Carnegie Mellon
Tepper
SCHOOL OF BUSINESS

# DISSERTATION

*Submitted in partial fulfillment of the requirements for the degree of*

**DOCTOR OF PHILOSOPHY**
**INDUSTRIAL ADMINISTRATION**
**(OPERATIONS RESEARCH)**

*Titled*
**"ELEMENTARY ALGORITHMS FOR SOLVING CONVEX OPTIMIZATION PROBLEMS"**

*Presented by*
**Negar Soheili Azad**

***Accepted by***

Javier F. Peña

**Chair: Prof. Javier Peña**

April 25/2014
**Date**

***Approved by The Dean***

Robert M. Dammon

**Dean Robert M. Dammon**

5/2/14
**Date**

# Elementary Algorithms for Solving Convex Optimization Problems

by

Negar Soheili Azad

A thesis
presented to the Carnegie Mellon University
in partial fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Operations Research

Pittsburgh, PA, USA, 2014

## Abstract

The rapid growth in data availability has led to modern large scale convex optimization problems that pose new practical and theoretical challenges. Examples include classification problems such as customer segmentation in retail and credit scoring in insurance. Classical optimization and machine learning techniques are typically inadequate to solve these large optimization problems because of high memory requirements and slow convergence guarantees.

This thesis develops two research threads to address these issues. The first involves improving the effectiveness of a class of algorithms with simple computational steps for solving convex optimization problems arising in machine learning, data mining, and decision making. The second involves the refinement of conditioning and geometry of convex optimization problems via preconditioning. I elaborate on these two threads below.

1. The main theme of this thesis focuses on the class of *elementary algorithms* for solving convex optimization problems. These algorithms only involve simple operations such as matrix-vector multiplications, vector updates, and separation oracles. This simplicity makes the computational cost per iteration and memory requirement of these algorithms low. Thus, elementary algorithms are promising for solving emerging big data applications in areas such as classification, pattern recognition, and online learning. A major hurdle that needs to be overcome is the slow convergence of these algorithms. We develop new elementary algorithms that are enhanced via smoothing and dilation techniques. These enhancements yield algorithms that retain the attractive simplicity of the elementary algorithms while achieving substantially improved convergence rates. Thus, these enhanced algorithms are better suited for solving modern large scale convex optimization problems.

2. A significant difficulty when solving large convex optimization problems is poor conditioning caused by the existence of flat and nearly flat geometries. This thesis shows that a combination of two simple preprocessing steps generally improve the geometric structure of problem instances. We improve instances' geometric structure by reconciling the properties of three different but related notions of conditioning. More precisely, when one of these measures is large, in which case the problem instance is certainly poorly conditioned, our procedure reduces it without making the remaining measures worse. Our preconditioning procedures can be potentially useful for the convergence properties of a large class of iterative methods without changing their ultimate solution.

**Acknowledgements**

I would like to express the deepest appreciation to my advisor Javier Pena for his kindness, continuous support and motivation throughout every step of my PhD, and for all he has taught me. He has patiently helped me mature as a researcher. This thesis would not be possible without him. I would also like to thank my committee members Lenore Blum, Gérard Cornuéjols, and Tamás Terlaky for their time and support.

My PhD life was made more enjoyable with interactions with my amazing friends: Zahra Afjeh, Ishani Aggarwal, Hamid Akhlaghi, Amitabh Basu, David Bergman, Elvin Coban, Andre Cire, Xin Fang, Sam Hoda, Alex Kazachkov, Qihang Lin, Carla Michini, Marco Molinaro, Selvaprabu Nadarajah, Andrea Qualizza, Zeynab Saeedian, Amin Sayedi, Nandana Sengupta, Vince Slaugh, Tala Talai, Somayeh Vojdani, and Jessie Wang. Andre Cire and Selvaprabu Nadarajah made the best colleagues to navigate the PhD program. Working together with them greatly facilitated my learning.

I owe a special thank to Lawrence Rapp for being the best assistant director. He always goes above and beyond to make the life of PhD students easier. I have benefited tremendously from his help.

I would like to gratefully thank my family and extended family for their unconditional love and support; especially my parents, Nahid Nourjah and Aliakbar Soheili Azad, who have done everything for me to enjoy life and reach my goals. I must also thank my aunt, Parvaneh Nourjah, and my cousin, Pantea Alirezazadeh, for their encouragement throughout this journey.

Finally, and above all, I cannot begin to express my boundless gratitude and love to my husband, Selvaprabu Nadarajah who has constantly supported me throughout my PhD and has always taken care of me when things seemed onerous. He is the best partner and soul-mate that I could wish for. I would have been lost without him. Thank you.

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Motivation

The abundance of new technologies has given rise to unprecedentedly large amounts of information. A deluge of data creates new challenges across all sectors of industries such as retail, healthcare, and energy. Classification and signal processing problems are at the heart of applications that have appeared in recent years. These data rich problems can be formulated as large convex programs. More generally, convex optimization is a fundamental tool for solving emerging large applications arising at the intersection of machine learning, operations research and many other areas.

However, classical optimization algorithms are not designed to scale to instances of large size. For example, popular algorithms such as interior point methods and quasi-Newton methods become impractical when solving large scale convex optimization problems since their memory requirement and computational effort per iteration grows nonlinearly with growth of the problem's dimension. This phenomenon imposes limitations on the sizes of problems which can be processed by these algorithms. This suggests that new approaches for solving large scale optimization problems are needed.

The primary motivation for this thesis is to design algorithms that are suitable for solving large scale optimization problems. Therefore, we have focused on a class of *elementary algorithms* which have always played an important role in optimization and machine learning literature. These algorithms are closely related to first-order algorithms that are currently the most popular class of methods to deal with large-scale problems [13, 27, 41]. Elementary algorithms have low computational cost per iteration, low memory requirements, and can deal with large problems in an online fashion. These attractive features make elementary algorithms a promising class of methods for problems emerging in big data applications. However, a major hurdle that has to be overcome is that these algorithms suffer from slow convergence rates. This thesis aims to discuss several approaches for accelerating elementary algorithms for solving large optimization problems via smoothing techniques, space dilation and adaptive problem preconditioning.

## 1.2 Contributions and results

A substantial part of this thesis focuses on developing algorithms with low memory requirement and simple computational steps for solving *convex feasibility problems.* Convex feasibility problems are fundamental in optimization because any convex optimization problem can be written in this form via suitable optimality conditions. Alongside with the design of new algorithmic techniques, this thesis elucidate some geometric properties of convex optimization problems via refinement of concept of conditioning. We summarize the contributions of this thesis in the following points.

- **Accelerating elementary algorithms via smoothing**
  The perceptron algorithm [59] is a greedy elementary algorithm for finding a solution to a finite set of linear inequalities, $A^T y > 0$ where $A$ is a matrix in $\mathbb{R}^{m \times n}$. This algorithm was introduced in the late fifties in the machine learning community for solving classification problems. It is known [15, 50] that the perceptron algorithm requires $\mathcal{O}(1/\rho(A)^2)$ iterations to find a feasible solution, where the parameter $\rho(A)$ measures the thickness of the cone of feasible solutions $\{y : A^T y > 0\}$ [18, 21, 57]. We develop a modified perceptron algorithm that can find a solution with only $\mathcal{O}(\sqrt{\log(n)}/\rho(A))$ iterations [64]. This improvement is based on viewing the perceptron algorithm as a first-order algorithm for solving an optimization problem that is a reformulation of the feasibility problem $A^T y > 0$. This key interpretation and a subsequent refinement via a smoothing technique [47, 49], proposed by Nesterov, enable the convergence rate improvement by making only minor modifications to the original algorithm, thus retaining most of the perceptron algorithm's original simplicity. Our numerical experiments demonstrate that both the number of iterations and total CPU time required by the smooth perceptron algorithm are indeed lower than those required by the classical perceptron algorithm. Moreover, the observed relationship between any one of these metrics for the smoothed and classical perceptron algorithms are related in a strikingly consistent manner with their theoretically predicted relationship. The paper based on this chapter was published in the SIAM Journal on Optimization in 2011.

- **A primal-dual elementary algorithm**
  The von-Neumann algorithm [29], which can be seen as a dual of the perceptron algorithm, is another well-known elementary algorithm that solves the alternative system to $A^T y > 0$, that is, $Ax = 0$, $x \geq 0$, $x \neq 0$. The von-Neumann algorithm finds an $\epsilon$-solution to this system in at most $\mathcal{O}(1/\rho(A)^2 \cdot \log(1/\epsilon))$ iterations [31] where $\rho(A)$ here is the distance of zero from the convex-hull of normalized columns of $A$. We improve this complexity bound by proposing a primal-dual algorithm that can solve both primal and dual feasibility problems [65]. Our new algorithm is a combination of perceptron and von-Neumann algorithms. It finds an $\epsilon$-solution to $Ax = 0$, $x \geq 0$, $x \neq 0$, in at most $\mathcal{O}(\sqrt{n}/\rho(A) \cdot \log(1/\epsilon))$ elementary iterations or finds a solution to the alternative system $A^T y > 0$ in at most $\mathcal{O}(\sqrt{n}/\rho(A) \cdot \log(1/\rho(A)))$ elementary iterations. we show that this algorithm can be extended for solving

general conic systems with similar complexity results. The paper based on this chapter was published in the 2013 Fields Institute Communication monograph.

- **Near classification for unclassifiable data sets**
  As previously mentioned, the perceptron algorithm is a popular algorithm to solve classification problems in machine learning. This algorithm and our modifications of it find a linear separator under the assumption that the data is linearly separable. However, in many real-world applications data may not be linearly separable. Support Vector Machine (SVM) introduced by Vapnik et al. [26, 69, 70] is a very popular classification model that finds the "optimal" hyperplane to separate a dataset. The optimal hyperplane is chosen so that the distance to the nearest data point of each classes is maximized and misclassifications are penalized by the penalty parameter $1/\lambda$. Due to the popularity of the SVM, various algorithms are developed to solve this problem. For example, Newton method or Quasi-Newton method efficiently find an $\epsilon$-solution to the SVM in $\mathcal{O}\left(\sqrt{n}\log(1/\epsilon)\right)$ [71]. However, when the number of samples is large, these methods are impractical since the space and the time costs of Hessian matrix computation rapidly increases with the dimensions of the problem instances. Pegasos algorithm [62] is a well-known first-order algorithm that solves large scale SVM problem in $\mathcal{O}\left(\|A\|/\lambda\epsilon\right)$ iterations. We show that a slightly modified version of our smooth perceptron algorithm improves Pegasos algorithm's complexity bound and finds an $\epsilon$-solution to the SVM problem in $\mathcal{O}\left(\sqrt{n}\|A\|/\sqrt{\lambda\epsilon}\right)$ iterations.

- **Achieving exponential improvements in complexity via deterministic rescaling**
  A main contribution of this thesis is a variant of the perceptron algorithm to solve the feasibility problem $y \in F$ where $F$ is a general convex cone. The new algorithm uses a novel periodic rescaling idea to find a solution in at most $\mathcal{O}(m^5 \log(1/\tau_F))$ perceptron updates where $\tau_F$ measures the thickness of the cone $F$. When $F := \left\{y : A^T y > 0\right\}$, $\tau_F$ is equivalent to $\rho(A)$ already defined and this algorithm is polynomial in the bit-length representation of $A \in \mathbb{Z}^{m\times n}$. Our algorithm has the best deterministic complexity result among elementary algorithms in the literature. Aside from its theoretical merits, this algorithm provides a solid foundation for potential speed ups in the convergence of the widely popular first-order methods for large-scale convex optimization, since the perceptron algorithm and first-order methods are closely related. The paper based on this chapter is under first round of revision and will likely be accepted for publication in Mathematical Programming.

- **Elementary online learning algorithms**
  Learning algorithms are procedures for making predictions based on data. Established learning algorithms focus on constructing a predictive model efficiently given the full data set. However, in many real-life scenarios, data arrives serially over time and learning has to adapt to the availability of new data. Online learning algorithms construct predictive models and specialize in updating these models as data is received in an online fashion. For example, when the predictive model is a classifier, an online learning algorithm will be scheme for efficiently updating the current classifier

to a new one that accounts for new data point that is received. The objective of online learning algorithms is to minimize prediction errors.

The classical perceptron algorithm and our enhancement of this algorithm, namely the rescaled perceptron algorithm are examples of online learning algorithms. As mentioned above, the perceptron and the rescaled perceptron algorithms find a solution to $A^{\mathrm{T}}y > 0$ after making no more than $\mathcal{O}\left(1/\rho(A)^2\right)$ and $\mathcal{O}\left(m^5 \log(1/\rho(A))\right)$ mistakes, respectively. An interesting question is whether the complexity bound of the rescaled perceptron algorithm is tight. As a first step to answer this question, we study the lower bound complexity of online algorithms. We find that any online algorithms has a lower bound of $\mathcal{O}\left(m \log(1/\rho(A)\right)$ on the number of mistakes, which is substantially smaller than the complexity bound of our rescaled perceptron algorithm. This result motivates future research into closing this gap by either designing elementary online learning algorithms with better worst case complexity or tightening the lower bound complexity.

- **Preconditioning systems of linear inequalities**
  An important challenge when using an iterative method to solve a linear system of equations is to precondition a given problem instance [37, 67]. Preconditioning is a data processing operation that transforms a given instance into an equivalent but better conditioned one that is easier to solve. We show that a combination of two simple preprocessing steps would generally improve the conditioning of a homogeneous system of linear inequalities. Our approach is based on a comparison among three different but related notions of conditioning for linear inequalities, namely: Renegar's [57], Goffin-Cucker-Cheung's [21, 36], and the Grassmann condition number [5, 11]. More precisely, we show that when one of these measures is large, in which case the problem instance is certainly poorly conditioned, our procedure reduces it without making the remaining measures worse. The paper based on this chapter is published in Optimization Letters in 2014.

## 1.3 Outline of the thesis

We present the above results in the subsequent chapters of this thesis. In Chapter 2, we present some basic background required for the following chapters. In particular, we describe the perceptron and von Neumann algorithms and review some classical results on the performance of these algorithms. In Chapters 3 and 4, we present our modifications of the perceptron and the von Neumann algorithms for solving convex feasibility problems. Our modifications are based on smoothing techniques proposed in [47, 49]. Chapter 5 extends our modified algorithm for solving support vector machine. We describe our rescaled version of the perceptron algorithm in Chapter 6 and show this algorithm belongs to the class of online learning algorithms. We also provide a lower bound complexity for online learning algorithms. Chapter 7 addresses some preconditioning procedures to refine the geometry of a system of linear inequalities. Finally, we provide a discussion of future research directions in Chapter 8.

# Chapter 2

# Background

Throughout this thesis, we study the following generic convex feasibility problem

$$\text{Find } y \in F. \tag{2.1}$$

Here $F \subseteq \mathbb{R}^m$ is a convex set with an available *separation oracle*, that is given a test point $y \in \mathbb{R}^m$, the separating oracle either certifies that $y \in F$ or else it finds a hyperplane separating $y$ from $F$, that is, $u \in \mathbb{R}^m, b \in \mathbb{R}$ such that $\langle u, y\rangle \leq b$ and $\langle u, v\rangle > b$ for all $v \in F$.

A separation oracle for $F$ is readily available for the following common specifications of (2.1):

- Linear programming:

$$F := \left\{y \in \mathbb{R}^m : A^{\mathrm{T}} y \geq b\right\}, \text{ where } A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^n.$$

- Conic programming:

$$F := \left\{y \in \mathbb{R}^m : \sum_{j=1}^{n} A_j y_j - B \succeq 0\right\}, \text{ where } A_1, \ldots, A_n, B \in \mathbb{R}^{n\times n} \text{ are symmetric.}$$

Convex feasibility problem (2.1) is fundamental, since any convex optimization problems can be recast in this form via suitable optimality conditions.

Elementary algorithms are iterative algorithms that solve the problem (2.1) by only making calls to a separation oracle for $F$ and applying simple operations. They thus have low computational cost per iteration and low memory requirements. Therefore, elementary algorithms are a promising class of methods for problems emerging in big data applications. However, these algorithms have slow convergence rates.

The relaxation method, introduced in the classical articles of Agmon [1], and Motzkin and Schoenberg [45], is an elementary algorithm for solving convex feasibility problems (2.1). The relaxation method starts with an arbitrary initial trial solution. At each iteration, the algorithm queries the separation oracle for $F$ at the current trial solution $y$. If $y \in F$ then the algorithm terminates. Otherwise, the algorithm generates a new trial point $y_+ = y + \eta u$ for some step length $\eta > 0$ where $u \in \mathbb{R}^m, b \in \mathbb{R}$ determine a hyperplane separating $y$ from $F$.

There are numerous papers in the optimization literature related to various versions of the relaxation method [3, 4, 8, 9, 24]. However, the algorithms proposed in this thesis are based on two popular relaxation methods, namely the perceptron and the von Neumann algorithms. The perceptron algorithm is a simple greedy algorithm that was introduced by Rosenblatt [59] in 1958 for solving the polyhedral feasibility problem $A^{\mathrm{T}}y > 0$. As it was noted by Belloni, Freund, and Vempala [12], this algorithm readily extends to the more general feasibility problem (2.1) provided a special separation oracle for $F$ is available.

