Computational performance of a projection and rescaling algorithm

Javier Peña*  Negar Soheili†

March 19, 2018

Abstract

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

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

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 [13] is a recent polynomial-time algorithm designed for solving the polyhedral feasibility problem

\[
\text{find } x \in L \cap \mathbb{R}^n_+,
\]

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

\[
P_L z \in \mathbb{R}^n_+ \quad \text{(2)}
\]

or

\[
\|(P_L z)^+\|_1 \leq \frac{1}{2} z_i = \|z\|_\infty, \quad \text{(3)}
\]

*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 $P_L z \in L \cap \mathbb{R}_+^n$ is a solution to the original problem (1). On the other hand, if (3) holds, then the rescaling step takes $D := I + e_i e_i^T$ and transforms problem (1) into the following equivalent rescaled problem:

$$\text{find } x \in D(L) \cap \mathbb{R}_+^n. \quad (4)$$

Observe that a solution to the rescaled problem (4) can readily be transformed back to a solution to (1).

As it is easy to see and detailed in [13], 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):

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

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 [13], 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 [3, 4] and is in the same spirit as other rescaling methods in [1, 7, 12]. In addition to [13], a number of articles [5, 6, 8, 9, 10, 11, 14] have proposed new algorithmic developments by extending the projection and rescaling templates introduced in [1, 3, 4, 12]. 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. [10] and by Roos [14] 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 [13]. Our work differs from [10, 14] in several ways. Unlike the algorithms in [10, 14], 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 [13]:

1. Let $L^\perp$ denote the orthogonal complement of $L$. Algorithm 1 finds most interior solutions to the problems

$$\text{find } x \in L \cap \mathbb{R}_+^n, \quad (6)$$
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 [13] and the algorithms in [3, 4, 10, 14], 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.

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 [13]: 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 [13], 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. 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 [13]. 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 [13]. 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 [2].
Proposition 1 Let $L \subseteq \mathbb{R}^n$ be a linear subspace. Then there exists a unique partition $B \cup N = \{1, \ldots, n\}$ such that

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

and

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

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

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

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 $L^+ \cap \mathbb{R}^n_+ \neq \emptyset$. 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 $D(L^+) \cap \mathbb{R}^n_+$ 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 $\hat{x} \in L^+$ satisfying (8). Otherwise, update the rescaling matrices $D$ and $\hat{D}$ and proceed to the next main iteration: Apply the basic procedure to $\hat{D}(L) \cap \mathbb{R}^n_+$ and $\hat{D}(L^+) \cap \mathbb{R}^n_+$, 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 $L^+ \cap \mathbb{R}^n_+$. More precisely, Algorithm 1 will terminate with points $x \in L$ and $\hat{x} \in L^+$ satisfying the following approximation of (8):

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

Algorithm 1 Enhanced Projection and Rescaling Algorithm (EPRA)

1. **(Initialization)**
   - Let $D := I$ and $\hat{D} := I$.
   - Let $P := P_L$ and $\hat{P} := P_{L^+}$.
   - Let $U > 0$ be a pre-specified upper bound on the rescaling matrices $D$ and $\hat{D}$.

2. **(Basic Procedure)**
   - Find $z \geq 0$ such that either $Pz > 0$ or $(Pz)^+||_1 \leq \frac{1}{2} ||z||_\infty$.
   - Find $\hat{z} \geq 0$ 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 $B := \{i : |x_i| < \frac{1}{U} ||x||_\infty\}$ and $N := \{i : |x_i| < \frac{1}{U} ||x||_\infty\}$.

4. if $(B, N)$ partitions $\{1, \ldots, n\}$ then
   - HALT and return $x \in \text{ri}(L \cap \mathbb{R}^n_+)$, $\hat{x} \in \text{ri}(L^+ \cap \mathbb{R}^n_+)$ end if

5. **(Rescaling step)**
   - Let $e := (z/(Pz)^+||_1 - 1)^+$, $D := \min((I + \text{diag}(e))D, U)$ and $P := P_{D(L)}$.
   - Let $\hat{e} := (\hat{z}/(\hat{P}\hat{z})^+||_1 - 1)^+$, $\hat{D} := \min((I + \text{diag}(\hat{e}))\hat{D}, U)$ and $\hat{P} := P_{D(L^+)}$.
   - 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 [13]. Algorithm 2 describes the simplest of these schemes, namely the perceptron scheme. In the algorithms below \( \Delta_{n-1} \) 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
   3. Pick \( u \in \Delta_{n-1} \) such that \( \langle u, Pz_t \rangle \leq 0 \).
   4. Let \( z_{t+1} := (1 - \frac{1}{t+1}) z_t + \frac{1}{t+1} u \).
   5. \( t := t + 1 \).
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

