb{R}^n_+) \neq \{0\}$ and $\text{ri}\,(L^\perp \cap \mathbb{R}^n_+) \neq \{0\}$

n	Average m	Average # of rescaling iterations	Average total # of iterations	Average CPU time (in seconds)	Success rate
100	51.21	16.16	657.01	0.075	0.946
200	100.55	17.27	1031.45	0.12	0.974
500	249.64	17.60	1763.59	1.09	0.984
800	405.86	17.72	2269.69	3.52	0.998
1000	499.72	17.76	2625.84	6.73	0.994
2000	1006.93	17.59	3752.04	47.07	0.994

To further illustrate the partition and the solutions found by Algorithm 1, Figure 3 and Figure 4 plot the coordinates of the points x and $\hat{x}$ found by Algorithm 1 for two representative instances of dimension $n = 1000$ and $n = 2000$ respectively. The two plots in the first row of Figure 3 and Figure 4 show the components of the points $x = (x_B, x_N)$ and $\hat{x} = (\hat{x}_B, \hat{x}_N)$ returned by Algorithm 1 for an instance of size $n = 1000$ and for an instance of size $n = 2000$. The set B is $\{1, \ldots, 424\}$ in the first instance and it is $\{1, \ldots, 1137\}$ in the second instance. The large red circle in the plots show the size of B. For scaling purposes, in both instances the vectors x and $\hat{x}$ are normalized so that $\|x\|_\infty = \|\hat{x}\|_\infty = 1$. As Figure 3 and Figure 4 show, in both cases the solutions x and $\hat{x}$ satisfy the conditions in (8).

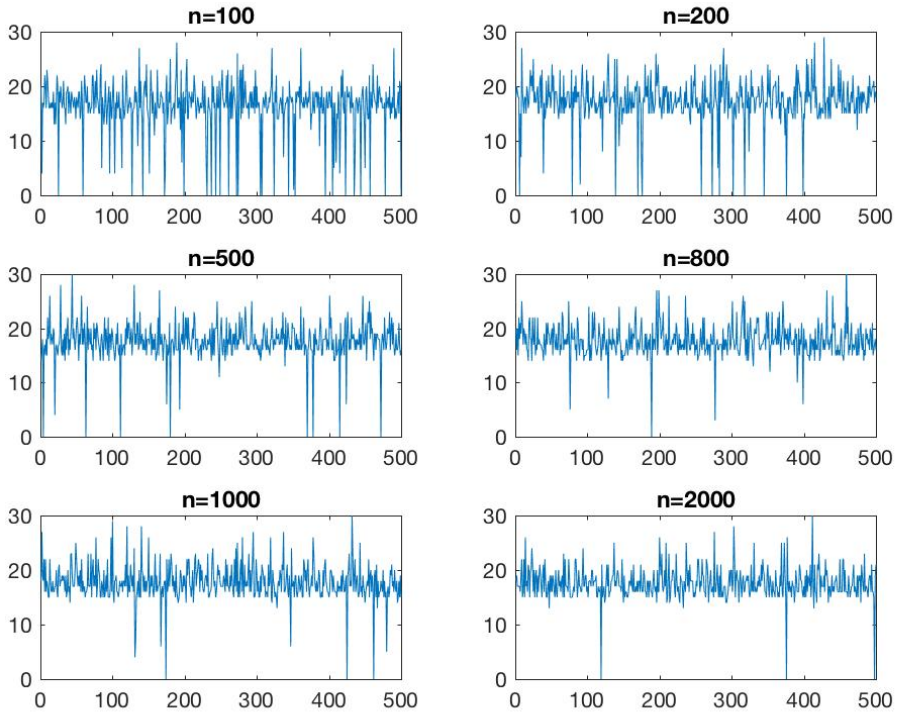
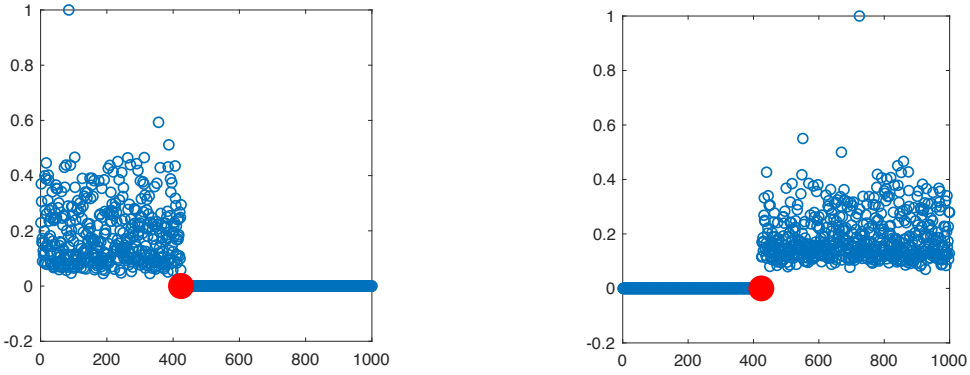
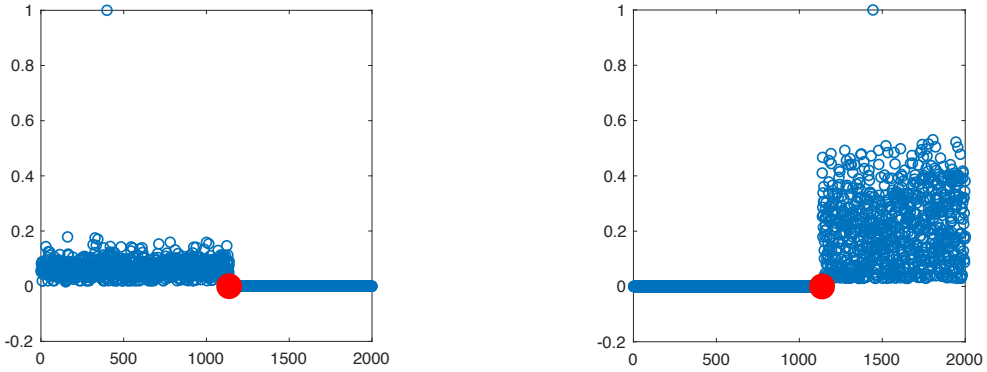