The perceptron algorithm has played major roles in machine learning and in optimization. The perceptron algorithm has attractive properties concerning noise tolerance [17]. It is also closely related to large-margin classification [34] and to the highly popular and computationally effective Pegasos algorithm [62] for training support-vector machines.

The von Neumann algorithm, which can be seen as a dual of the perceptron algorithm [44], was privately communicated by von Neumann to Dantzig in the late 1940s, and later studied by Dantzig [29]. It is also a simple greedy algorithm that finds an approximate solution to $Ax = 0$, $x \geq 0$, $x \neq 0$. This algorithm is closely related to the Frank-Wolfe algorithm [38]. We describe the perceptron and von Neumann algorithms in more details in Section 2.1.

The convergence rate of both the perceptron and von Neumann algorithms is determined by the parameter $\rho(A)$, where $\rho(A)$ is a certain *radius of well-posedness* of the matrix $A$ [21] (see Section 2.1 for more details), and is a natural parameter for studying the feasibility problems in the real number model of computation [18].

Block [15] and Novikoff [50] showed that the perceptron algorithm finds a solution to $A^{\mathrm{T}}y > 0$ in at most $\mathcal{O}(1/\rho(A)^2)$ perceptron updates. Belloni et al. [12, Lemma 3.1] extended this result to solve the general conic feasibility problem.

Given $\epsilon > 0$, we say that $x$ is an $\epsilon$-solution to $Ax = 0$, $x \geq 0$, $x \neq 0$ if $x \geq 0$, $\|x\|_1 = 1$ and $\|Ax\| \leq \epsilon$. Under the assumption that the columns of $A$ have Euclidean norm one, Dantzig [29] showed that von Neumann algorithm finds an $\epsilon$-solution to $Ax = 0$, $x \geq 0$, $x \neq 0$ in at most $\frac{1}{\epsilon^2}$ iterations when it is feasible. Epelman and Freund [31] showed that von Neumann algorithm either computes an $\epsilon$-solution to $Ax = 0$, $x \geq 0$, $x \neq 0$ in $\mathcal{O}(\frac{1}{\rho(A)^2}\log(\frac{1}{\epsilon}))$ iterations when this problem is feasible, or finds a solution to the alternative system $A^{\mathrm{T}}y > 0$ in $\mathcal{O}(\frac{1}{\rho(A)^2})$ iterations. They also extend these results for more general conic feasibility problems.

The perceptron and von Neumann algorithms are not polynomial in the bit model of computation because the quantity $\rho(A)$ can be exponentially small in the bit length description of an input matrix $A$ with rational entries. Although the perceptron and von Neumann algorithms have slow rate of convergence, the simplicity of their iterations makes them attractive. A main focus of this thesis is to improve the convergence rate of these algorithms while maintaining their simplicity that sometimes result in exponential improvements in theoretical convergence guarantees.

## 2.1 The perceptron and von Neumann algorithms

We next recall the classical perceptron and von Neumann algorithms to solve the *polyhedral feasibility problems*

$$Ax = 0, \;\; x \geq 0, \;\; x \neq 0, \tag{2.2}$$

and its alternative

$$A^{\mathrm{T}} y > 0, \tag{2.3}$$

where $A$ is a matrix in $\mathbb{R}^{m \times n}$. We refer to the systems (2.2) and (2.3) as *primal* and *dual* feasibility problems, respectively. The linear systems (2.2) and (2.3) are essentially alternative systems: either the system has a strict solution if and only if the other does not have a non-zero solution.

To solve (2.3), the perceptron algorithm starts with a trial point (usually zero). At each iteration, it updates the current trial point using the direction normal to one of the violated constraints (if any). This procedure is repeated until a solution to (2.3) is found.

For ease of notation we make the following assumption on the input matrix $A$ throughout this section. Let $\|\cdot\|$ denote the Euclidean norm in $\mathbb{R}^m$.

**Assumption 1** $A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} \in \mathbb{R}^{m \times n}$ where $\|a_j\| = 1$ for $j = 1, \ldots, n$.

Notice that the above assumption can be made without loss of generality since the matrix $A$ can always be pre-processed accordingly.

**Classical Perceptron Algorithm**

```
begin
    $y_0 := 0$;
    for $k = 0, 1, 2, \ldots$
        if $A^{\mathrm{T}} y_k > 0$ then Halt; $y_k$ is a solution to (2.3) fi
        $j := \operatorname*{argmin}_{i=1,\ldots,n} a_i^{\mathrm{T}} y_k$;
        $y_{k+1} := y_k + a_j$;
    end
end
```

We next introduce some convenient notation. Let $\Delta_n := \{x \geq 0 : \|x\|_1 = 1\}$, and for $y \in \mathbb{R}^m$, let $x(y) \in \Delta_n$ denote an arbitrary point in the set $\operatorname*{argmin}_{x \in \Delta_n} \langle A^{\mathrm{T}} y, x \rangle$. Observe that for a given $y \in \mathbb{R}^m$, we have $a_j^{\mathrm{T}} y = \min_i a_i^{\mathrm{T}} y$ if and only if $a_j = Ax(y)$. It follows that at iteration $k$ the classical perceptron algorithm generates an iterate $y_k$ with $y_k = Ax_k$ for some $x_k \geq 0$ with $\|x_k\|_1 = k$. This observation leads to the following normalized version of the perceptron algorithm.

**Normalized Perceptron Algorithm**

```
begin
    $y_0 := 0$;
    for $k = 0, 1, 2, \ldots$
        if $A^{\mathrm{T}} y_k > 0$ then Halt; $y_k$ is a solution to (2.3) fi
        $j := \operatorname{argmin}_{i=1,\ldots,n} a_i^{\mathrm{T}} y_k$;
        $\theta_k := \frac{1}{k+1}$;
        $y_{k+1} := (1 - \theta_k) y_k + \theta_k a_j$;
    end
end
```

It is easy to see that the $k$-th iterate generated by the normalized perceptron algorithm is exactly the same as the $k$-th iterate generated by the classical perceptron algorithm divided by $k$. In particular, the $k$-th iterate $y_k$ generated by the normalized perceptron algorithm satisfies $y_k = Ax_k$ for some $x_k \in \Delta_n$. Indeed, the main loop in the normalized perceptron algorithm can also be written in the following fashion to maintain both $y_k$ and $x_k$:

```
for $k = 0, 1, 2, \ldots$
    if $A^{\mathrm{T}} y_k > 0$ then Halt; $y_k$ is a solution to (2.3) fi
    $\theta_k := \frac{1}{k+1}$;
    $y_{k+1} := (1 - \theta_k) y_k + \theta_k Ax(y_k)$;
    $x_{k+1} := (1 - \theta_k) x_k + \theta_k x(y_k)$;
end
```

In contrast to the perceptron algorithm, von Neumann algorithm finds an approximate solution to the alternative system (2.2) and can be seen as a dual of the perceptron algorithm [44]. Von Neumann algorithm starts with an initial point $x_0 \geq 0$ such that $\|x_0\|_1 = 1$, and iteratively generates a sequence $x_1, x_2, \ldots$ such that $x_k \geq 0$, $\|x_k\|_1 = 1$ and $\|Ax_k\| \to 0$, provided (2.2) is feasible. The point $x_{k+1}$ is constructed as follows. First, identify the column $a_j$ of $A$ that forms the largest angle with $Ax_k$. Next, define $x_{k+1}$ so that $Ax_{k+1}$ is the point with smallest Euclidean norm along the segment joining $Ax_k$ and $a_j$.

Let $e \in \mathbb{R}^n$ denote the $n$-dimensional vector of all ones and $e_i \in \mathbb{R}^n$ denote the unitary vector whose $i$-th entry is equal to one and all others are equal to zero.

Assume $\epsilon > 0$ is a given input.

**Von Neumann Algorithm**($\epsilon$)

```
begin
    $x_0 := \frac{e}{n}$; $y_0 := Ax_0$;
    for $k = 0, 1, 2, \dots$
        if $A^{\mathrm{T}} y_k > 0$ then Halt // $y_k$ is a solution to (2.3) fi
        if $\|Ax_k\| < \epsilon$ then Halt // $x_k$ is an $\epsilon$-solution to (2.2) fi
        $j := \operatorname*{argmin}_{i=1,\dots,n} a_i^{\mathrm{T}} y_k$;
        $\theta_k := \operatorname*{argmin}_{\theta\in[0,1]} \{\|(1-\theta)y_k + \theta a_j\|\}$;
        $x_{k+1} := (1-\theta_k)x_k + \theta_k e_j$;
        $y_{k+1} := Ax_{k+1} = (1-\theta_k)y_k + \theta_k a_j$;
    end
end
```

The iterations in the above perceptron and von Neumann algorithms are similar. Each of these algorithms can be seen as a special case of the *Perceptron–von Neumann Template* below.

Assume $\epsilon > 0$ is a given input.

**Perceptron–von Neumann Template**($\epsilon$)

```
  begin
    $x_0 = \frac{e}{n}$; $y_0 := Ax_0$;
    for $k = 0, 1, 2, \dots$
        if $A^{\mathrm{T}} y_k > 0$ then Halt // $y_k$ is a solution to (2.3) fi
        if $\|Ax_k\| < \epsilon$ then Halt //$x_k$ is an $\epsilon$-solution to (2.2) fi;
        $x_{k+1} := (1-\theta_k)x_k + \theta_k x(y_k)$;
        $y_{k+1} := (1-\theta_k)y_k + \theta_k Ax(y_k)$;
    end
end
```

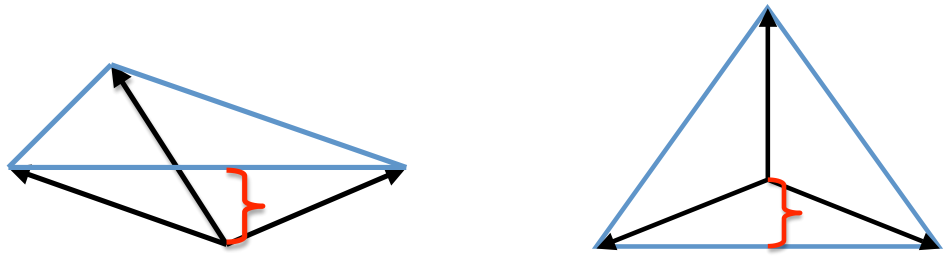
Observe that the above perceptron–von Neumann template recovers the normalized perceptron algorithm for $\theta_k := \frac{1}{k+1}$, and von Neumann algorithm for $\theta_k := \operatorname{argmin}_{\theta\in[0,1]} \|(1-\theta)y_k + \theta Ax(y_k)\|$ provided $x(y)$ is always chosen as one of the extreme points of the set $\operatorname{argmin}_{x\in\Delta_n} \langle A^{\mathrm{T}} y, x \rangle$.

The convergence rate of both the perceptron and von Neumann algorithms is determined by the following parameter $\rho(A)$.

$$\rho(A) := \left| \max_{\|y\|=1} \min_{j=1,\dots,n} \frac{a_j^{\mathrm{T}} y}{\|a_j\|} \right|. \tag{2.4}$$

When (2.3) is feasible, $\rho(A)$ is precisely the *width* of the feasibility cone $\{y : A^{\mathrm{T}} y \geq 0\}$, as defined by Freund and Vera [33]. Furthermore, when (2.2) is feasible, $\rho(A)$ is the Euclidean distance from the origin to the boundary of the convex hull of $\{a_1, \dots, a_n\}$. This geometric interpretation is illustrated in Figure 2.1 where the arrows depict the column vectors of $A$.

The parameter $\rho(A)$ is also a certain *radius of well-posedness* of the matrix $A$ as stated in Proposition 1.



(a) system (2.3) is feasible (b) system (2.2) is feasible

Figure 2.1: Geometric interpretation of $\rho(A)$.

**Proposition 1 (Cheung and Cucker [21])** *Let $\Sigma$ be the set of ill-posed instances, that is the set of matrices $A \in \mathbb{R}^{m\times n}$ such that arbitrary small perturbations makes both* (2.2) *and* (2.3) *have non-trivial solutions. Then*

$$\rho(A) = \min\left\{ \max_{1\le j\le n} \frac{\|a_j - \tilde{a}_j\|}{\|a_j\|} : \tilde{A} \in \Sigma \right\}.$$

Cheung and Cucker [21] define $1/\rho(A)$ as a condition number of the matrix $A$ in relation to the problems (2.2)–(2.3). The quantity $1/\rho(A)$ can also be seen as a special case of Renegar's condition number [57] for the systems (2.2)–(2.3). (See [19, 21, 57, 58] for further details.) The parameter $\rho(A)$ can be generalized [22] for the conic systems $Ax = 0$, $x \in \mathcal{K}$ and $A^{\mathrm{T}}y \in \mathcal{K}^*$, where $\mathcal{K}$ is a convex cone and $\mathcal{K}^*$ is the its dual cone.

**Theorem 1 (Block [15] and Novikoff [50])** *If the problem* (2.3) *is feasible, the perceptron algorithm finds a solution to $A^{\mathrm{T}}y > 0$ in at most*

$$\mathcal{O}\left(\frac{1}{\rho(A)^2}\right)$$

*iterations.*

Epelman and Freund showed a similar result for the von Neumann algorithm.

**Theorem 2 (Epelman and Freund [31])** *The von Neumann algorithm either finds an $\epsilon$-solution to* (2.2) *in at most*

$$\mathcal{O}\left(\frac{1}{\rho(A)^2}\log\left(\frac{1}{\epsilon}\right)\right)$$

*iterations when* (2.2) *is feasible, or determines infeasibility by finding a solution to the alternative system* (2.3) *in at most*

$$\mathcal{O}\left(\frac{1}{\rho(A)^2}\right)$$

*iterations.*

# Chapter 3

# Smooth Perceptron Algorithm

(Joint work with Javier Peña)

## 3.1 Introduction

In this chapter, we introduce a modified version of the perceptron algorithm, namely, *a smooth perceptron algorithm*, that solves the polyhedral feasibility problem $A^{\mathrm{T}}y > 0$. Our modification retains the perceptron's original simplicity but terminates in $\mathcal{O}(\sqrt{\log(n)}/\rho(A))$ iterations as compared to $\mathcal{O}(1/\rho(A)^2)$ for the classical perceptron algorithm. A key insight for our work is the observation that the perceptron algorithm can be seen as a first-order algorithm for a certain canonical optimization problem associated to the feasibility problem $A^{\mathrm{T}}y > 0$. Our approach is based on this interpretation and a subsequent refinement via a smoothing technique proposed by Nesterov [47, 49]. We note that similar improvements on convergence rate could be achieved via other accelerated first-order techniques such as the mirror-prox method of Nemirovski [46] or other accelerated first-order methods for saddle-point problems such as those described in [43, 68]. The approach that we have followed enables us to achieve the improvement on convergence rate with only minor modifications on the original algorithm. In particular, our modified algorithm retains most of the algorithm's original simplicity. Furthermore, although our approach is based on Nesterov's smoothing technique [49], we provide a succinct and self-contained proof of the algorithm's convergence rate. We run some numerical experiments to illustrate the behavior of the classical perceptron and the smooth perceptron algorithms. The experiments demonstrate that both the number of iterations and total CPU time required by the smooth perceptron algorithm are indeed lower than those required by the classical perceptron algorithm in a way that is commensurate with our main theoretical result.

This chapter is based on [64] published in SIAM Journal of Optimization.

## 3.2 Smooth perceptron algorithm

We next describe a new *smooth* version of the perceptron algorithm by tweaking the steps in the main loop of the normalized version. The smooth version has the improved convergence rate stated in Theorem 3 below.

Throughout this chapter, we assume the matrix $A$ satisfies Assumption 1. We start by considering the following *smooth* version of the map $y \mapsto x(y)$ defined in Chapter 2. Given $\mu > 0$ let $x_\mu : \mathbb{R}^m \to \Delta_n$ be defined as

$$x_\mu(y) := \frac{e^{\frac{-A^{\mathrm{T}}y}{\mu}}}{\|e^{\frac{-A^{\mathrm{T}}y}{\mu}}\|_1}. \tag{3.1}$$

In this expression we are using *vectorized* notation. In other words, $e^{\frac{-A^{\mathrm{T}}y}{\mu}}$ denotes the $n$-dimensional vector

$$e^{\frac{-A^{\mathrm{T}}y}{\mu}} := \begin{bmatrix} e^{\frac{-a_1^{\mathrm{T}}y}{\mu}} \\ \vdots \\ e^{\frac{-a_n^{\mathrm{T}}y}{\mu}} \end{bmatrix}.$$

Our smooth perceptron algorithm is as follows:

**Smooth Perceptron Algorithm:**

```
  begin
    $y_0 := \frac{Ae}{n}$;  $\mu_0 := 1$;  $x_0 := x_{\mu_0}(y_0)$;
    for $k = 0, 1, 2, \ldots$
      if $A^{\mathrm{T}}y_k > 0$ then Halt; $y_k$ is a solution to (2.3) fi
      $\theta_k := \frac{2}{k+3}$;
      $y_{k+1} := (1-\theta_k)(y_k + \theta_k A x_k) + \theta_k^2 A x_{\mu_k}(y_k)$;
      $\mu_{k+1} := (1-\theta_k)\mu_k$;
      $x_{k+1} := (1-\theta_k)x_k + \theta_k x_{\mu_{k+1}}(y_{k+1})$;
    end
end
```

We can now present our main result for the smooth perceptron algorithm.