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
   3. Let \( u = u(Pz_t) \) and \( z_{t+1} := z_t + \theta (u - z_t) \) where
      \[ \theta_t = \operatorname{argmin}_{\theta \in [0, 1]} \|P(z_t + \theta(u - z_t))\|_2^2 = \frac{\|Pz_t\|_2^2 - \langle u, Pz_t \rangle}{\|Pz_t\|_2^2 + \|Pu\|_2^2 - 2 \langle u, Pz_t \rangle} \]
   4. \( t := t + 1 \).
3. 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

\[ v(z) := \operatorname{argmax}_{\substack{u \in \Delta_{n-1} \\ S(u) \subseteq S(z) \\ \langle v, Pu \rangle \geq \langle v, Pz \rangle}} \langle u, Pz \rangle \]
Algorithm 4 Von Neumann with Away Steps Scheme

1. Pick \( z_0 \in \Delta_{n-1} \), and \( t := 0 \).
2. while \( Pz_t \neq 0 \) and \( \| (Pz_t)^+ \|_1 > \epsilon \| z_t \|_\infty \) do
   Let \( u = u(Pz_t) \) and \( v = v(z_t) \).
   if \( \| Pz_t \| - \langle u, Pz_t \rangle > \langle v, Pz_t \rangle - \| Pz_t \|^2 \) then (regular step)
      \( a := u - z_t; \quad \theta_{\text{max}} = 1 \),
   else (away step)
      \( a := z_t - v; \quad \theta_{\text{max}} = \frac{\langle v, z \rangle}{1 - \langle v, z \rangle} \),
   endif
   Let \( z_{t+1} := z_t + \theta a \) where
   \[
   \theta = \arg\min_{\theta \in [0, \theta_{\text{max}}]} \| P(z_t + \theta a) \|^2 = \min \left\{ \theta_{\text{max}}, \frac{\langle z_t, P a \rangle}{\| P a \|^2} \right\}
   \]
   \( t := t + 1 \).
3. end while

Algorithm 5 Smooth Perceptron Scheme

1. Let \( u_0 := \bar{u}; \mu_0 = 2; \) and \( t := 0 \).
2. while \( Pz_t \neq 0 \) and \( \| (Pz_t)^+ \|_1 > \epsilon \| z_t \|_\infty \) do
   \( \theta_t := \frac{2}{t+3} \)
   \( u_{t+1} := (1 - \theta_t)u_t + \theta_t z_t + \theta_t^2 u_{\mu_t}(Pz_t) \)
   \( \mu_{t+1} := (1 - \theta_t)\mu_t \)
   \( t := t + 1 \).
3. 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 half the dimension of the ambient space \( \mathbb{R}^n \), a naive approach to generate random instances yields results of limited interest. In most cases such an approach provides instances that
are easily solvable by any of the basic procedure schemes without any rescaling. 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 \( A = \begin{bmatrix} a_1 & \cdots & a_m \end{bmatrix}^T \in \mathbb{R}^{m \times n} \) be such that \( a_1 = \bar{u} - \bar{X}^{-1}1 \) 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

\[
\bar{x} = \arg\max_x \left\{ \prod_{j=1}^n x_j : x \in L \cap \mathbb{R}^n_+, \|x\|_\infty = 1 \right\}.
\]

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

\[
\bar{x} = \arg\max_x \left\{ \ln \left( \prod_{i=1}^n x_i \right) : x \in L \cap \mathbb{R}^n_+, \|x\|_\infty = 1 \right\} = \arg\max_x \left\{ \ln \left( \prod_{i=1}^n x_i \right) : x \in L \cap \mathbb{R}^n_+, \|x\|_\infty \leq 1 \right\}.
\]

(9)

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 (9). Since \( \zeta = \bar{X}^{-1}\mathbf{1} \) is the gradient of the objective function in (9) 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 (9).