Figure 2: Number of rescaling iterations for instances L with $\operatorname{ri}(L \cap \mathbb{R}^n_+) \neq \{0\}$ and $\operatorname{ri}(L^\perp \cap \mathbb{R}^n_+) \neq \{0\}$.



(a) $x \in L \cap \mathbb{R}^n_{++}$ (b) $\hat{x} \in L^\perp \cap \mathbb{R}^n_{++}$

Figure 3: Coordinates of $x \in L \cap \mathbb{R}^n_+$ and $\hat{x} \in L^\perp \cap \mathbb{R}^n_+$ found by Algorithm 1 for some $L \subseteq \mathbb{R}^n$ and $n = 1000$.



(a) $x \in L \cap \mathbb{R}^n_{++}$ (b) $\hat{x} \in L^\perp \cap \mathbb{R}^n_{++}$

Figure 4: Coordinates of $x \in L \cap \mathbb{R}^n_+$ and $\hat{x} \in L^\perp \cap \mathbb{R}^n_+$ found by Algorithm 1 for some $L \subseteq \mathbb{R}^n$ and $n = 2000$.

3.4 Other experiments

We also performed experiments to assess the effect of rescaling along multiple directions, as in Algorithm 1, versus rescaling only along one direction, as in the original Projection and Rescaling Algorithm in [14]. More precisely, we compare the performance of Algorithm 1 versus the modification obtained by changing the update on D and $\hat{D}$ in Step 5 to $D = \min\left(\left(I + \operatorname{diag}(e_i)\right) D, U\right)$ and $\hat{D} = \min\left(\left(I + \operatorname{diag}(e_j)\right) \hat{D}, U\right)$ where i and j are such that for $z_i = \|z\|_\infty$ and $\hat{z}_j = \|\hat{z}\|_\infty$. In most instances the modified version that rescales along one direction failed to find a solution within a reasonable number (a hundred) of rescaling iterations. We note that [14, Section 6] provides a closed-form formula to update the projection matrix P at low cost after rescaling along one direction. The formula can be extended to handle multiple rescaling directions. However, the formula is computationally attractive only when the number of rescaling directions is small because it requires computing the spectral decomposition of a matrix with rank equal to the number of rescaling directions. Our numerical experiments indicate that even using the closed-form formula in [14] does not compensate for the additional number of iterations required by the modified version of Algorithm 1 with a single rescaling direction.

Table 9 provides a summary similar to that displayed on Table 7 of the performance of Algorithm 1 on a set of naive random instances. These instances were generated in the same way as those used in the experiments summarized Table 1 and Table 2, namely, $L = \ker(A)$ where the entries of $A \in \mathbb{R}^{m \times n}$ are independently drawn from a standard normal distribution. In contrast to the results summarized in Table 7 for controlled condition instances, Algorithm 1 solves most instances easily without rescaling and after a much lower number of total basic iterations. In particular, Algorithm 1 solves all naive random instances without rescaling when $m \neq n/2$ and only a few instances require a small number of rescaling steps when $m = n/2$. For additional illustration of the latter fact, Figure 5 plots the number of rescaling iterations for the naive random instances with $m = n/2$. The last column of Table 9 shows the fraction of instances where $L \cap \mathbb{R}^n_{++} \neq \emptyset$. In contrast to the controlled condition instances, this is unknown for naive random instances. The fraction of instances where $L \cap \mathbb{R}^n_{++} \neq \emptyset$ is consistent with (9) and the subsequent discussion.