**Theorem 3** *Assume $A \in \mathbb{R}^{m\times n}$ satisfies Assumption 1 and the problem* (2.3) *is feasible. Then the smooth perceptron algorithm terminates in at most*

$$\frac{2\sqrt{\log(n)}}{\rho(A)} - 1$$

*iterations.*

We present the proof of Theorem 3 in Section 3.4 below.

## 3.3 Perceptron algorithm as a first-order algorithm

A key observation that leads to our smooth perceptron algorithm is that the normalized perceptron algorithm can be seen as first-order algorithm for the canonical maximization problem (3.3) below associated to (2.3). The smooth perceptron algorithm in turn is a smooth first-order algorithm for (3.3). For ease of exposition, throughout this section we assume that the problem (2.3) is feasible. In this case the parameter $\rho(A)$ defined in (2.4) satisfies

$$\begin{aligned}\rho(A) &= \max_{\|y\|=1}\min_{i\le n} a_i^{\mathrm{T}} y \\ &= \max_{\|y\|\le 1}\min_{i\le n} a_i^{\mathrm{T}} y \\ &= \max_{\|y\|\le 1}\psi(y),\end{aligned} \tag{3.2}$$

where $\psi(y) := \min_{x\in\Delta_n} \left\langle A^{\mathrm{T}}y, x\right\rangle$.

Observe that a point $y \in \mathbb{R}^m$ with $\|y\| \le 1$ is a solution to (2.3) if and only if $\psi(y) > 0$, which in turn holds if and only if $\psi(y)$ is within $\rho(A)$ of its maximum on $\{y : \|y\| \le 1\}$.

Consider now the function $\varphi : \mathbb{R}^m \to \mathbb{R}$ defined as

$$\varphi(y) := -\frac{1}{2}\|y\|^2 + \psi(y) = -\frac{1}{2}\|y\|^2 + \min_{x\in\Delta_n}\left\langle A^{\mathrm{T}}y, x\right\rangle.$$

It readily follows that

$$\frac{1}{2}\rho(A)^2 = \max_{y\in\mathbb{R}^m}\varphi(y). \tag{3.3}$$

Since

$$\begin{aligned}\max_{y\in\mathbb{R}^m} -\frac{1}{2}\|y\|^2 + \psi(y) &= \max_{y\in\mathbb{R}^m} -\frac{1}{2}\|y\|^2 + \|y\|\psi\left(\frac{y}{\|y\|}\right) \\ &= \max_{\alpha>0} -\frac{1}{2}\alpha^2 + \alpha\rho(A) \\ &= \frac{1}{2}\rho(A)^2.\end{aligned}$$

Furthermore, a point $y \in \mathbb{R}^m$ solves (2.3) if $\varphi(y) > 0$, that is, if $\varphi(y)$ is within $\frac{1}{2}\rho(A)^2$ of its maximum.

Recall the main step in the normalized perceptron algorithm:

$$y_+ := (1-\theta)y + \theta Ax(y) = y + \theta(-y + Ax(y)).$$

Observe that $-y + Ax(y) \in \partial\varphi(y)$. Hence the normalized perceptron can be seen as a subgradient algorithm applied to the maximization problem (3.3).

## 3.4 Proof of the Theorem 3

The specific steps in the smooth perceptron algorithm as well as the proof of Theorem 3 are based on applying Nesterov's excessive gap technique [49] to (3.3) as we next explain. Consider the dual function

$$f(x) = \max_y \left\{ -\frac{1}{2}\|y\|^2 + \langle A^{\mathrm{T}}y, x\rangle \right\} = \frac{1}{2}\|Ax\|^2.$$

Next, for $\mu > 0$ define the smooth approximation $\varphi_\mu$ of $\varphi$ as:

$$\varphi_\mu(y) := -\frac{1}{2}\|y\|^2 + \min_{x\in\Delta_n}\{\langle A^{\mathrm{T}}y, x\rangle + \mu d(x)\},$$

where $d(x)$ is the entropy prox-function on $\Delta_n$, that is,

$$d(x) := \sum_{j=1}^{n} x_j \log x_j + \log n.$$

It is easy to see that the minimizer in the expression for $\varphi_\mu$ is precisely the point $x_\mu(y)$ defined in (3.1). Hence

$$\varphi_\mu(y) = -\frac{1}{2}\|y\|^2 + \langle A^{\mathrm{T}}y, x_\mu(y)\rangle + \mu d(x_\mu(y)).$$

Theorem 3 is a consequence of the following two lemmas.

**Lemma 1** *For any given $x \in \Delta_n$ and $y \in \mathbb{R}^m$, we have*

$$\varphi(y) \le \frac{1}{2}\rho(A)^2 \le f(x), \tag{3.4}$$

*and*

$$0 \le \varphi_\mu(y) - \varphi(y) \le \mu\log(n). \tag{3.5}$$

**Proof:** The inequalities in (3.4) readily follow from the definitions of $\varphi(y)$ and $f(x)$. On the other hand, the inequalities in (3.5) follow from the constructions of $\varphi$, $\varphi_\mu$, and the facts that $\min\limits_{x\in\Delta_n} d(x) = 0$, and $\max\limits_{x\in\Delta_n} d(x) = \log(n)$. ■

**Lemma 2** *The iterates $x_k \in \Delta_n$, $y_k \in \mathbb{R}^m$, $k = 0, 1, \ldots$ generated by the smooth perceptron algorithm satisfy the* excessive gap condition

$$f(x_k) \le \varphi_{\mu_k}(y_k). \tag{3.6}$$

Before we prove Lemma 2, recall that for $z, x \in \Delta_n$ the Bregman distance $h(z,x)$ is defined as $h(z,x) = d(z) - d(x) - \langle \nabla d(x), z - x\rangle$ and satisfies

$$h(z,x) \ge \frac{1}{2}\|z - x\|_1^2. \tag{3.7}$$

Observe also that by Assumption 1 we have

$$\|A\|_{1,2} = \max\{\|Ax\|_2 : \|x\|_1 = 1\} = \max\{\|a_1\|_2, \dots, \|a_n\|_2\} = 1.$$

**Proof of Lemma 2:** We proceed by induction. For $k = 0$ we have:

$$\begin{array}{rcl} f(x_0) & = & \frac{1}{2}\|Ax_0\|^2 \\ & \leq & \frac{1}{2}\|A\frac{\mathbf{1}}{n}\|^2 + \left\langle A\frac{\mathbf{1}}{n}, A\left(x_0 - \frac{\mathbf{1}}{n}\right)\right\rangle + \frac{1}{2}\|x_0 - \frac{\mathbf{1}}{n}\|_1^2 \\ & \leq & -\frac{1}{2}\|A\frac{\mathbf{1}}{n}\|^2 + \left\langle A^{\mathrm{T}}A\frac{\mathbf{1}}{n}, x_0\right\rangle + d(x_0) \\ & = & -\frac{1}{2}\|y_0\|^2 + \left\langle A^{\mathrm{T}}y_0, x_{\mu_0}(y_0)\right\rangle + d(x_{\mu_0}(y_0)) \\ & = & \varphi_{\mu_0}(y_0). \end{array}$$

The first inequality above follows from the the fact that $\|A\|_{2,1} = 1$. The second inequality follows from (3.7) and the fact that $d(\frac{\mathbf{1}}{n}) = 0$.

Now we will show that if (3.6) holds for $k$ then it also holds for $k+1$. To ease notation, drop the index $k$ and write $y_+$, $x_+$, $\mu_+$ for $y_{k+1}$, $x_{k+1}$, $\mu_{k+1}$ respectively. Let $\hat{x} = (1-\theta)x + \theta x_\mu(y)$. Therefore,

$$\begin{array}{rcl} \varphi_{\mu_+}(y_+) & = & -\frac{\|y_+\|^2}{2} + \left\langle A^{\mathrm{T}}y_+, x_{\mu_+}(y_+)\right\rangle + \mu_+ d(x_{\mu_+}(y_+)) \\ & = & -\frac{\|(1-\theta)y+\theta A\hat{x}\|^2}{2} + (1-\theta)\big[\left\langle A^{\mathrm{T}}y, x_{\mu_+}(y_+)\right\rangle + \mu d(x_{\mu_+}(y_+))\big] \\ & & +\,\theta\left\langle A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+)\right\rangle \\ & \geq & (1-\theta)\left[-\frac{\|y\|^2}{2} + \left\langle A^{\mathrm{T}}y, x_{\mu_+}(y_+)\right\rangle + \mu d(x_{\mu_+}(y_+))\right]_1 \\ & & +\,\theta\left[-\frac{\|A\hat{x}\|^2}{2} + \left\langle A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+)\right\rangle\right]_2. \end{array} \tag{3.8}$$

The last inequality follows from the concavity of the function $y \mapsto -\frac{\|y\|^2}{2}$. We can now estimate the expression in the first bracket in (3.8) as follows:

$$\begin{array}{rcl} [\cdot]_1 & = & -\frac{\|y\|^2}{2} + \left\langle A^{\mathrm{T}}y, x_\mu(y)\right\rangle + \mu d(x_\mu(y)) + \left\langle A^{\mathrm{T}}y, x_{\mu_+}(y_+) - x_\mu(y)\right\rangle \\ & & +\mu(d(x_{\mu_+}y_+) - d(x_\mu(y))) \\ & = & \varphi_\mu(y) + \mu\big(d(x_{\mu_+}(y_+)) - d(x_\mu(y)) - \left\langle \nabla d(x_\mu(y)), x_{\mu_+}(y_+) - x_\mu(y)\right\rangle\big) \\ & = & \varphi_\mu(y) + \mu h(x_{\mu_+}(y_+), x_\mu(y)) \\ & \geq & f(x) + \mu h(x_{\mu_+}(y_+), x_\mu(y)) \\ & \geq & f(\hat{x}) + \langle \nabla f(\hat{x}), x - \hat{x}\rangle + \mu h(x_{\mu_+}(y_+), x_\mu(y)). \end{array} \tag{3.9}$$

The last three steps follow respectively from the definition of Bregman distance $h$, the induction hypothesis (3.6), and the convexity of $f$.

For the expression in the second bracket in (3.8) we have

$$\begin{aligned}[\cdot]_2 &= \frac{\|A\hat{x}\|^2}{2} - \left\langle A^{\mathrm{T}}A\hat{x}, \hat{x}\right\rangle + \left\langle A^{\mathrm{T}}\hat{x}, x_{\mu_+}(y_+)\right\rangle \\ &= f(\hat{x}) + \left\langle A^{\mathrm{T}}\hat{x}, x_{\mu_+}(y_+) - \hat{x}\right\rangle.\end{aligned} \tag{3.10}$$

Moreover,

$$\begin{aligned}x_+ - \hat{x} &= (1-\theta)x + \theta x_{\mu_+}(y_+) - (1-\theta)x - \theta x_\mu(y) \\ &= \theta(x_{\mu_+}(y_+) - x_\mu(y)).\end{aligned} \tag{3.11}$$

Plugging (3.9) and (3.10) into (3.8), we finish the proof as follows:

$$\begin{aligned}\varphi_{\mu_+}(y_+) &= (1-\theta)\big[f(\hat{x}) + \langle \nabla f(\hat{x}), x - \hat{x}\rangle + \mu h(x_{\mu_+}(y_+), x_\mu(y))\big] \\ &\quad + \theta\big[f(\hat{x}) + \left\langle \nabla f(\hat{x}), x_{\mu_+}(y_+) - \hat{x}\right\rangle\big] \\ &= f(\hat{x}) + \theta\left\langle \nabla f(\hat{x}), x_{\mu_+}(y_+) - x_\mu(y)\right\rangle + (1-\theta)\mu h(x_{\mu_+}(y_+), x_\mu(y)) \\ &\geq f(\hat{x}) + \theta\left\langle \nabla f(\hat{x}), x_{\mu_+}(y_+) - x_\mu(y)\right\rangle + \frac{1}{2}\theta^2\|x_{\mu_+}(y_+) - x_\mu(y)\|_1^2 \\ &= f(\hat{x}) + \left\langle A^{\mathrm{T}}\hat{x}, x_+ - \hat{x}\right\rangle + \frac{1}{2}\|x_+ - \hat{x}\|_1^2 \\ &\geq \frac{\|A\hat{x}\|^2}{2} + \left\langle A^{\mathrm{T}}\hat{x}, x_+ - \hat{x}\right\rangle + \frac{1}{2}\|A(x_+ - \hat{x})\|^2 \\ &= f(x_+).\end{aligned}$$

The second step above follows because $\hat{x} = (1-\theta)x + \theta x_\mu(y)$. The third step follows from (3.7) and the fact that $\dfrac{\theta^2}{2(1-\theta)} \leq \mu$. The fourth step follows from (3.11). The fifth step follows because $\|A\|_{1,2} = 1$. ■

**Proof of Theorem 3:** By putting Lemma 1 and Lemma 2 together, we have

$$\tfrac{1}{2}\rho(A)^2 \leq f(x_k) \leq \varphi_{\mu_k}(y_k) \leq \varphi(y_k) + \mu_k \log(n).$$

In our algorithm $\mu_0 = 1$ and $\mu_{k+1} = \frac{k+1}{k+3}\mu_k$. So

$$\mu_k = \frac{2}{(k+1)(k+2)} < \frac{2}{(k+1)^2}.$$

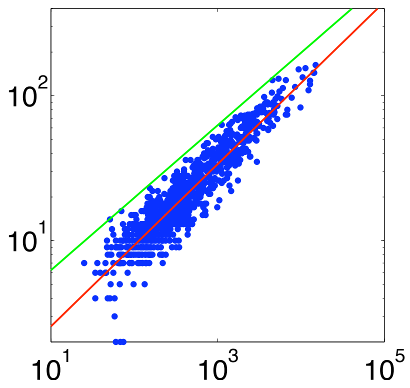
Therefore $\varphi(y_k) > 0$, and consequently $A^{\mathrm{T}}y_k > 0$, as soon as

$$k \geq \frac{2\sqrt{\log(n)}}{\rho(A)} - 1.$$

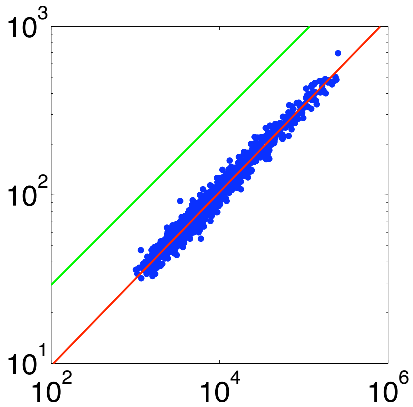
■

## 3.5 Numerical experiments

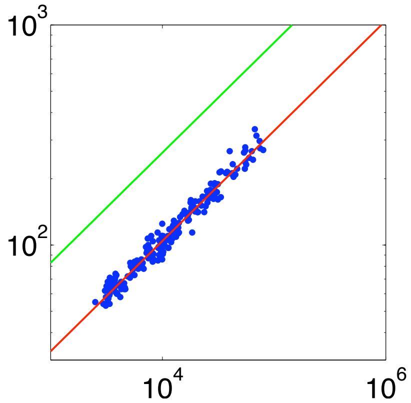
We next report results of some numerical experiments comparing the performance of the smooth perceptron algorithm and the classical perceptron algorithm. The theoretical convergence rates of the smooth and the classical perceptron algorithms suggest the relationship $Y = 2\sqrt{\log(n)}X^{\frac{1}{2}}$ between the number of iterations $Y$ taken by the smooth perceptron algorithm, and the number of iterations $X$ taken by the classical perceptron algorithm to find a feasible solution to (2.3). We test this relationship by estimating the parameter $\beta$ in the model $Y = cX^{\beta}$ for a collection of empirically generated values of the pair $(X, Y)$. To estimate the exponent $\beta$ via linear regression, we apply a logarithmic transformation and estimate the slope $\beta$ in the linearized model $\log(Y) = \log(c) + \beta \log(X)$.



(a) $A \in \mathbb{R}^{10\times 50}$



(b) $A \in \mathbb{R}^{100\times 500}$



(c) $A \in \mathbb{R}^{200\times 1000}$

Figure 3.1: Scatter plot of the number of iterations taken by smooth and classical versions of the perceptron algorithm.

We implemented the classical and the smooth perceptron algorithms in MATLAB and ran them in collections of randomly generated matrices of sizes $10 \times 50$, $100 \times 500$, and $200 \times 1000$ with a wide spectrum of thickness parameter $\rho(A)$. To generate the matrix $A$ such that $A^{\mathrm{T}}y > 0$ is feasible, we generate a random vector $\bar{y} \in \mathbb{R}^m$ and a random matrix $B \in \mathbb{R}^{m\times n}$. Then we set $A = B + \bar{y} \cdot (\epsilon e - \bar{y}^{\mathrm{T}} B)$, where $\epsilon > 0$ is an arbitrary number. In this case, we assure that $\bar{y}$ is a feasible solution for $A^{\mathrm{T}}y > 0$ and $\epsilon$ is the width of the feasibility cone $\{y \mid A^{\mathrm{T}}y \geq 0\}$. We recorded the number of iterations and CPU time required by each of these algorithms to find a solution to (2.3). Figure 3.1 displays log-log scatter plots of the number of iterations taken by the smooth perceptron algorithm versus those taken by the classical perceptron algorithm. Each plot also shows the fitted regression line in red. In addition, we plot the theoretical linear relationship $\log(Y) = \log(2\sqrt{\log(n)}) + \frac{1}{2}\log(X)$ in green. Table 3.1 displays the regression estimates of $\beta$. For example, the linear regression yields the point estimate $\hat{\beta} = 0.5009$ with 95% confidence interval $= [0.4880, 0.5137]$ for the instances with size $200 \times 1000$. As Table 3.1 shows, the slope of the regression line is close to 0.5 in all cases. Consequently, the green and red lines are approximately parallel.