Take \( x \in L \cap \mathbb{R}^n_+ \) with \( \|x\|_\infty \leq 1 \). Since \( x \in L \), we have \( \langle \bar{X}^{-1}\mathbf{1} - \bar{u}, x \rangle = \langle a_1, x \rangle = 0 \). Therefore \( \langle \zeta, x - \bar{x} \rangle = \langle \bar{X}^{-1}\mathbf{1}, x \rangle - n \leq \langle \bar{u}, x \rangle - \|\bar{u}\|_1\|x\|_\infty \leq 0 \). The last two steps follow from \( \|\bar{u}\|_1 = n \) and Hölder’s inequality respectively. \( \blacksquare \)

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

\[
A = \begin{bmatrix} A_{BB} & 0 \\ A_{NB} & A_{NN} \end{bmatrix}
\]

(10)

is such that \( L_B = \ker(A_{BB}) \subseteq \mathbb{R}^B \) and \( L_N = \text{Im}(A_{NN}^T) \subseteq \mathbb{R}^N \) satisfy \( L_B \cap \mathbb{R}^n_+ \neq \emptyset \) 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 \( L^\perp = \text{Im}(A^T) \) satisfy

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

and

\[
\text{ri} (L^\perp \cap \mathbb{R}^n_+) = \{ \hat{x} \in L^\perp \cap \mathbb{R}^n_+ : \hat{x}_i > 0 \text{ for all } i \in N \}.
\]
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 \( \text{Im}(A_{NN}) \cap \mathbb{R}^N_+ \neq \emptyset \). Finally let \( L = \ker(A) \) where \( A \) is of the form (10) 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 \( n = 1000 \) and \( m = 100, 200, 800, 900 \). 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 \( \epsilon \): \( \epsilon = 10^{-1} \) (large), and \( \epsilon = 10^{-4} \) (small).

**Table 1: Results for naive random instances and large \( \epsilon \) (\( \epsilon = 10^{-1} \))**

<table>
<thead>
<tr>
<th>( m )</th>
<th>( n )</th>
<th>perceptron</th>
<th>VN</th>
<th>VNA</th>
<th>smooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>200</td>
<td>(6956.28, 0.27, 0.74)</td>
<td>(5070.41, 0.26, 0.69)</td>
<td>(3021.73, 0.23, 0.95)</td>
<td>(27.04, 0.03, 1)</td>
</tr>
<tr>
<td>250</td>
<td>500</td>
<td>(9963.91, 0.85, 0.02)</td>
<td>(9207.1, 0.26, 0.2)</td>
<td>(8737.9, 0.23, 0.38)</td>
<td>(43.88, 0.13, 1)</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>(10000, 8.67, 0)</td>
<td>(9981.29, 8.84, 0.01)</td>
<td>(9992.46, 14.3, 0.01)</td>
<td>(58.50, 0.42, 1)</td>
</tr>
<tr>
<td>1000</td>
<td>2000</td>
<td>(10000, 34.72, 0)</td>
<td>(10000, 35.54, 0)</td>
<td>(10000, 67.24, 0)</td>
<td>(80.21, 1.42, 1)</td>
</tr>
</tbody>
</table>
Table 2: Results for naive random instances and small $\epsilon$ ($\epsilon = 10^{-4}$)

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>perceptron</th>
<th>VN</th>
<th>VNA</th>
<th>smooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>200</td>
<td>(8236.24, 0.33, 0.35)</td>
<td>(5395.12, 0.28, 0.67)</td>
<td>(5861, 0.45, 0.66)</td>
<td>(123.62, 0.14, 1)</td>
</tr>
<tr>
<td>250</td>
<td>500</td>
<td>(9981.92, 0.94, 0.01)</td>
<td>(9258.12, 0.99, 0.2)</td>
<td>(9518.46, 1.64, 0.16)</td>
<td>(231.84, 0.64, 1)</td>
</tr>
<tr>
<td>500</td>
<td>1000</td>
<td>(10000, 8.28, 0)</td>
<td>(9973.7, 8.39, 0.01)</td>
<td>(9988.46, 13.57, 0.01)</td>
<td>(337.64, 2.15, 1)</td>
</tr>
<tr>
<td>1000</td>
<td>2000</td>
<td>(10000, 35.61, 0)</td>
<td>(10000, 36.34, 0)</td>
<td>(10000, 68.71, 0)</td>
<td>(465.94, 7.87, 1)</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>$m$</th>
<th>$n$</th>
<th>perceptron</th>
<th>VN</th>
<th>VNA</th>
<th>smooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>1000</td>
<td>(9939.49, 8.80, 0.01)</td>
<td>(9915.70, 16.94, 0.01)</td>
<td>(9939.14, 16.94, 0.01)</td>
<td>(7537.20, 14.85, 0.97)</td>
</tr>
<tr>
<td>200</td>
<td>1000</td>
<td>(9964.52, 8.94, 0)</td>
<td>(9961.16, 17.15, 0)</td>
<td>(9967.60, 17.15, 0)</td>
<td>(8557.46, 17.05, 0.75)</td>
</tr>
<tr>
<td>800</td>
<td>1000</td>
<td>(9952.62, 8.74, 0.01)</td>
<td>(9942.13, 16.95, 0.01)</td>
<td>(9950.09, 16.95, 0.01)</td>
<td>(9579.90, 19.06, 0.13)</td>
</tr>
<tr>
<td>900</td>
<td>1000</td>
<td>(9961.86, 8.26, 0)</td>
<td>(9926.91, 16.06, 0.01)</td>
<td>(9934.59, 16.01, 0.01)</td>
<td>(9463.48, 17.65, 0.16)</td>
</tr>
</tbody>
</table>