Table 9: Algorithm 1 on naive random instances

m	n	Average # of rescaling iterations	Average total # of iterations	Average CPU time (in seconds)	Fraction of instances with $L \cap \mathbb{R}^n_{++} \neq \emptyset$
100	200	0.496	147.094	0.0213	0.5
250	500	0.338	257.660	0.1428	0.496
100	1000	0	3.700	0.1574	1
200	1000	0	9.520	0.2008	1
500	1000	0.228	373.352	0.7646	0.502
800	1000	0	5.738	0.1175	0
900	1000	0	2.616	0.1156	0
1000	2000	0.188	602.674	4.4935	0.482

We also compared the performance of Algorithm 1 with the state-of-the-art commercial solver CPLEX. Similar comparisons with Gurobi and MATLAB solvers are reported in [11, 15]. Consistent with the reported results in [11, 15], we observe that on average Algorithm 1 is faster than CPLEX by nearly an order of magnitude for problem instances where $L = \ker(A)$ with $A \in \mathbb{R}^{m\times n}$ generated naively at random as in [11, 15] and as in the set of experiments summarized in Table 9. On the other hand, the difference in speed is about the opposite, that is, CPLEX is nearly an order of magnitude faster when $A \in \mathbb{R}^{m\times n}$ is generated so that $L \cap \mathbb{R}^n_{++}$ has a controlled condition measure via the procedure suggested by Proposition 2 as in the set of experiments summarized in Table 7. We attribute this sharp difference to the fact that the naively generated instances are generally easier and can usually be solved within one single round of basic procedure and without the need for rescaling even when $m = n/2$ as Table 9 illustrates. By contrast, instances with controlled condition measure, such as those in the set of experiments summarized in Table 7, are significantly more challenging and require on average ten or more rounds of basic and rescaling steps. We note that for similarly generated instances, the numerical experiments reported in [15] generally require several rescaling iterations when $m = n/2$ while our algorithm solves most of these instances without any rescaling iterations. This difference is likely due to the different basic procedures used in [15] and in our numerical experiments. The rescaling method in [15] uses a variant of the von Neumann algorithm as its basic procedure while we use the smooth perceptron scheme.

4 Concluding remarks

We have described a computational implementation and numerical experiments of an Enhanced Projection and Rescaling algorithm for finding most interior solutions to the feasibility problems

$$\text{find } x \in L \cap \mathbb{R}^n_+ \quad \text{and} \quad \text{find } \hat{x} \in L^\perp \cap \mathbb{R}^n_+,$$

where L denotes a linear subspace in $\mathbb{R}^n$ and $L^\perp$ denotes its orthogonal complement. Our numerical results provide promising evidence of the effectiveness of this algorithmic approach.

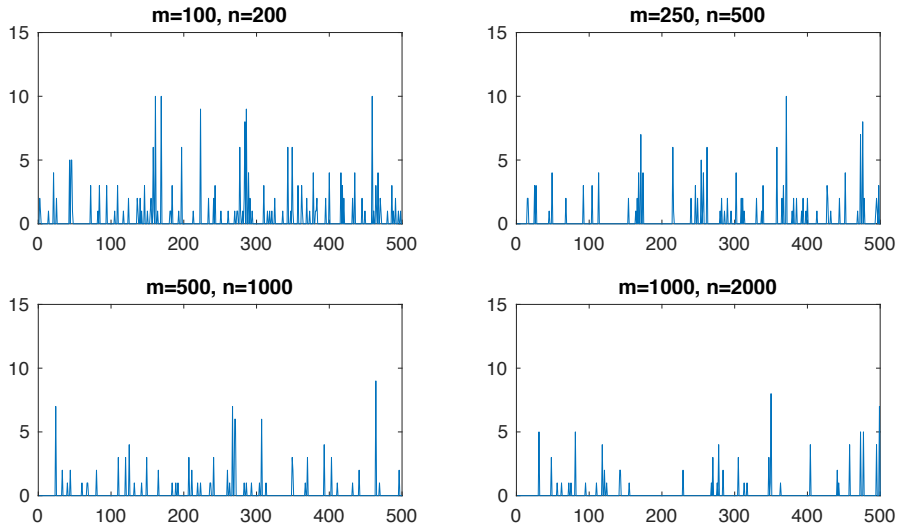


Figure 5: Number of rescaling iterations for naive random instances

The MATLAB code for our implementation is comprised of a set of MATLAB functions with verbatim implementations of Algorithm 1 through Algorithm 5. Our MATLAB code is publicly available at the following website

`http://www.andrew.cmu.edu/user/jfp/epra.html`

The tables presented in this paper were created by averaging the results obtained from running the following MATLAB functions.