The experimental results confirm that the number of iterations required by the smooth perceptron algorithm is indeed lower than those required by the classical perceptron al-

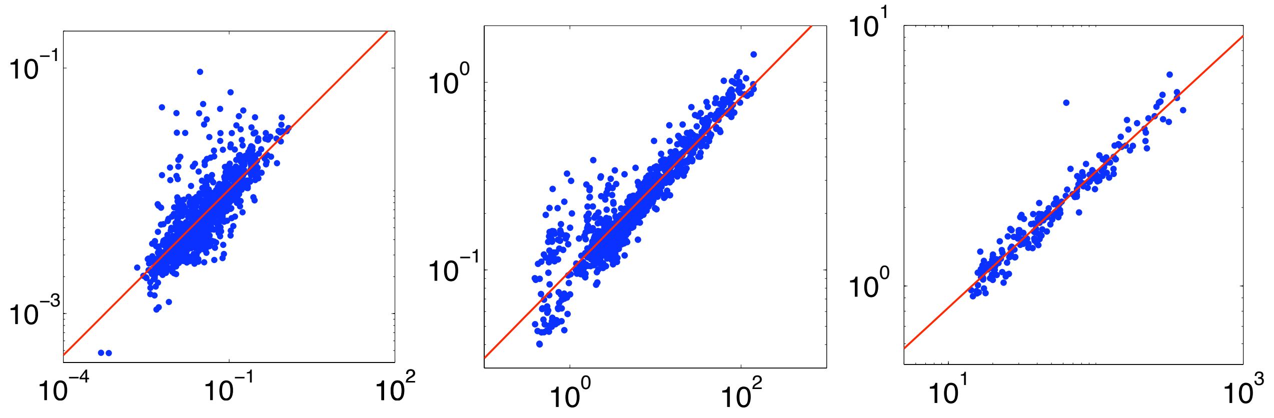
Table 3.1: Regression results for number of iterations

| size $10 \times 50$ | | size $100 \times 500$ | | size $200 \times 1000$ | |
|---|---|---|---|---|---|
| $\hat{\beta}$ | 95% CI | $\hat{\beta}$ | 95% CI | $\hat{\beta}$ | 95% CI |
| 0.5597 | $[0.5462, 0.5732]$ | 0.5156 | $[0.5104, 0.5207]$ | 0.5009 | $[0.4880, 0.5137]$ |

Table 3.2: Regression results for CPU times

| size $10 \times 50$ | | size $100 \times 500$ | | size $200 \times 1000$ | |
|---|---|---|---|---|---|
| $\hat{\beta}$ | 95% CI | $\hat{\beta}$ | 95% CI | $\hat{\beta}$ | 95% CI |
| 0.4498 | $[0.4272, 0.4724]$ | 0.4631 | $[0.4511, 0.4752]$ | 0.5216 | $[0.5029, 0.5402]$ |

gorithm. However, each iteration of the smooth perceptron algorithm involves more computational work. This naturally raises a question about the type of relationship between the computational times taken by the two algorithms. Figure 3.2 and Table 3.2 respectively display log-log scattered plots with regression lines and summaries of the regression estimates for the CPU times taken by the smooth perceptron algorithm and by the classical perceptron algorithm. The scatter plots show that the total CPU time taken by the smooth perceptron algorithm is also substantially lower than that taken by the classical perceptron algorithm. Furthermore, Table 3.2 shows that the slopes of the regression lines in these figures are also close to 0.5.



(a) $A \in \mathbb{R}^{10\times 50}$ (b) $A \in \mathbb{R}^{100\times 500}$ (c) $A \in \mathbb{R}^{200\times 1000}$

Figure 3.2: Scatter plot of CPU time taken by smooth and classical versions of the perceptron algorithm.

# Chapter 4

# A Primal-Dual Elementary Algorithm

(Joint work with Javier Peña)

## 4.1 Introduction

The main contribution of this chapeter is an *Iterated Smooth Perceptron–von Neumann Algorithm* (Algorithm ISPVN) that solves the pair of feasibility problems (2.2) and (2.3). Algorithm ISPVN relies on Nesterov's smoothing techniques [47, 49], and extends our smooth perceptron algorithm proposed in Chapter 3.

For a given $A \in \mathbb{R}^{m\times n}$ with $\rho(A) > 0$, Algorithm ISPVN either finds an $\epsilon$-solution to (2.2) in $\mathcal{O}\left(\frac{\sqrt{n}}{\rho(A)} \log\left(\frac{1}{\epsilon}\right)\right)$ elementary iterations, or finds a solution to the alternative system (2.3) in at most $\mathcal{O}\left(\frac{\sqrt{n}}{\rho(A)} \log\left(\frac{1}{\rho(A)}\right)\right)$ elementary iterations. Like the perceptron and von Neumann algorithms, the iterations in Algorithm ISPVN are *elementary* in the sense that they only involve simple computational steps. The iteration complexity of Algorithm ISPVN substantially improves the dependence on $\rho(A)$ of the iteration complexity $\mathcal{O}\left(\frac{1}{\rho(A)^2}\right)$ of the perceptron algorithm [15, 50], and the iteration complexity $\mathcal{O}\left(\frac{1}{\rho(A)^2} \log\left(\frac{1}{\epsilon}\right)\right)$ of von Neumann algorithm [31]. However, the new iteration bound incurs an extra factor of $\sqrt{n}$ whereas the former bounds depend solely on $\rho(A)$. In contrast to our smooth perceptron algorithm that only applies to (2.3), our algorithm simultaneously handles both (2.3) and its alternative system (2.2). This comes at the expense of a weaker complexity bound in the case when (2.3) is feasible.

An extended version of Algorithm ISPVN applies to more general conic systems. More precisely, assume $\mathcal{K} \subseteq \mathbb{R}^n$ is a convex regular cone and $A \in \mathbb{R}^{m\times n}$. Then Algorithm

This chapter is based on [65] published in Fields Institute Communication monograph.

ISPVN and its convergence properties extend in a natural fashion to the conic system $A^{\mathrm{T}}y \in \operatorname{int}(\mathcal{K}^*)$, and its alternative $Ax = 0,\ x \in \mathcal{K},\ x \neq 0$ provided that a suitable smooth separation oracle for the cone $\mathcal{K}$ is available. An oracle of this kind is readily available for the main cones of interest in convex optimization, namely the non-negative orthant, the semidefinite cone, the second-order cone, and direct products of these types cones.

## 4.2 Iterated smooth perceptron–von Neumann algorithm

This section presents our main contribution, namely an *iterated smooth perceptron–von Neuman algorithm* (ISPVN) for solving both (2.2) and (2.3). Algorithm ISPVN relies on Nesterov's smoothing techniques [47, 49]. Throughout this section, we assume the matrix $A$ satisfies Assumption 1.

Here we consider the following *smooth* version of the map $y \mapsto x(y)$ which is slightly different with (3.1). Given $\bar{x} \in \Delta_n$ and $\mu > 0$ let $x_\mu : \mathbb{R}^m \to \Delta_n$ be defined as

$$x_\mu(y) := \underset{x \in \Delta_n}{\operatorname{argmin}} \left\{ \langle A^{\mathrm{T}}y, x \rangle + \frac{\mu}{2}\|x - \bar{x}\|^2 \right\}.$$

The minimizer $x_\mu(y)$ can be easily found by sorting the entries of $\bar{x} - \frac{1}{\mu}A^{\mathrm{T}}y$.

The following algorithm SPVN (Smooth Perceptron–von Neumann) is a smooth version of the Perceptron–von Neumann template. Assume $\bar{x} \in \Delta_n$ and $\delta > 0$ are given inputs.

```
Algorithm SPVN($\bar{x}, \delta$)
  begin
    $y_0 := A\bar{x}$;  $\mu_0 := 2n$;  $x_0 := x_{\mu_0}(y_0)$;
    for $k = 0, 1, 2, \ldots$
      if $A^{\mathrm{T}}y_k > 0$ then Halt // $y_k$ is a solution to (2.3) fi
      if $\|Ax_k\| \leq \delta$ then Return $x_k$ fi;
      $\theta_k := \frac{2}{k+3}$;
      $y_{k+1} := (1 - \theta_k)(y_k + \theta_k Ax_k) + \theta_k^2 Ax_{\mu_k}(y_k)$;
      $\mu_{k+1} := (1 - \theta_k)\mu_k$;
      $x_{k+1} := (1 - \theta_k)x_k + \theta_k x_{\mu_{k+1}}(y_{k+1})$;
    end
end
```

Algorithm SPVN is a slight modification of our smooth perceptron algorithm. The main difference is that algorithm SPVN uses the Euclidean prox-function to smooth the map $x(y)$ instead of the entropy prox-function used in (3.1). As Proposition 2 in Section 4.4 shows, Algorithm SPVN finds a solution to (2.3) in at most $\frac{2\sqrt{2n}}{\rho(A)} - 1$ iterations provided (2.3) is feasible and $\delta < \rho(A)$. On the other hand, when (2.2) is feasible, it can be shown that Algorithm SPVN halts after at most $\mathcal{O}\left(\frac{1}{\sqrt{\delta}}\right)$ iterations with a $\delta$-solution to (2.2).

The following iterated version ISPVN of algorithm SPVN achieves a substantially better complexity when (2.2) is feasible.

Assume $\gamma > 1$ is a fixed constant and $\epsilon > 0$ is a given input.

**Algorithm** ISPVN$(\gamma, \epsilon)$

```
begin
  $\tilde{x}_0 = \frac{e}{n}$;
  for $i = 0, 1, 2, \ldots$
    $\delta_i := \frac{\|A\tilde{x}_i\|}{\gamma}$;
    $\tilde{x}_{i+1} :=$ SPVN$(\tilde{x}_i, \delta_i)$;
    if $\delta_i < \epsilon$ then Halt fi
  end
end
```

We are now ready to state our main result.

**Theorem 4** *Assume $A \in \mathbb{R}^{m\times n}$ is such that $\rho(A) > 0$.*

**(i)** *If the system (2.2) is feasible then each call to* SPVN *in Algorithm* ISPVN *halts in at most*

$$\frac{2\sqrt{2n}\gamma}{\rho(A)} - 1$$

*iterations.*

*For any given $\epsilon > 0$ Algorithm* ISPVN *finds an $\epsilon$-solution to (2.2) in at most*

$$\frac{\log(1/\epsilon)}{\log(\gamma)}$$

*outer iterations, that is, in at most $\left(\frac{2\sqrt{2n}\gamma}{\rho(A)} - 1\right) \cdot \left(\frac{\log(1/\epsilon)}{\log(\gamma)}\right) = \mathcal{O}\left(\frac{\sqrt{n}}{\rho(A)} \log\left(\frac{1}{\epsilon}\right)\right)$ elementary iterations.*

**(ii)** *If (2.3) is feasible, then each call to* SPVN *in Algorithm* ISPVN *halts in at most*

$$\frac{2\sqrt{2n}}{\rho(A)} - 1$$

*iterations.*

*Algorithm* ISPVN *finds either an $\epsilon$-solution to (2.2) or a solution to (2.3) in at most*

$$\frac{\log(1/\rho(A))}{\log(\gamma)}$$

*outer iterations, that is, in at most $\left(\frac{2\sqrt{2n}}{\rho(A)} - 1\right) \cdot \left(\frac{\log(1/\rho(A))}{\log(\gamma)}\right) = \mathcal{O}\left(\frac{\sqrt{n}}{\rho(A)} \log\left(\frac{1}{\rho(A)}\right)\right)$ elementary iterations.*

Theorem 4 is a special case of the more general Theorem 5 presented in the next section.

## 4.3 Smooth perceptron–von Neuman algorithm for conic systems

Assume $\mathcal{K} \subseteq \mathbb{R}^n$ is a fixed *regular* convex cone, that is, $\mathcal{K}$ is closed, pointed and has non-empty interior. We next generalize Algorithms SPVN and ISPVN to the homogeneous conic system

$$A^{\mathrm{T}} y \in \operatorname{int}(\mathcal{K}^*), \tag{4.1}$$

and its alternative

$$Ax = 0, \ x \in \mathcal{K}, \ x \neq 0, \tag{4.2}$$

for a given matrix $A \in \mathbb{R}^{m \times n}$. We note that in contrast to Sections 4.2, we do not assume that the columns of $A$ are normalized.

We proceed by defining general versions of $\Delta_n, x_\mu$, and $\rho(A)$. Let $\mathbf{1} \in \operatorname{int}(\mathcal{K}^*)$ be fixed. Define the set $\Delta(\mathcal{K})$ as

$$\Delta(\mathcal{K}) := \{x \in \mathbb{R}^n : \ x \in \mathcal{K}, \ \langle \mathbf{1}, x\rangle = 1\}.$$

Given $\epsilon > 0$, we say that $x \in \Delta(\mathcal{K})$ is an $\epsilon$-solution to (4.2) if $\|Ax\| \leq \epsilon$.

Observe that for $\mathcal{K} = \mathbb{R}^n_+$ and $\mathbf{1} = e = \begin{bmatrix}1 & \cdots & 1\end{bmatrix}^{\mathrm{T}}$, the set $\Delta(\mathcal{K})$ is precisely the standard simplex $\Delta_n$. For the cone $\mathcal{K} = \mathbb{S}^n_+$ of symmetric positive semidefinite matrices in the space $\mathbb{S}^n$ of $n \times n$ symmetric matrices and $\mathbf{1} = I_n$, the set $\Delta(\mathcal{K})$ is $\{X \in \mathbb{S}^n_+ : \operatorname{trace}(X) = 1\}$ which is sometimes called the *spectraplex.* For the second-order cone $\mathbb{L}_n = \left\{\begin{bmatrix} x_0 \\ \bar{x}\end{bmatrix} \in \mathbb{R}^n : \|\bar{x}\| \leq x_0\right\}$ and $\mathbf{1} = \begin{bmatrix} 1 \\ \mathbf{0}\end{bmatrix}$, the set $\Delta(\mathcal{K})$ is the lifted ball $\left\{\begin{bmatrix} 1 \\ \bar{x}\end{bmatrix} \in \mathbb{R}^n : \|\bar{x}\| \leq 1\right\}$.

Our extension of Algorithms SPVN and ISPVN to the conic systems (4.1) and (4.2) relies on the following key assumption.

**Assumption 2** There is an available oracle that computes

$$\operatorname*{argmin}_{x \in \Delta(\mathcal{K})} \left\{\frac{1}{2}\|x\|^2 - \langle g, x\rangle\right\} \tag{4.3}$$

for any given $g \in \mathbb{R}^n$.

Assumption 2 readily holds when $\mathcal{K} = \mathbb{R}^n_+$ and $\mathbf{1} = e = \begin{bmatrix}1 & \cdots & 1\end{bmatrix}^{\mathrm{T}}$. In this case $\Delta(\mathcal{K}) = \Delta_n$ and the solution to (4.3) is $x = (g - \theta e)^+$ where $\theta \in \mathbb{R}$ is such that $\|(g-\theta e)^+\|_1 = 1$. This $\theta$ can be obtained by sorting the values of $g$. Likewise, Assumption 2 holds when $\mathcal{K} = \mathbb{S}^n_+$ and $\mathbf{1} = I_n$. In this case the solution to (4.3) for $g \in \mathbb{S}^n$ is $x = U\operatorname{Diag}((\lambda(g) - \theta e)^+)U^{\mathrm{T}}$ where $g = U\operatorname{Diag}(\lambda(g))U^{\mathrm{T}}$ is the spectral decomposition of $g$ and $\theta \in \mathbb{R}$ is such that $\|(\lambda(g) - \theta e)^+\|_1 = 1$. This time the value of $\theta$ can be obtained by sorting the values of the vector of eigenvalues $\lambda(g)$. Similarly, Assumption 2 also holds when $\mathcal{K} = \mathbb{L}_n$ and $\mathbf{1} = \begin{bmatrix} 1 \\ \mathbf{0} \end{bmatrix}$. In this case the solution to (4.3) for $g = \begin{bmatrix} g_0 \\ \bar{g} \end{bmatrix} \in \mathbb{R}^n$ is $x = (\lambda_1(g) - \theta)^+ \begin{bmatrix} 1 \\ \frac{\bar{g}}{\|\bar{g}\|} \end{bmatrix} + (\lambda_2(g) - \theta)^+ \begin{bmatrix} 1 \\ -\frac{\bar{g}}{\|\bar{g}\|} \end{bmatrix}$ where $g = \frac{1}{2}\lambda_1(g) \begin{bmatrix} 1 \\ \frac{\bar{g}}{\|\bar{g}\|} \end{bmatrix} + \frac{1}{2}\lambda_2(g) \begin{bmatrix} 1 \\ -\frac{\bar{g}}{\|\bar{g}\|} \end{bmatrix}$ is the Jordan algebra spectral decomposition of $g$ (see [2]), that is, $\lambda_1(g) = g_0 + \|\bar{g}\|$, $\lambda_2(g) = g_0 - \|\bar{g}\|$, and $\theta \in \mathbb{R}$ is such that $(\lambda_1(g) - \theta)^+ + (\lambda_2(g) - \theta)^+ = 1$. This value of $\theta$ is readily computable: $\theta = \begin{cases} \lambda_1(g) - 1 & \text{if } \lambda_1(g) \geq \lambda_2(g) + 1 \\ \frac{\lambda_1(g)+\lambda_2(g)-1}{2} & \text{otherwise.} \end{cases}$

Proceeding in a similar fashion to the three cases above, it is easy to see that Assumption 2 also holds when $\mathcal{K}$ is a direct product of non-negative orthants, semidefinite cones, and second-order cones.