When $\epsilon$ is large (Table 1 and Table 3), the algorithms often stop when the condition $\|(Pz)^+\|_1 \leq \epsilon$ is satisfied. Not surprisingly, when $\epsilon$ 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 $\epsilon$ 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 the perceptron, von Neumann, and von Neumann with away steps are comparable in terms of number of iterations, CPU time, and success rate. 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 $\epsilon$ 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.
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 $\text{ri}(L \cap \mathbb{R}^n_{++}) \neq \emptyset$ and $\text{ri}(L^\perp \cap \mathbb{R}^n_{++}) \neq \emptyset$. 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 5 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 5: Performance of Algorithm 1 on instances with $L \cap \mathbb{R}^n_{++} \neq \emptyset$

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

Table 6 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 instances $L$ with $L \cap \mathbb{R}^n_{++} \neq \emptyset$.

Table 6: Algorithm 1 on instances with $ri(L \cap \mathbb{R}^n_{+}) \neq \emptyset$ and $ri(L^\perp \cap \mathbb{R}^n_{+}) \neq \emptyset$

<table>
<thead>
<tr>
<th>$n$</th>
<th>Average $m$</th>
<th>Average # of rescaling iterations</th>
<th>Average total # of iterations</th>
<th>Average CPU time (in seconds)</th>
<th>Success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>49.79</td>
<td>16.16</td>
<td>657.01</td>
<td>0.075</td>
<td>0.946</td>
</tr>
<tr>
<td>200</td>
<td>99.45</td>
<td>17.27</td>
<td>1031.45</td>
<td>0.12</td>
<td>0.974</td>
</tr>
<tr>
<td>500</td>
<td>251.36</td>
<td>17.60</td>
<td>1763.59</td>
<td>1.09</td>
<td>0.984</td>
</tr>
<tr>
<td>800</td>
<td>394.14</td>
<td>17.72</td>
<td>2269.69</td>
<td>3.52</td>
<td>0.998</td>
</tr>
<tr>
<td>1000</td>
<td>500.28</td>
<td>17.76</td>
<td>2625.84</td>
<td>6.73</td>
<td>0.994</td>
</tr>
<tr>
<td>2000</td>
<td>993.07</td>
<td>17.59</td>
<td>3752.04</td>
<td>47.07</td>
<td>0.994</td>
</tr>
</tbody>
</table>

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 instances. The 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 $\text{ri} (L \cap \mathbb{R}^n_+ ) \neq \emptyset$ and $\text{ri} (L^\perp \cap \mathbb{R}^n_+ ) \neq \emptyset$.

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 [13]. 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 $D = \min \left( (I + \text{diag}(e_i)) D, U \right)$ and $\hat{D} = \min \left( (I + \text{diag}(e_j)) \hat{D}, U \right)$ where $i$ and $j$ are such that $z_i = \|z\|_\infty$ 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 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 [10, 14]. Consistent with the reported results in [10, 14], 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 [10, 14] and as in the first set of experiments summarized in Table 1 and Table 2. 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. This sharp difference can be explained upon a closer look at the performance of Algorithm 1: The naively generated instances are easier and can usually be solved within one single round of basic procedure and without the need for rescaling. By contrast, instances with controlled condition measure, such as those in the set of experiments summarized in Table 5, are significantly more challenging and require on average ten or more rounds of basic and rescaling steps.
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.

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
Acknowledgements

This research has been funded by NSF grant CMMI-1534850.

References