- `TestSimpleBasicProcedures(m,n,N,epsilon)`: This is the code used to generate and test the set of instances summarized in each row of Table 1, Table 2, and Table 5.
- `TestControlledConditionBasicProcedures(m,n,N,epsilon,delta)`: This is the code used to generate and test the set of instances summarized in each row of Table 3, Table 4, and Table 6.
- `TestControlledConditionRescaled(m,n,N,delta)`: This is the code used to generate and test the set of instances summarized in each row of Table 7.
- `TestPartitionRescaled(n,N)`: This is the code used to generate and test the set of instances summarized in each row of Table 8.
- `TestSimpleRescaled(m,n,N)`: This is the code used to generate and test the set of instances summarized in each row of Table 9. This code also compares the performance of Algorithm 1 with a modified version that rescales along one direction only including a more efficient update on the projection matrix after each rescaling step.

The input parameters for the above functions are as follows

- `N`: Number of instances. We used `N = 1000` in Table 1 through Table 4, and `N = 100` in Table 5 and Table 6. We used `N = 500` in Table 7 through Table 9.
- `m`: Number of rows of $A \in \mathbb{R}^{m \times n}$ such that $L = \ker(A)$
- `n`: Dimension of the ambient space
- `epsilon`: Rescaling condition parameter

- `delta:` Upper bound on the values of a subset of randomly chosen positive entries of the most central solution. The smaller `delta,` the more ill-conditioned the problem. We used `delta = 0.001` for the experiments summarized in Table 3, Table 4, Table 6, and Table 7.

Algorithm 1 through Algorithm 5 are implemented via the following MATLAB functions.

- `MultiEPRA(A,AA,n,z0,U):` This code implements Algorithm 1. Assume $L = \ker(\mathtt{A}) \subseteq \mathbb{R}^n$ and $L^\perp = \ker(\mathtt{AA})$. Use `z0` as starting point for the basic procedure and `U` to upper bound the rescaling matrices.
- `perceptron(P,z0,epsilon):` This code implements Algorithm 2.
- `VN(P,z0,epsilon):` This code implements Algorithm 3.
- `VNA(P,z0,epsilon):` This code implements Algorithm 4.
- `smooth(P,u0,epsilon):` This code implements Algorithm 5

.

References

[1] A. Belloni, R. Freund, and S. Vempala. An efficient rescaled perceptron algorithm for conic systems. *Math. of Oper. Res.*, 34(3):621–641, 2009.

[2] P. Bürgisser, F. Cucker, and M. Lotz. Coverage processes on spheres and condition numbers for linear programming. *The Annals of Probability*, 38(2):570–604, 2010.

[3] D. Cheung, F. Cucker, and J. Peña. Unifying condition numbers for linear programming. *Math. Oper. Res*, 28(4):609–624, 2003.

[4] S. Chubanov. A strongly polynomial algorithm for linear systems having a binary solution. *Math. Program.*, 134:533–570, 2012.

[5] S. Chubanov. A polynomial projection algorithm for linear feasibility problems. *Math. Program.*, 153:687–713, 2015.

[6] D. Dadush, L. A Végh, and G. Zambelli. Rescaled coordinate descent methods for linear programming. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 26–37. Springer, 2016.

[7] D. Dadush, L. A Végh, and G. Zambelli. Rescaling algorithms for linear conic feasibility. *arXiv preprint arXiv:1611.06427*, 2017.

[8] R. Freund, R. Roundy, and M. Todd. Identifying the set of always-active constraints in a system of linear inequalities by a single linear program. *Working Paper, Massachusetts Institute of Technology, Alfred P. Sloan School of Management*, 1985.

[9] R. Hoberg and T. Rothvoss. An improved deterministic rescaling for linear programming algorithms. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 267–278. Springer, 2017.

[10] T. Kitahara and T. Tsuchiya. An extension of Chubanov's polynomial-time linear programming algorithm to second-order cone programming. *Optimization Methods and Software*, 33(1):1–25, 2018.

[11] D. Li, C. Roos, and T. Terlaky. A polynomial column-wise rescaling von Neumann algorithm. Technical report, Lehigh University, 2015.

[12] B. Lourenço, T. Kitahara, M. Muramatsu, and T. Tsuchiya. An extension of Chubanov's algorithm to symmetric cones. *Math. Program.*, pages 1–33, 2016.

[13] J. Peña and N. Soheili. A deterministic rescaled perceptron algorithm. *Math. Program.*, 155:497–510, 2016.

[14] J. Peña and N. Soheili. Solving conic systems via projection and rescaling. *Math. Program.*, 166:87–111, 2017.

[15] C. Roos. An improved version of Chubanov's method for solving a homogeneous feasibility problem. *Optimization Methods and Software*, 33:26–44, 2018.

[16] J. Wendel. A problem in geometric probability. *Math. Scand*, 11:109–111, 1962.