Given $\bar{x} \in \Delta(\mathcal{K})$ and $\mu > 0$ let $x_\mu : \mathbb{R}^m \to \Delta(\mathcal{K})$ be defined as

$$x_\mu(y) := \underset{x \in \Delta(\mathcal{K})}{\operatorname{argmin}} \left\{ \langle A^{\mathrm{T}} y, x \rangle + \frac{\mu}{2} \|x - \bar{x}\|^2 \right\}.$$

Observe that the mapping $x_\mu(\cdot)$ is computable by Assumption 2.

Assume $M$ is a known upper bound on $\|A\|$. We are now ready to give the general versions of Algorithms SPVN and ISPVN .

Assume $\bar{x} \in \Delta(\mathcal{K})$ and $\delta > 0$ are given inputs.

**Algorithm** SPVNC($\bar{x}, \delta$)

```
begin
   $y_0 := A\bar{x}$;  $\mu_0 := 2M^2$;  $x_0 := x_{\mu_0}(y_0)$;
   for $k = 0, 1, 2, \ldots$
      if $A^{\mathrm{T}} y_k > 0$ then Halt // $y_k$ is a solution to (4.1) fi
      if $\|Ax_k\| \leq \delta$ then Return $x_k$ fi;
      $\theta_k := \frac{2}{k+3}$;
      $y_{k+1} := (1 - \theta_k)(y_k + \theta_k A x_k) + \theta_k^2 A x_{\mu_k}(y_k)$;
      $\mu_{k+1} := (1 - \theta_k)\mu_k$;
      $x_{k+1} := (1 - \theta_k) x_k + \theta_k x_{\mu_{k+1}}(y_{k+1})$;
   end
end
```

Assume $\gamma > 1$ is a fixed constant and $\epsilon > 0$ is a given input.

**Algorithm** ISPVNC($\gamma, \epsilon$)

```
begin
   pick $\tilde{x}_0 \in \Delta(\mathcal{K})$;
   for $i = 0, 1, 2, \ldots$
      $\delta_i := \dfrac{\|A\tilde{x}_i\|}{\gamma}$;
```

```
        $\tilde{x}_{i+1}$ := SPVN ($\tilde{x}_i, \delta_i$);
        if $\delta_i < \epsilon$ then Halt fi
    end
end
```

Let $D$ denote the *diameter* of the set $\Delta(\mathcal{K})$, that is

$$D := \max_{u,v \in \Delta(\mathcal{K})} \|u - v\|. \tag{4.4}$$

Notice that $D$ is well-defined and finite since the set $\Delta(\mathcal{K})$ is compact.

For a given $A \in \mathbb{R}^{m\times n}$ let

$$\rho(A) := \left| \max_{\|y\|=1} \min_{x\in\Delta(\mathcal{K})} \left\langle A^{\mathrm{T}} y, x \right\rangle \right|. \tag{4.5}$$

We have the following general version of Theorem 4.

**Theorem 5** *Assume $A \in \mathbb{R}^{m\times n}$ is such that $\rho(A) > 0$.*

**(i)** *If the system (4.2) is feasible then each call to* SPVNC *in Algorithm* ISPVNC *halts in at most*

$$\frac{2MD\gamma}{\rho(A)} - 1 \tag{4.6}$$

*iterations.*

*For any given $\epsilon > 0$ Algorithm* ISPVNC *finds an $\epsilon$-solution to (4.2) in at most*

$$\frac{\log(\|A\tilde{x}_0\|/\epsilon)}{\log(\gamma)} \tag{4.7}$$

*outer iterations, that is, in at most $\left(\frac{2MD\gamma}{\rho(A)} - 1\right) \cdot \left(\frac{\log(\|A\tilde{x}_0\|/\epsilon)}{\log(\gamma)}\right) = \mathcal{O}\left(\frac{MD}{\rho(A)} \log\left(\frac{\|A\tilde{x}_0\|}{\epsilon}\right)\right)$ elementary iterations.*

**(ii)** *If (4.1) is feasible, then each call to* SPVNC *in Algorithm* ISPVNC *halts in at most*

$$\frac{2MD}{\rho(A)} - 1 \tag{4.8}$$

*iterations.*

*Algorithm* ISPVNC *finds either an $\epsilon$-solution to (4.2) or a solution to (4.1) in at most*

$$\frac{\log(\|A\tilde{x}_0\|/\rho(A))}{\log(\gamma)} \tag{4.9}$$

*outer iterations, that is, in at most $\left(\frac{2MD}{\rho(A)} - 1\right) \cdot \left(\frac{\log(\|A\tilde{x}_0\|/\rho(A))}{\log(\gamma)}\right) = \mathcal{O}\left(\frac{MD}{\rho(A)} \log\left(\frac{\|A\tilde{x}_0\|}{\rho(A)}\right)\right)$ elementary iterations.*

We conclude this section by showing that Theorem 4 follows from Theorem 5.

**Proof of Theorem 4:** Since the columns of $A$ are normalized we have $\|A\| \leq \sqrt{n}$. Hence Algorithms SPVN and ISPVN are recovered as special cases of Algorithms SPVNC and ISPVNC respectively for $\mathcal{K} = \mathbb{R}^n_+$ and $\Delta(\mathcal{K}) = \Delta_n$. Next, observe that the diameter of $\Delta_n$ is $\sqrt{2}$. Furthermore, for the initial point $\tilde{x}_0 = \frac{e}{n}$ we have $\|A\tilde{x}_0\| \leq 1$ because $\|\tilde{x}_0\|_1 = 1$ and the columns of $A$ are normalized. Therefore for $\mathcal{K} = \mathbb{R}^n_+,\ \Delta(\mathcal{K}) = \Delta_n,\ \tilde{x}_0 = \frac{e}{n}$, and $A$ with normalized columns, we have $D = \sqrt{2}, M = \sqrt{n}$, and $\|A\tilde{x}_0\| \leq 1$. Consequently, in this case Theorem 5 reduces to Theorem 4. ■

## 4.4 Proof of Theorem 5

Proposition 2 and Proposition 4 below are the crux of the proof of Theorem 5. These propositions show that each call to SPVNC in Algorithm ISPVNC halts in $\mathcal{O}\left(\frac{MD}{\rho(A)}\right)$ iterations. The proofs of these propositions use ideas introduced in Chapter 3. Let $\varphi : \mathbb{R}^m \to \mathbb{R}$ be defined as

$$\varphi(y) := -\frac{1}{2}\|y\|^2 + \min_{x\in\Delta(\mathcal{K})} \left\langle A^{\mathrm{T}}y, x\right\rangle.$$

Observe that if $y \in \mathbb{R}^m$ is such that $\varphi(y) > 0$, then $A^{\mathrm{T}}y \in \operatorname{int}(\mathcal{K}^*)$.

Given $\bar{x} \in \Delta(\mathcal{K})$ and $\mu > 0$, consider the smooth approximation $\varphi_\mu$ of $\varphi$ defined as follows:

$$\begin{aligned}\varphi_\mu(y) &= -\tfrac{1}{2}\|y\|^2 + \min_{x\in\Delta(\mathcal{K})} \left\{\left\langle A^{\mathrm{T}}y, x\right\rangle + \frac{\mu}{2}\|x-\bar{x}\|^2\right\} \\ &= -\tfrac{1}{2}\|y\|^2 + \left\langle A^{\mathrm{T}}y, x_\mu(y)\right\rangle + \tfrac{\mu}{2}\|x_\mu(y)-\bar{x}\|^2.\end{aligned} \tag{4.10}$$

We will rely on the following properties of the functions $\varphi, \varphi_\mu$. Recall that $D$ stands for the diameter of $\Delta(\mathcal{K})$ as defined in (4.4).

**Lemma 3** *Assume $A \in \mathbb{R}^{m\times n}$ is given.*

**(i)** *For all $\mu > 0$*

$$0 \leq \varphi_\mu(y) - \varphi(y) \leq \frac{1}{2}\mu D^2.$$

**(ii)** *The iterates $x_k \in \Delta(\mathcal{K})$, $y_k \in \mathbb{R}^m$, $\mu_k \in \mathbb{R}$, $k = 0, 1, \ldots$ generated by Algorithm* SPVNC *satisfy*

$$\frac{1}{2}\|Ax_k\|^2 \leq \varphi_{\mu_k}(y_k). \tag{4.11}$$

**(iii)** *If $\rho(A) > 0$ and the system* (4.1) *is feasible then for all $y \in \mathbb{R}^m$ and $x \in \Delta(\mathcal{K})$,*

$$\varphi(y) \leq \frac{1}{2}\rho(A)^2 \leq \frac{1}{2}\|Ax\|^2.$$

Lemma 3 is a straightforward extension of the ideas and proof techniques in Chapter 3. However, due to minor differences, we present a proof for this lemma at the end of this chapter.

**Proposition 2** *Assume $A \in \mathbb{R}^{m\times n}$ is given and $\rho(A) > 0$. If the system* (4.1) *is feasible, Algorithm* SPVNC *halts in at most*

$$\frac{2MD}{\rho(A)} - 1$$

*iterations.*

**Proof:** It suffices to bound the number of iterations when $\delta < \rho(A)$ since otherwise the algorithm can only halt sooner. If indeed $\delta < \rho(A)$, Lemma 5(iii) implies that Algorithm SPVNC can only halt when a solution to (4.1) is found. In the algorithm $\mu_0 = 2M^2$ and $\mu_{k+1} = \frac{k+1}{k+3}\mu_k$, so

$$\mu_k = \frac{4M^2}{(k+1)(k+2)} < \frac{4M^2}{(k+1)^2}.$$

By Lemma 5(iii,ii,i) it follows that

$$\frac{1}{2}\rho(A)^2 \le \frac{1}{2}\|Ax_k\|^2 \le \varphi_{\mu_k}(y_k) \le \varphi(y_k) + \frac{1}{2}\mu_k D^2 < \varphi(y_k) + \frac{2M^2D^2}{(k+1)^2}.$$

Thus $\varphi(y_k) > 0$, and consequently $y_k$ is a solution to (4.1), if $k \ge \frac{2MD}{\rho(A)} - 1$. ■

We will rely on the characterization (4.12) below of $\rho(A)$ when (4.2) is feasible. Figure 2.1(b) illustrates this characterization in the special case when $\mathcal{K}$ is the non-negative orthant. We note that this property is closely related to a characterization of Renegar's *distance to ill-posedness,* see [58, Theorem 3.5]. Related properties of $\rho(A)$ are also discussed in [21] and [19, Chapter 6].

**Proposition 3** *Assume $\rho(A) > 0$ and the problem* (4.2) *is feasible. Then*

$$\rho(A) = \sup\{\delta : y \in \mathbb{R}^m,\ \|y\| \le \delta \Rightarrow y \in A(\Delta(\mathcal{K}))\}. \tag{4.12}$$

**Proof:** We first show the inequality "$\ge$" in (4.12). To that end, suppose $\delta > 0$ is such that $\tilde{y} \in A(\Delta(\mathcal{K}))$ for any $\tilde{y} \in \mathbb{R}^m$ with $\|\tilde{y}\| \le \delta$. Given an arbitrary $y \in \mathbb{R}^m$ with $\|y\| = 1$, put $\tilde{y} := -\delta y$. By our assumption on $\delta$, there exists $\tilde{x} \in \Delta(\mathcal{K})$ such that $A\tilde{x} = \tilde{y}$. In addition, $\langle A\tilde{x}, y\rangle = \langle \tilde{y}, y\rangle = -\delta$. So $\min\limits_{x\in\Delta(\mathcal{K})} \langle Ax, y\rangle \le -\delta$. Since this holds for any arbitrary $y \in \mathbb{R}^m$ with $\|y\| = 1$, we have $\max\limits_{\|y\|=1} \min\limits_{x\in\Delta(\mathcal{K})} \langle Ax, y\rangle \le -\delta$. Therefore, $\rho(A) \ge \delta$.

Next we show the inequality "$\le$" in (4.12). To do so, it suffices to show that if $y \notin A(\Delta(\mathcal{K}))$ then $\rho(A) < \|y\|$. Observe that $A(\Delta(\mathcal{K}))$ is closed and convex because $\Delta(\mathcal{K})$ is compact and convex. Therefore, if $y \notin A(\Delta(\mathcal{K}))$ then there exists a hyperplane that separates $y$ and $A(\Delta(\mathcal{K}))$. More precisely, there exists $z \in \mathbb{R}^m$ with $\|z\| = 1$ such that

$$\langle z, y\rangle < \min_{x\in\Delta(\mathcal{K})} \langle z, Ax\rangle \le \max_{\|y\|=1} \min_{x\in\Delta(\mathcal{K})} \langle y, Ax\rangle = -\rho(A).$$

Hence by Schwarz inequality,

$$\rho(A) < |\langle z, y\rangle| \leq \|z\|\|y\| = \|y\|.$$

■

Throughout the rest of this section let $S := \{x \in \Delta(\mathcal{K}) : Ax = 0\}$, and for $v \in \mathbb{R}^n$ let $\mathsf{dist}(v, S) := \min\{\|v - x\| : x \in S\}$.

**Lemma 4** *Assume* (4.2) *is feasible and* $\rho(A) > 0$. *Then for all* $v \in \Delta(\mathcal{K})$

$$\mathsf{dist}(v, S) \leq \frac{\|Av\|D}{\rho(A)}. \tag{4.13}$$

**Proof:** Given an arbitrary $v \in \Delta(\mathcal{K})$, the inequality (4.13) is clearly true if $v \in S$. Assume $v \in \Delta(\mathcal{K}) \setminus S$. Consider $y := -\frac{Av}{\|Av\|}\rho(A)$. By Proposition 3 there exists $u \in \Delta(\mathcal{K})$ such that $Au = y = -\frac{Av}{\|Av\|}\rho(A)$. Let $x = \lambda u + (1-\lambda)v$ for $\lambda = \frac{\|Av\|}{\|Av\|+\rho(A)}$. Then $x \in S$ and

$$\|v - x\| = \lambda\|u - v\| \leq \lambda D = \frac{\|Av\|D}{\|Av\| + \rho(A)} \leq \frac{\|Av\|D}{\rho(A)}.$$

■

**Proposition 4** *Assume* (4.2) *is feasible and* $\rho(A) > 0$. *If* $\bar{x} \in \Delta(\mathcal{K})$ *and* $\delta = \frac{\|A\bar{x}\|}{\gamma}$ *for some* $\gamma > 1$, *then Algorithm* $\mathsf{SPVNC}$ *with input* $(\bar{x}, \delta)$ *terminates in at most*

$$\frac{2MD\gamma}{\rho(A)} - 1 \tag{4.14}$$

*iterations.*

**Proof:** By Lemma 5(ii), at iteration $k$ of Algorithm $\mathsf{SPVN}$ we have

$$\begin{array}{rcl} \frac{1}{2}\|Ax_k\|^2 & \leq & \varphi_{\mu_k}(y_k) \\ & \leq & -\dfrac{\|y_k\|^2}{2} + \displaystyle\min_{x\in S}\left\{\left\langle A^{\mathrm{T}}y_k, x\right\rangle + \frac{\mu_k}{2}\|x - \bar{x}\|^2\right\} \\ & \leq & \dfrac{\mu_k}{2}\displaystyle\min_{x\in S}\|x - \bar{x}\|^2 \\ & = & \dfrac{\mu_k}{2}\mathsf{dist}(\bar{x}, S)^2. \end{array} \tag{4.15}$$

Thus by Lemma 4

$$\|Ax_k\| \leq \sqrt{\mu_k}\cdot\mathsf{dist}(\bar{x}, S) \leq \sqrt{\mu_k}\cdot\frac{D\|A\bar{x}\|}{\rho(A)} \leq \frac{2MD\|A\bar{x}\|}{(k+1)\rho(A)}.$$

So when $k \geq \frac{2MD\|A\bar{x}\|}{\rho(A)\delta} - 1 = \frac{2MD\gamma}{\rho(A)} - 1$ we have $\|Ax_k\| \leq \delta$ and Algorithm $\mathsf{SPVNC}$ halts.

■

**Proof of Theorem 5:**

**(i)** The bound (4.6) readily follows from Proposition 4. For (4.7), observe that after $N$ outer iterations algorithm ISPVNC yields $\tilde{x}_N \in \Delta(\mathcal{K})$ with

$$\|A\tilde{x}_N\| \leq \delta_{N-1} \leq \frac{\|A\tilde{x}_0\|}{\gamma^N}.$$

Thus, $\|A\tilde{x}_N\| \leq \delta_{N-1} < \epsilon$, and so algorithm ISPVNC halts, in at most $N = \frac{\log(\|A\tilde{x}_0\|/\epsilon)}{\log(\gamma)}$ outer iterations.

**(ii)** Proposition 2 readily yields the bound (4.8). Furthermore, the proof of Proposition 2 shows that algorithm ISPVNC halts with a solution to (4.1) when $\delta_{N-1} < \rho(A)$. By part (i) we know that $\delta_{N-1} \leq \frac{\|A\tilde{x}_0\|}{\gamma^N}$. Thus $\delta_{N-1} < \rho(A)$, and so algorithm ISPVNC halts with a solution to (4.1), in at most $N = \frac{\log(\|A\tilde{x}_0\|/\rho(A))}{\log(\gamma)}$ outer iterations. Note that Algorithm ISPVNC may halt with an $\epsilon$-solution to (4.2) in fewer outer iterations if $\epsilon > \rho(A)$.

■

It is natural to ask whether our main results hold if a different prox-function is used to smooth the mapping $x(\cdot)$. In particular, the entropy function $\sum_{i=1}^n x_i \log x_i + \log n$ can be used in Algorithm SPVN in place of the Euclidean distance function $\frac{1}{2}\|x - \bar{x}\|^2$. In this case, a non-iterated version of Algorithm SPVN solves (2.3) in $\mathcal{O}(\sqrt{\log n}/\rho(A))$ iterations provided it is feasible, as shown in Chapter 3. We conjecture that the main factor $\mathcal{O}(\sqrt{n}/\rho(A))$ in Theorem 4 can be improved to $\mathcal{O}(\sqrt{\log n}/\rho(A))$ if the entropy function is suitably used. It is tempting to look for a proof of this conjecture by modifying Algorithm ISPVN and the proof of Theorem 4 in obvious ways. However, this attempt runs into a roadblock because it needs a bound as that in Lemma 4 but with the entropy-induced Bregman distance in place of the Euclidean distance. Such an analog of Lemma 4 does not hold.

## 4.5 Proof of Lemma 3

**(i)** From the construction of $\varphi$ and $\varphi_\mu$ it follows that

$$\begin{aligned} \varphi_\mu(y) &= -\tfrac{1}{2}\|y\|^2 + \left\langle A^{\mathrm{T}}y, x_\mu(y)\right\rangle + \tfrac{\mu}{2}\|x_\mu(y) - \bar{x}\|^2 \\ &\geq -\tfrac{1}{2}\|y\|^2 + \left\langle A^{\mathrm{T}}y, x_\mu(y)\right\rangle \\ &\geq -\tfrac{1}{2}\|y\|^2 + \min_{x\in\Delta(\mathcal{K})} \left\langle A^{\mathrm{T}}y, x\right\rangle \\ &= \varphi(y). \end{aligned}$$

In addition,

$$\begin{array}{rcl}\varphi_\mu(y) & = & -\frac{1}{2}\|y\|^2 + \min\limits_{x\in\Delta(\mathcal{K})}\left\{\left\langle A^{\mathrm{T}}y, x\right\rangle + \frac{\mu}{2}\|x-\bar{x}\|^2\right\} \\ & \le & -\frac{1}{2}\|y\|^2 + \left\langle A^{\mathrm{T}}y, x(y)\right\rangle + \frac{\mu}{2}\|x(y)-\bar{x}\|^2 \\ & \le & \varphi(y) + \frac{1}{2}\mu D^2.\end{array}$$

**(ii)** We proceed by induction. For $k = 0$ we have:

$$\begin{array}{rcl}\frac{1}{2}\|Ax_0\|^2 & = & \frac{1}{2}\|A\bar{x}\|^2 + \langle A\bar{x}, A\,(x_0-\bar{x})\rangle + \frac{1}{2}\|A(x_0-\bar{x})\|^2 \\ & \le & -\frac{1}{2}\|A\bar{x}\|^2 + \left\langle A^{\mathrm{T}}A\bar{x}, x_0\right\rangle + \frac{1}{2}\|A\|^2\|x_0-\bar{x}\|^2 \\ & \le & -\frac{1}{2}\|y_0\|^2 + \left\langle A^{\mathrm{T}}y_0, x_{\mu_0}(y_0)\right\rangle + \frac{1}{2}\mu_0\|x_{\mu_0}(y_0)-\bar{x}\|^2 \\ & = & \varphi_{\mu_0}(y_0).\end{array}$$

Now we will show that if (4.11) holds for $k$ then it also holds for $k+1$. To ease notation, drop the index $k$ and write $y_+,\ x_+,\ \mu_+$ for $y_{k+1},\ x_{k+1},\ \mu_{k+1}$ respectively. Also, let $\hat{x} = (1-\theta)x + \theta x_\mu(y)$ so that $y_+ = (1-\theta)y + \theta A\hat{x}$. We have

$$\begin{array}{rcl}\varphi_{\mu_+}(y_+) & = & -\frac{\|y_+\|^2}{2} + \left\langle A^{\mathrm{T}}y_+, x_{\mu_+}(y_+)\right\rangle + \frac{\mu_+}{2}\|x_{\mu_+}(y_+)-\bar{x}\|^2 \\ & = & -\frac{\|(1-\theta)y+\theta A\hat{x}\|^2}{2} + (1-\theta)\left[\left\langle A^{\mathrm{T}}y, x_{\mu_+}(y_+)\right\rangle + \frac{\mu}{2}\|x_{\mu_+}(y_+)-\bar{x}\|^2\right] \\ & & +\,\theta\left\langle A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+)\right\rangle \\ & \ge & (1-\theta)\left[-\frac{\|y\|^2}{2} + \left\langle A^{\mathrm{T}}y, x_{\mu_+}(y_+)\right\rangle + \frac{\mu}{2}\|x_{\mu_+}(y_+)-\bar{x}\|^2\right]_1 \\ & & +\,\theta\left[-\frac{\|A\hat{x}\|^2}{2} + \left\langle A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+)\right\rangle\right]_2.\end{array} \tag{4.16}$$

The last inequality follows from the concavity of the function $y \mapsto -\frac{\|y\|^2}{2}$. Using (4.10), we can estimate the expression in the first bracket in (4.16) as follows:

$$\begin{array}{rcl}\left[\cdot\right]_1 & = & \varphi_\mu(y) + \left\langle A^{\mathrm{T}}y, x_{\mu_+}(y_+) - x_\mu(y)\right\rangle \\ & & +\frac{\mu}{2}\left(\|x_{\mu_+}(y_+)-\bar{x}\|^2 - \|x_\mu(y)-\bar{x}\|^2\right) \\ & = & \varphi_\mu(y) + \left\langle A^{\mathrm{T}}y + \mu(x_\mu(y)-\bar{x}), x_{\mu_+}(y_+) - x_\mu(y)\right\rangle \\ & & +\frac{\mu}{2}\|x_{\mu_+}(y_+) - x_\mu(y)\|^2 \\ & \ge & \varphi_\mu(y) + \frac{\mu}{2}\|x_{\mu_+}(y_+) - x_\mu(y)\|^2 \\ & \ge & \frac{1}{2}\|Ax\|^2 + \frac{\mu}{2}\|x_{\mu_+}(y_+) - x_\mu(y)\|^2 \\ & \ge & \frac{1}{2}\|A\hat{x}\|^2 + \left\langle A^{\mathrm{T}}A\hat{x}, x-\hat{x}\right\rangle + \frac{\mu}{2}\|x_{\mu_+}(y_+) - x_\mu(y)\|^2.\end{array} \tag{4.17}$$

The third step above follows from the optimality conditions for (4.10) at $x_\mu(y)$. The fourth step follows from the induction hypothesis (4.11).

The expression in the second bracket in (4.16) can be written as

$$\left[\cdot\right]_2 = \frac{1}{2}\|A\hat{x}\|^2 + \left\langle A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+) - \hat{x}\right\rangle. \tag{4.18}$$

Observe also that

$$\begin{aligned} x_+ - \hat{x} &= (1-\theta)x + \theta x_{\mu_+}(y_+) - (1-\theta)x - \theta x_\mu(y) \\ &= \theta(x_{\mu_+}(y_+) - x_\mu(y)). \end{aligned} \tag{4.19}$$

Plugging (4.17) and (4.18) into (4.16) we get

$$\begin{aligned} \varphi_{\mu_+}(y_+) &\geq (1-\theta)\left[\tfrac{1}{2}\|A\hat{x}\|^2 + \left\langle A^{\mathrm{T}}A\hat{x}, x-\hat{x}\right\rangle + \tfrac{\mu}{2}\|x_{\mu_+}(y_+) - x_\mu(y)\|^2\right] \\ &\qquad + \theta\left[\tfrac{1}{2}\|A\hat{x}\|^2 + \left\langle A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+) - \hat{x}\right\rangle\right] \\ &= \tfrac{1}{2}\|A\hat{x}\|^2 + \theta\left\langle A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+) - x_\mu(y)\right\rangle + \tfrac{(1-\theta)\mu}{2}\|x_{\mu_+}(y_+) - x_\mu(y)\|^2 \\ &\geq \tfrac{1}{2}\|A\hat{x}\|^2 + \theta\left\langle A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+) - x_\mu(y)\right\rangle + \tfrac{1}{2}\theta^2\|A\|^2\|x_{\mu_+}(y_+) - x_\mu(y)\|^2 \\ &\geq \tfrac{1}{2}\|A\hat{x}\|^2 + \left\langle A^{\mathrm{T}}A\hat{x}, x_+ - \hat{x}\right\rangle + \tfrac{1}{2}\|A(x_+ - \hat{x})\|^2 \\ &= \tfrac{1}{2}\|Ax_+\|^2. \end{aligned}$$

The second step above follows because $\hat{x} = (1-\theta)x + \theta x_\mu(y)$. The third step follows because at iteration $k$ we have $\frac{\theta^2\|A\|^2}{1-\theta} = \frac{4\|A\|^2}{(k+1)(k+3)} \leq \frac{4M^2}{(k+1)(k+2)} = \mu$. The fourth step follows from (4.19).

**(iii)** Since the mapping $v \mapsto \min_{x\in\Delta(\mathcal{K})} \left\langle A^{\mathrm{T}}v, x\right\rangle$ is positively homogeneous and (4.1) is feasible, it follows that

$$\begin{aligned} \varphi(y) &\leq \max_{v\in\mathbb{R}^m\setminus\{0\}} \varphi(v) \\ &= \max_{v\in\mathbb{R}^m\setminus\{0\}} \left\{-\tfrac{1}{2}\|v\|^2 + \|v\| \min_{x\in\Delta(\mathcal{K})} \left\langle A^{\mathrm{T}}\tfrac{v}{\|v\|}, x\right\rangle\right\} \\ &= \max_{t>0} \left\{-\tfrac{1}{2}t^2 + t\rho(A)\right\} \\ &= \tfrac{1}{2}\rho(A)^2. \end{aligned}$$

In addition, $\rho(A) = \min_{u\in\Delta(\mathcal{K})} \max_{\|y\|=1} \left\langle A^{\mathrm{T}}y, u\right\rangle \leq \max_{\|y\|=1} \left\langle A^{\mathrm{T}}y, x\right\rangle = \|Ax\|$ for any $x \in \Delta(\mathcal{K})$.

■

# Chapter 5

# Near Classification for Unclassifiable Data Sets

(Joint work with Javier Peña)

## 5.1 Introduction

Classification, which is the task of assigning objects to one of several predefined classes, is probably the most widely used machine learning technique. Classification is a pervasive problem in machine learning that encompasses many diverse applications. Examples include text categorization (e.g. detecting spam email messages based on upon the message header and content), fraud detection, market segmentation (e.g. predict if customer will respond to promotion) and etc.

The simplest type of classification problem is binary classification. In binary classification, we seek to separate two sets of data points in $\mathbb{R}^m$ using a linear hyperplane. In particular, we are given a collection of data points $z_i \in \mathbb{R}^m$, which comes with a label $w_i \in \{-1, 1\}$ that determines which class it belongs to. The linear hyperplane $a^{\mathrm{T}}z = b$ is said to correctly classify these two sets if all data points with $w_i = +1$ fall on one side (hence $a^{\mathrm{T}}z > b$) and all the others on the other side (hence $a^{\mathrm{T}}z < b$). Hence, the affine inequalities $w_i(a^{\mathrm{T}}z_i - b) > 0$, for all $i = 1, \dots, n$ guarantee correct classification. Therefore, binary classification problems can be recast as a system of linear inequalities $A^{\mathrm{T}}y > 0$ via homogenization.

The classical perceptron algorithm [59] is a popular algorithm to solve the binary classification problems. The perceptron algorithm and our modifications of it find a classifier if the data is linearly separable (i.e. $A^{\mathrm{T}}y > 0$ is feasible). However, data in many real-world applications may not be linearly separable. In this case, it is desirable to choose a hyperplane that allows for some mistakes in the training set and minimizes misclassification.

The support vector machine (SVM) introduced by Vapnik [69] in 1963 is a state-of-the art classification model . The popularity of SVM is due to its high accuracy and the ability to deal with high-dimensional data [26, 60, 70].

When data is separable, the goal of SVM is to find an "optimal" linear hyperplane that separates data in such a way that the minimum distance of this hyperplane from each point of the classes is maximized. This distance is called the *margin* of the hyperplane. Typically there are many possible hyperplanes with their associated margins that separate two classes. The logic behind SVM is that if we choose the hyperplane with maximum margin, we are less likely to misclassify new instances.

The margin of the canonical separating hyperplane defined by $y \in \mathbb{R}^m$ is $1/\|y\|$ (*canonical separating hyperplane* is the one that separates the data from the hyperplane by a distance of at least one). Therefore, the problem of finding a separating hyperplane with maximum margin can be formulated as

$$\begin{array}{ll} \min & \frac{1}{2}\|y\|^2 \\ & A^{\mathrm{T}}y \geq e, \end{array}$$

where $e \in \mathbb{R}^n$ be the $n$-dimensional vector of all ones. If the data is linearly separable, the maximum margin matches the parameter $\rho(A)$ after homogenization and normalization of the data set.

If data is not linearly classifiable, the SVM finds a soft margin classifier by allowing the maximum margin classifier to mis-classify some points and each misclassification is penalized. Let $(e - A^{\mathrm{T}}y)^+ := \max\{0, e - A^{\mathrm{T}}y\}$. The SVM problem can be written as a unconstrained quadratic optimization problem

$$\min_{y \in \mathbb{R}^m} \frac{\lambda}{2}\|y\|^2 + \| \left(e - A^{\mathrm{T}}y\right)^+ \|_p, \tag{5.1}$$

where $\lambda$ is the regularization parameter that controls the amount of misclassification, and $\| \cdot \|_p$ denote $p$-norm in $\mathbb{R}^n$. The term $\| \left(e - A^{\mathrm{T}}y\right)^+ \|_p$ is called *hinge loss* when $p = 1$, and *quadratic loss* when $p = 2$. Notice that when data is linearly separable the loss function is zero.

Due to the centrality of the SVM, quite a few methods were devised and analyzed to solve this problem. In particular, the Newton or Quasi-Newton methods can efficiently find an $\epsilon$-solution to (5.1) in $\sqrt{n}\log(1/\epsilon)$ [71]. However, when the number of training samples is large, these methods are impractical. This is because the space and the time costs of Hessian matrix computation rapidly increase with the dimensions of the problem instances. Decomposition algorithms [20, 39, 55] were developed to reduce the memory requirement of these interior-point methods by operating on a small working set of variables in every iteration. However, they cannot handle large-scale problem because their time complexities are super linear in $n$. Recently, Joachims [40] proposed SVM-Perf, which uses a cutting planes method to find a solution with $\epsilon$-accuracy in time $\mathcal{O}(n/(\lambda\epsilon^2))$. This bound was later improved by Smola et al. [63] to $\mathcal{O}(n/(\lambda\epsilon))$. In addition, Shwartz. et al. [62] introduced a stochastic sub-gradient descent algorithm referred to as the Pegasos Algorithm that finds an $\epsilon$-accurate solution in $\mathcal{O}\left(\|A\|/\lambda\epsilon\right)$ iterations. The complexity guarantee for Pegasos avoids the dependence on the data set size $n$. In this chapter, we present and analyze a modification of our smooth perceptron algorithm that solves (5.1) in at most $\mathcal{O}(\sqrt{n}\|A\|/\sqrt{\lambda\epsilon})$ iterations.

## 5.2 Smooth perceptron algorithm for solving SVM

In this section, we show that a small modification of our smooth perceptron algorithm can solve (5.1). Let $\|\cdot\|_{p^*}$ denote the dual norm of $\|\cdot\|_p$ and $S := \left\{x \in \mathbb{R}^n_+ : \|x\|_{p^*} \leq 1\right\}$. The problem (5.1) can be written as

$$\min_{y\in\mathbb{R}^m} f(y) := \frac{\lambda}{2}\|y\|^2 + \max_{x\in S}\left\langle x, e - A^{\mathrm{T}}y\right\rangle. \tag{5.2}$$

Let $x(y)$ denote an arbitrary point in the set $\underset{x\in S}{\mathsf{argmax}}\left\langle x, e - A^{\mathrm{T}}y\right\rangle$. Therefore $f(y) = \frac{\lambda}{2}\|y\|^2 + \left\langle x(y), e - A^{\mathrm{T}}y\right\rangle$. Notice that when $\lambda = 1$ and $p = \infty$, we recover a solution to (3.3) from (5.2).

Our extension of smooth perceptron algorithm to the problem (5.2) relies on the following assumption.

**Assumption 3** There is an available oracle that computes

$$\underset{x\in S}{\mathsf{argmax}}\left\langle x, g\right\rangle - \frac{1}{2}\|x\|^2 \tag{5.3}$$

for any given $g \in \mathbb{R}^n$.

Assumption 3 holds when $p = 1$, $p = 2$. In each of these cases, the solution to (5.3) is $x = \text{median}\left\{0, g, e\right\}$ where "median" is a component-wise operato, and $x = \begin{cases} g^+ & \text{if } \|g^+\| \leq 1 \\ \frac{g^+}{\|g^+\|} & \text{otherwise} \end{cases}$, respectively.

Given $\bar{x} \in S$ and $\mu > 0$ let $x_\mu : \mathbb{R}^m \to S$ be defined as

$$x_\mu(y) := \underset{x\in S}{\mathsf{argmax}}\left\{\left\langle x, e - A^{\mathrm{T}}y\right\rangle - \frac{\mu}{2}\|x - \bar{x}\|^2\right\}. \tag{5.4}$$

Observe that the mapping $x_\mu(\cdot)$ is computable by Assumption 3. $x_\mu(y)$ is a smooth version of the map $y \mapsto x(y)$.

Given $\lambda > 0$ and $\bar{x} \in S$, our smooth perceptron algorithm can be modified as follows:

**Modified Smooth Perceptron Algorithm for Unclassifiable Data:**

```
begin
  $y_0 := \frac{A\bar{x}}{\lambda}$;  $\mu_0 := 2\|A\|^2/\lambda$;  $x_0 := x_{\mu_0}(y_0)$;
  for $k = 0, 1, 2, \ldots$
      $\theta_k := \frac{2}{k+3}$;
```

```
        $y_{k+1} := (1-\theta_k)(y_k + \frac{\theta_k}{\lambda}Ax_k) + \frac{\theta_k^2}{\lambda}Ax_{\mu_k}(y_k);$
        $\mu_{k+1} := (1-\theta_k)\mu_k;$
        $x_{k+1} := (1-\theta_k)x_k + \theta_k x_{\mu_{k+1}}(y_{k+1});$
    end
end
```

**Theorem 6** *Given $\lambda, \epsilon > 0$, the modified smooth perceptron algorithm finds an $\epsilon$-accurate solution* (5.2) *in at most*

$$\frac{\sqrt{2D}\|A\|}{\sqrt{\lambda\epsilon}} - 1$$

*iterations, where $D = n$ or $D = 1$ if we choose hinge loss or quadratic loss respectively.*

The specific steps in our new algorithm as well as the proof of Theorem 6 are almost identical to that of our smooth perceptron algorithm in Chapter 3. However, due to the small differences and to make our exposition self-contained, we next present the proof of Theorem 6.

For $\mu > 0$ define the smooth approximation $f_\mu$ of $f$ as:

$$f_\mu(y) := \frac{\lambda}{2}\|y\|^2 + \max_{x\in S}\left\{\langle e - A^{\mathrm{T}}y, x\rangle - \frac{\mu}{2}\|x-\bar{x}\|^2\right\}, \tag{5.5}$$

It is easy to see that the maximizer in the expression for $f_\mu$ is precisely the point $x_\mu(y)$ defined (5.4). Hence

$$f_\mu(y) = \frac{\lambda}{2}\|y\|^2 + \langle e - A^{\mathrm{T}}y, x_\mu(y)\rangle - \frac{\mu}{2}\|x_\mu(y)-\bar{x}\|^2.$$

Theorem 6 is a consequence of the following lemma.

**Lemma 5** *Assume $A \in \mathbb{R}^{m\times n}$ is given.*

**(i)** *For all $\mu > 0$*

$$0 \le f(y) - f_\mu(y) \le \frac{\mu D}{2},$$

*where $D := \max_{u,v\in S}\|u-v\|$.*

**(ii)** *For all $y \in \mathbb{R}^m$ and $x \in S$,*

$$x^{\mathrm{T}}e - \frac{1}{\lambda}\|Ax\|^2 \le f(y).$$

**(iii)** *The iterates $x_k \in S$, $y_k \in \mathbb{R}^m$, $\mu_k \in \mathbb{R}$, $k = 0, 1, \ldots$ generated by the modified smooth perceptron algorithm satisfy*

$$f_{\mu_k}(y_k) \le x_k^{\mathrm{T}}e - \frac{1}{2\lambda}\|Ax_k\|^2. \tag{5.6}$$

**Proof:**

**(i)** From the construction of $f$ and $f_\mu$ it follows that

$$\begin{aligned} f_\mu(y) &= \frac{\lambda}{2}\|y\|^2 + \left\langle e - A^{\mathrm{T}}y, x_\mu(y)\right\rangle - \frac{\mu}{2}\|x_\mu(y) - \bar{x}\|^2 \\ &\leq \frac{\lambda}{2}\|y\|^2 + \left\langle e - A^{\mathrm{T}}y, x_\mu(y)\right\rangle \\ &\leq \frac{\lambda}{2}\|y\|^2 + \max_{x\in S}\left\langle e - A^{\mathrm{T}}y, x\right\rangle \\ &= f(y). \end{aligned}$$

In addition,

$$\begin{aligned} f(y) &= \frac{\lambda}{2}\|y\|^2 + \max_{x\in S}\left\langle e - A^{\mathrm{T}}y, x\right\rangle \\ &\leq \frac{\lambda}{2}\|y\|^2 + \max_{x\in S}\left\{\left\langle e - A^{\mathrm{T}}y, x\right\rangle - \frac{\mu}{2}\|x - \bar{x}\|^2\right\} + \frac{\mu}{2}\|x(y) - \bar{x}\|^2 \\ &= f_\mu(y) + \frac{\mu}{2}\|x(y) - \bar{x}\|^2 \\ &\leq f_\mu(y) + \frac{1}{2}\mu D. \end{aligned}$$

Its easy to see $D = n$ when $p = 1$ and $D = 1$ when $p = 2$.

**(ii)** For any $y \in \mathbb{R}^m$ and $x \in S$, we have

$$\begin{aligned} x^{\mathrm{T}}e - \frac{1}{2\lambda}\|Ax\|^2 &= \min_{y\in\mathbb{R}^m}\frac{\lambda}{2}\|y\|^2 + \left\langle x, e - A^{\mathrm{T}}y\right\rangle \\ &\leq \frac{\lambda}{2}\|y\|^2 + \left\langle x, e - A^{\mathrm{T}}y\right\rangle \\ &\leq \frac{\lambda}{2}\|y\|^2 + \max_{x\in S}\left\langle x, e - A^{\mathrm{T}}y\right\rangle \\ &= f(y). \end{aligned}$$

**(iii)** We proceed by induction. For $k = 0$ we have:

$$\begin{aligned} x_0^{\mathrm{T}}e - \frac{1}{2\lambda}\|Ax_0\|^2 &= x_0^{\mathrm{T}}e - \frac{1}{2\lambda}\|A\bar{x}\|^2 - \frac{1}{\lambda}\left\langle A\bar{x}, A\left(x_0 - \bar{x}\right)\right\rangle - \frac{1}{2\lambda}\|A(x_0 - \bar{x})\|^2 \\ &\geq x_0^{\mathrm{T}}e + \frac{1}{2\lambda}\|A\bar{x}\|^2 - \frac{1}{\lambda}\left\langle A^{\mathrm{T}}A\bar{x}, x_0\right\rangle - \frac{1}{2\lambda}\|A\|^2\|x_0 - \bar{x}\|^2 \\ &= \frac{\lambda}{2}\|\frac{A\bar{x}}{\lambda}\|^2 + \left\langle e - \frac{A^{\mathrm{T}}A\bar{x}}{\lambda}, x_0\right\rangle - \frac{1}{2\lambda}\|A\|^2\|x_0 - \bar{x}\|^2 \\ &= \frac{\lambda}{2}\|y_0\|^2 + \left\langle e - A^{\mathrm{T}}y_0, x_{\mu_0}(y_0)\right\rangle - \frac{1}{2}\mu_0\|x_{\mu_0}(y_0) - \bar{x}\|^2 \\ &= f_{\mu_0}(y_0). \end{aligned}$$

Now we will show that if (5.6) holds for $k$ then it also holds for $k + 1$. To ease notation, drop the index $k$ and write $y_+$, $x_+$, $\mu_+$ for $y_{k+1}$, $x_{k+1}$, $\mu_{k+1}$ respectively. Also, let $\hat{x} = (1-\theta)x + \theta x_\mu(y)$ so that $y_+ = (1-\theta)y + \dfrac{\theta}{\lambda}A\hat{x}$. We have

$$\begin{array}{rcl} f_{\mu_+}(y_+) & = & \frac{\lambda}{2}\|y_+\|^2 + \left\langle e - A^{\mathrm{T}}y_+, x_{\mu_+}(y_+)\right\rangle - \frac{\mu_+}{2}\|x_{\mu_+}(y_+) - \bar{x}\|^2 \\ & = & \frac{\lambda}{2}\|(1-\theta)y + \frac{\theta}{\lambda}A\hat{x}\|^2 + (1-\theta)\left[\left\langle e - A^{\mathrm{T}}y, x_{\mu_+}(y_+)\right\rangle - \frac{\mu}{2}\|x_{\mu_+}(y_+) - \bar{x}\|^2\right] \\ & & +\,\theta\left\langle e - \frac{1}{\lambda}A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+)\right\rangle \\ & \leq & (1-\theta)\left[\frac{\lambda}{2}\|y\|^2 + \left\langle e - A^{\mathrm{T}}y, x_{\mu_+}(y_+)\right\rangle - \frac{\mu}{2}\|x_{\mu_+}(y_+) - \bar{x}\|^2\right]_1 \\ & & +\,\theta\left[\frac{1}{2}\|\frac{A\hat{x}}{\lambda}\|^2 + \left\langle e - \frac{1}{\lambda}A^{\mathrm{T}}A\hat{x}, x_{\mu_+}(y_+)\right\rangle\right]_2. \end{array} \tag{5.7}$$

The last inequality follows from the convexity of the function $y \mapsto \frac{\|y\|^2}{2}$. We can estimate the expression in the first bracket in (5.7) as follows:

$$\begin{array}{rcl} [.]_1 & = & f_\mu(y) + \left\langle e - A^{\mathrm{T}}y, x_{\mu_+}(y_+) - x_\mu(y)\right\rangle \\ & & -\frac{\mu}{2}\left(\|x_{\mu_+}(y_+) - \bar{x}\|^2 - \|x_\mu(y) - \bar{x}\|^2\right) \\ & = & f_\mu(y) + \left\langle e - A^{\mathrm{T}}y - \mu(x_\mu(y) - \bar{x}), x_{\mu_+}(y_+) - x_\mu(y)\right\rangle \\ & & -\frac{\mu}{2}\|x_{\mu_+}(y_+) - x_\mu(y)\|^2 \\ & \leq & f_\mu(y) - \frac{\mu}{2}\|x_{\mu_+}(y_+) - x_\mu(y)\|^2 \\ & \leq & x^{\mathrm{T}}e - \frac{1}{2\lambda}\|Ax\|^2 - \frac{\mu}{2}\|x_{\mu_+}(y_+) - x_\mu(y)\|^2 \\ & \leq & \hat{x}^{\mathrm{T}}e - \frac{1}{2\lambda}\|A\hat{x}\|^2 + \left\langle e - \dfrac{A^{\mathrm{T}}A\hat{x}}{\lambda}, x - \hat{x}\right\rangle - \frac{\mu}{2}\|x_{\mu_+}(y_+) - x_\mu(y)\|^2. \end{array} \tag{5.8}$$

The third step above follows from the optimality conditions for (5.5) at $x_\mu(y)$. The fourth step follows from the induction hypothesis (5.6).

The expression in the second bracket in (5.7) can be written as

$$[.]_2 = -\frac{1}{2\lambda}\|A\hat{x}\|^2 + \left\langle e - \frac{A^{\mathrm{T}}A\hat{x}}{\lambda}, x_{\mu_+}(y_+) - \hat{x}\right\rangle + \hat{x}^{\mathrm{T}}e. \tag{5.9}$$

Observe also that

$$\begin{array}{rcl} x_+ - \hat{x} & = & (1-\theta)x + \theta x_{\mu_+}(y_+) - (1-\theta)x - \theta x_\mu(y) \\ & = & \theta(x_{\mu_+}(y_+) - x_\mu(y)). \end{array} \tag{5.10}$$

Plugging (5.8) and (5.9) into (5.7) we get

$$\begin{aligned}
f_{\mu_+}(y_+) &= (1-\theta)\left[\hat{x}^{\mathrm{T}}e - \frac{1}{2\lambda}\|A\hat{x}\|^2 + \left\langle e - \tfrac{A^{\mathrm{T}}A\hat{x}}{\lambda}, x - \hat{x}\right\rangle + \frac{\mu}{2}\|x_{\mu_+}(y_+) - x_\mu(y)\|^2\right] \\
&\quad + \theta\left[-\frac{1}{2\lambda}\|A\hat{x}\|^2 + \hat{x}^{\mathrm{T}}e + \left\langle e - \tfrac{A^{\mathrm{T}}A\hat{x}}{\lambda}, x_{\mu_+}(y_+) - \hat{x}\right\rangle\right] \\
&= \hat{x}^{\mathrm{T}}e - \frac{1}{2\lambda}\|A\hat{x}\|^2 + \theta\left\langle e - \tfrac{A^{\mathrm{T}}A\hat{x}}{\lambda}, x_{\mu_+}(y_+) - x_\mu(y)\right\rangle - \tfrac{(1-\theta)\mu}{2}\|x_{\mu_+}(y_+) - x_\mu(y)\|^2 \\
&\leq \hat{x}^{\mathrm{T}}e - \frac{1}{2\lambda}\|A\hat{x}\|^2 + \theta\left\langle e - \tfrac{A^{\mathrm{T}}A\hat{x}}{\lambda}, x_{\mu_+}(y_+) - x_\mu(y)\right\rangle - \frac{1}{2\lambda}\theta^2\|A\|^2\|x_{\mu_+}(y_+) - x_\mu(y)\|^2 \\
&= \hat{x}^{\mathrm{T}}e - \frac{1}{2\lambda}\|A\hat{x}\|^2 + \left\langle e - \tfrac{A^{\mathrm{T}}A\hat{x}}{\lambda}, x_+ - \hat{x}\right\rangle - \frac{1}{2\lambda}\|A(x_+ - \hat{x})\|^2 \\
&= x_+^{\mathrm{T}}e - \frac{1}{2\lambda}\|Ax_+\|^2.
\end{aligned}$$

The second step above follows because $\hat{x} = (1-\theta)x + \theta x_\mu(y)$. The third step follows because at iteration $k$ we have $\frac{\theta^2\|A\|^2}{\lambda(1-\theta)} = \frac{4\|A\|^2}{\lambda(k+1)(k+3)} \leq \frac{4\|A\|^2}{\lambda(k+1)(k+2)} = \mu$. The fourth step follows from (5.10).

■

**Proof of Theorem 6:** Using Lemma 5(i,ii,iii), we have

$$f(y_k) - \frac{\mu_k D}{2} \leq f_{\mu_k}(y_k) \leq x_k^{\mathrm{T}}e - \frac{1}{2\lambda}\|Ax_k\|^2 \leq \min_y f(y).$$

In our algorithm $\mu_0 = \dfrac{2\|A\|^2}{\lambda}$ and $\mu_{k+1} = \dfrac{k+1}{k+3}\mu_k$. So

$$\mu_k = \frac{4\|A\|^2}{\lambda(k+1)(k+2)} < \frac{4\|A\|^2}{\lambda(k+1)^2}.$$

Therefore $f(y_k) - \min_y f(y) \leq \epsilon$ as soon as

$$k \geq \frac{\sqrt{2D}\|A\|}{\sqrt{\lambda\epsilon}} - 1.$$

■

# Chapter 6

# A Deterministic Re-scaled Perceptron Algorithm

(Joint work with Javier Peña)

## 6.1 Introduction

This chapter proposes a version of the perceptron algorithm that includes a periodic rescaling of the space $\mathbb{R}^m$ in the same spirit as in previous work by Dunagan and Vempala [30], and by Belloni et al. [12]. Our algorithm performs at most $\mathcal{O}\left(m^5 \log(1/\rho(A))\right)$ perceptron iterations to find a solution to (2.1) in homogeneous form. Although, our new algorithm's dependence on $1/\rho(A)$ is exponentially better than that of the classical perceptron, the algorithm retains the main advantages of the classical perceptron algorithm. In contrast to the rescaling procedure in [12, 30], which is randomized and relies on a deep separation oracle, our rescaling procedure is deterministic and relies only on a separation oracle.

When $F = \{y \in \mathbb{R}^m : A^{\mathrm{T}}y > 0\}$ for $A \in \mathbb{R}^{m\times n}$, a simplified version of our rescaled perceptron algorithm has iteration bound $\mathcal{O}(m^2n^2 \log(1/\rho(A)))$. A *smooth* version of this algorithm in turn has the improved iteration bound $\mathcal{O}(mn\sqrt{m \log(n)} \log(1/\rho(A)))$.

When $F = \left\{y \in \mathbb{R}^m : A^{\mathrm{T}}y > 0\right\}$, our re-scaled perceptron algorithm is polynomial in the bit-length representation of $A \in \mathbb{Z}^{m\times n}$ and is the best deterministic complexity result for elementary algorithms in the literature so far. Aside from its theoretical merits, given the close connection between the perceptron algorithm and first-order methods, our algorithm provides a solid foundation for potential speed ups in the convergence of the widely popular first-order methods for large-scale convex optimization. Some results of similar nature have been recently obtained by Gilpin et al. [35] and by O'Donoghue and Candès [51].

This chapter is based on [54] that is under the first round of revision in Mathematical Programming.

Our re-scaled perceptron algorithm consists of an outer loop with two main phases. The first one is a perceptron phase and the second one is a rescaling phase. The perceptron phase applies a restricted number of perceptron updates. If the perceptron phase does not find a feasible solution to $F$, then with essentially no additional costs, it finds a unitary column $d \in \mathbb{R}^m$ of $A$ such that

$$F \subseteq \left\{ y \in \mathbb{R}^m : 0 \leq \langle d, y \rangle \leq \frac{1}{\sqrt{6m}} \|y\| \right\}.$$

This inclusion means that the feasible cone $F$ is nearly perpendicular to $d$. The rescaling phase, stretches $\mathbb{R}^m$ along $d$ and is guaranteed to enlarge the volume of the spherical cap $\{y \in F :, \|y\| = 1\}$ by a constant factor. This key observation ensures that the combination of the perceptron phase and the rescaling phase is guaranteed to find a feasible solution in at most $\mathcal{O}(m \log(1/\rho(A)))$ outer iterations.

In Sections 6.2 and 6.3, we first propose our rescaled perceptron algorithm for solving the polyhedral feasibility problem (2.3). Section 6.4 extends this algorithm for the more general feasibility problem (2.1). In Section 6.5, we show that our rescaled algorithm is an online learning algorithm for solving classification problems. We also provide a lower bound complexity for the number of mistakes taken by online algorithms.

## 6.2 Polyhedral case

For ease of exposition, we first consider the case $F = \{y \in \mathbb{R}^m : A^{\mathrm{T}} y > 0\}$ for $A \in \mathbb{R}^{m \times n}$. We assume $A$ satisfies Assumption 1.

**Re-scaled Perceptron Algorithm**

```
1. let $B := I$;  $\tilde{A} := A$;  $N := 6mn^2$

2. (Perceptron Phase)
   $x_0 := 0 \in \mathbb{R}^n$; $y_0 := 0 \in \mathbb{R}^m$;
   for $k = 0, 1, \ldots, N-1$
       if $\tilde{A}^{\mathrm{T}} y_k > 0$ then Halt output $By_k$
       else
           let $j \in \{1, \ldots, n\}$ be such that $\tilde{a}_j^{\mathrm{T}} y_k \leq 0$
           $x_{k+1} := x_k + e_j$
           $y_{k+1} := y_k + \tilde{a}_j$
       end if
   end for

3. (Rescaling Phase)
       $j = \operatorname{argmax}_{i=1,\ldots,n} \langle e_i, x_N \rangle$
       $B := B(I - \frac{1}{2}\tilde{a}_j \tilde{a}_j^{\mathrm{T}})$;  $\tilde{A} := (I - \frac{1}{2}\tilde{a}_j \tilde{a}_j^{\mathrm{T}})\tilde{A}$
       normalize the columns of $\tilde{A}$
```

4. Go back to Step 2.

The re-scaled perceptron algorithm changes the initial constraint matrix $A$ to a new matrix $\tilde{A} = B^{\mathrm{T}}A$. Thus when $\tilde{A}^{\mathrm{T}}y > 0$, the non-zero vector $By$ returned by the algorithm solves $A^{\mathrm{T}}y > 0$.

Now we can state a special version of our main theorem.

**Theorem 7** *Assume $A \in \mathbb{R}^{m\times n}$ satisfies Assumption 1 and the problem (2.3) is feasible. Then the re-scaled perceptron algorithm terminates with a solution to $A^{\mathrm{T}}y > 0$ after at most*

$$\frac{1}{\log(1.5)} \cdot \left( (m-1)\log\left(\frac{1}{\rho(A)\sqrt{1-\rho(A)^2}}\right) + \log(m) + \frac{1}{2}\log(\pi) \right) = \mathcal{O}\left(m \log\left(\frac{1}{\rho(A)}\right)\right)$$

*rescaling steps. Since the algorithm performs $\mathcal{O}(mn^2)$ perceptron updates between rescaling steps, the algorithm terminates after at most*

$$\mathcal{O}\left(m^2n^2 \log\left(\frac{1}{\rho(A)}\right)\right)$$

*perceptron updates.*

The key ingredients in the proof of Theorem 7 are the three lemmas below. The first of these lemmas states that if the perceptron phase does not solve $\tilde{A}^{\mathrm{T}}y > 0$, then the rescaling phase identifies a column $\tilde{a}_j$ of $\tilde{A}$ that is nearly perpendicular to the feasible cone $\{y : \tilde{A}^{\mathrm{T}}y > 0\}$. The second lemma in turn implies that the rescaling phase increases the volume of this cone by a constant factor. The third lemma states that the volume of the initial feasible cone $\{y : A^{\mathrm{T}}y > 0\}$ is bounded below by a factor of $\rho(A)^{m-1}$.

**Lemma 6** *If the perceptron phase in the rescaled perceptron algorithm does not find a solution to $\tilde{A}^{\mathrm{T}}y > 0$ then the vector $\tilde{a}_j$ in the first step of the rescaling phase satisfies*

$$\{y : \tilde{A}^{\mathrm{T}}y \geq 0\} \subseteq \left\{ y : 0 \leq \tilde{a}_j^{\mathrm{T}}y \leq \frac{1}{\sqrt{6m}}\|y\| \right\}. \tag{6.1}$$

**Proof:** Observe that at each iteration of the perceptron phase we have

$$\|y_{k+1}\|^2 = \|y_k\|^2 + 2\tilde{a}_j^{\mathrm{T}}y_k + 1 \leq \|y_k\|^2 + 1.$$

Hence $\|y_k\|^2 \leq k$. Also, throughout the perceptron phase $x_k \geq 0,\ y_k = \tilde{A}x_k$ and $\|x_{k+1}\|_1 = \|x_k\|_1 + 1$. Thus if the perceptron phase does not find a solution to $\tilde{A}^{\mathrm{T}}y > 0$ then the last iterates $y_N$ and $x_N$ satisfy $x_N \geq 0, \|x_N\|_1 = N = 6mn^2$ and $\|y_N\| = \|\tilde{A}x_N\| \leq \sqrt{N} = n\sqrt{6m}$. In particular, the index $j$ in the first step of the rescaling phase satisfies $\langle e_j, x_N\rangle \geq \|x_N\|_1/n = 6mn$. Next observe that if $\tilde{A}^{\mathrm{T}}y \geq 0$ then

$$0 \leq 6mn\,\tilde{a}_j^{\mathrm{T}}y \leq \langle e_j, x_N\rangle\,\tilde{a}_j^{\mathrm{T}}y \leq x_N^{\mathrm{T}}\tilde{A}^{\mathrm{T}}y \leq \|\tilde{A}x_N\|\,\|y\| \leq n\sqrt{6m}\|y\|.$$

So (6.1) follows. ■

The following two lemmas rely on geometric arguments concerning the unit sphere $\mathbb{S}^{m-1} := \{u \in \mathbb{R}^m : \|u\| = 1\}$. Given a measurable set $C \subseteq \mathbb{S}^{m-1}$, let $V_{m-1}(C)$ denote its volume in $\mathbb{S}^{m-1}$.

We rely on the following construction proposed by Betke [14]. Given $a \in \mathbb{S}^{m-1}$ and $\alpha > 1$, let $\Psi_{a,\alpha} : \mathbb{S}^{m-1} \to \mathbb{S}^{m-1}$ denote the transformation

$$u \mapsto \frac{(I + (\alpha-1)aa^{\mathrm{T}})u}{\|(I + (\alpha-1)aa^{\mathrm{T}})u\|} = \frac{u + (\alpha-1)(a^{\mathrm{T}}u)a}{\sqrt{1 + (\alpha^2-1)(a^{\mathrm{T}}u)^2}}.$$

This transformation stretches the sphere in the direction $a$. The magnitude of the stretch is determined by $\alpha$.

**Lemma 7** *Assume $a \in \mathbb{S}^{m-1}$, $0 < \delta < 1$, and $\alpha > 1$. If $C \subseteq \{y \in \mathbb{S}^{m-1} : 0 \leq a^{\mathrm{T}}y \leq \delta\}$ is a measurable set, then*

$$V_{m-1}\left(\Psi_{a,\alpha}(C)\right) \geq \frac{\alpha}{\left(1 + \delta^2\left(\alpha^2 - 1\right)\right)^{m/2}} V_{m-1}(C). \tag{6.2}$$

*In particular, if $\delta = \frac{1}{\sqrt{6m}}$ and $\alpha = 2$ then*

$$V_{m-1}\left(\Psi_{a,\alpha}(C)\right) \geq 1.5 V_{m-1}(C). \tag{6.3}$$

**Proof:** Without loss of generality assume $a = e_m$. Also for ease of notation, we shall write $\Psi$ as shorthand for $\Psi_{a,\alpha}$. Under these assumptions, for $y = (\bar{y}, y_m) \in \mathbb{S}^{m-1}$ we have

$$\Psi(\bar{y}, y_m) = \frac{(\bar{y}, \alpha y_m)}{\sqrt{\alpha^2 + (1-\alpha^2)\|\bar{y}\|^2}}.$$

To calculate the volume of $C$ and of $\Psi(C)$, consider the differentiable map $\Phi : \mathbb{B}^{m-1} \to \mathbb{R}^m$, defined by $\Phi(v) = \left(v, \sqrt{1 - \|v\|^2}\right)$ that maps the unit ball $\mathbb{B}^{m-1} := \{v \in \mathbb{R}^{m-1} : \|v\| \leq 1\}$ to the surface of the hemisphere $\{(\bar{y}, y_m) \in \mathbb{S}^m : y_m \geq 0\}$ containing the set $C$. The volume of $C$ is

$$V_{m-1}(C) = \int_{\Phi^{-1}(C)} |||\Phi'||| dv.$$

where $|||\Phi'|||$ denotes the volume of the $(m-1)$-dimensional parallelepiped spanned by the vectors $\partial\Phi/\partial v_1, \ldots, \partial\Phi/\partial v_{m-1}$. Likewise, the volume of $\Psi(C)$ is

$$V_{m-1}(\Psi(C)) = \int_{\Phi^{-1}(C)} |||(\Psi \circ \Phi)'||| dv.$$

Hence to prove (6.2) it suffices to show that

$$\frac{|||(\Psi \circ \Phi)'(v)|||}{|||\Phi'(v)|||} \geq \frac{\alpha}{\left(1 + \delta^2\left(\alpha^2 - 1\right)\right)^{m/2}} \text{ for all } v \in \Phi^{-1}(C). \tag{6.4}$$

Some straightforward calculations show that for all $v \in \operatorname{int}(\mathbb{B}^{m-1})$

$$|||(\Psi \circ \Phi)'(v)||| = \frac{\alpha}{\sqrt{1-\|v\|^2}\,(\alpha^2 + (1-\alpha^2)\|v\|^2)^{m/2}} \quad \text{and} \quad |||\Phi'(v)||| = \frac{1}{\sqrt{1-\|v\|^2}}.$$

Hence for all $v \in \operatorname{int}(\mathbb{B}^{m-1})$

$$\frac{|||(\Psi \circ \Phi)'(v)|||}{|||\Phi'(v)|||} = \frac{\alpha}{(\alpha^2 + (1-\alpha^2)\|v\|^2)^{m/2}}.$$

To obtain (6.4), observe that if $v \in \Phi^{-1}(C)$ then $0 \le 1 - \|v\|^2 \le \delta^2$ and thus

$$\alpha^2 + (1-\alpha^2)\|v\|^2 \le 1 + \delta^2\left(\alpha^2 - 1\right).$$

If $\delta = \frac{1}{\sqrt{6m}}$ and $\alpha = 2$ then

$$\frac{\alpha}{\left(1+\delta^2\left(\alpha^2-1\right)\right)^{m/2}} = \frac{2}{\left(1+\frac{1}{2m}\right)^{2m/4}} \ge \frac{2}{\exp(0.25)} \ge 1.5.$$

Thus (6.3) follows from (6.2). ■

**Lemma 8** *Assume $F := \{y : A^{\mathrm{T}}y > 0\}$ is a convex cone. Then*

$$V_{m-1}(F \cap \mathbb{S}^{m-1}) \ge \left(\rho(A)\sqrt{1-\rho(A)^2}\right)^{m-1} \frac{1}{m\sqrt{\pi}} V_{m-1}(\mathbb{S}^{m-1}). \tag{6.5}$$

**Proof:** From the definition of the cone width $\tau_F$ it follows that $\mathbb{B}(z, \rho(A)) \subseteq F$ for some $z \in F$ with $\|z\| = 1$. Therefore $(1-\rho(A)^2)z + v \in F$ for all $v \in \mathbb{R}^m$ such that $\|v\| \le \rho(A)\sqrt{1-\rho(A)^2}$ and $\langle z, v\rangle = 0$. This implies that $F \cap \mathbb{S}^{m-1}$ contains a spherical cap of $\mathbb{S}^{m-1}$ with base radius $\rho(A)\sqrt{1-\rho(A)^2}$. Hence

$$V_{m-1}(F \cap \mathbb{S}^{m-1}) \ge \left(\rho(A)\sqrt{1-\rho(A)^2}\right)^{m-1} \operatorname{Vol}(\mathbb{B}^{m-1}),$$

where $\operatorname{Vol}(\mathbb{B}^{m-1})$ denote the volume of $\mathbb{B}^{m-1}$. The bound (6.5) now follows from the facts $\operatorname{Vol}(\mathbb{B}^{m-1}) = \frac{\pi^{\frac{m-1}{2}}}{\Gamma(\frac{m-1}{2}+1)}$, $V_{m-1}(\mathbb{S}^{m-1}) = \frac{2\pi^{\frac{m}{2}}}{\Gamma(\frac{m}{2})} = \frac{m\pi^{\frac{m}{2}}}{\Gamma(\frac{m}{2}+1)}$, and $\Gamma(\frac{m}{2}+1) \ge \Gamma(\frac{m-1}{2}+1)$. ■

**Proof of Theorem 7:** Let $\tilde{F} := \{y \in \mathbb{R}^m : \tilde{A}^{\mathrm{T}}y > 0\}$. Observe that the rescaling phase rescales $\tilde{F}$ to $(I + \tilde{a}_j\tilde{a}_j^{\mathrm{T}})\tilde{F}$. Therefore, Lemma 6 and Lemma 7 imply that after each rescaling phase the quantity $V_{m-1}(\tilde{F} \cap \mathbb{S}^{m-1})$ increases by a factor of 1.5 or more. Since the set $\tilde{F} \cap \mathbb{S}^{m-1}$ is always contained in a hemisphere, we conclude that the number of rescaling steps before the algorithm halts cannot be larger than

$$\frac{1}{\log(1.5)} \cdot \log\left(\frac{V_{m-1}(\mathbb{S}^{m-1})/2}{V_{m-1}(\tilde{F} \cap \mathbb{S}^{m-1})}\right)$$

To finish, apply Lemma 8. ■

## 6.3 Smooth version for the re-scaled perceptron algorithm

We next show that the perceptron phase in our re-scaled perceptron algorithm can be substituted by a *smooth perceptron phase* in Section 3. This leads to an algorithm with an improved convergence rate but whose work per main iteration is roughly comparable to that in the re-scaled perceptron algorithm.

Let $x_\mu$ be defined as in (3.1). Consider the following smooth version of the re-scaled perceptron algorithm.

**Smooth Re-scaled Perceptron Algorithm**

```
1. let $B := I$; $\tilde{A} := A$; $N := \lfloor 7n\sqrt{m \log(n)} \rfloor$

2. (Smooth Perceptron Phase)
   $y_0 := \frac{\tilde{A}e}{n}$;  $\mu_0 := 2$;  $x_0 := x_{\mu_0}(y_0)$;
   for $k = 0, 1, 2, \ldots, N-1$
       if $\tilde{A}^{\mathrm{T}} y_k > 0$ then Halt and output $By_k$
       else
           $\theta_k := \frac{2}{k+3}$;
           $y_{k+1} := (1-\theta_k)(y_k + \theta_k \tilde{A} x_k) + \theta_k^2 \tilde{A} x_{\mu_k}(y_k)$;
           $\mu_{k+1} := (1-\theta_k)\mu_k$;
           $x_{k+1} := (1-\theta_k)x_k + \theta_k x_{\mu_{k+1}}(y_{k+1})$;
       end if
   end for

3. (Rescaling Phase)
       $j = \operatorname{argmax}_{i=1,\ldots,n} \langle e_i, x_N \rangle$
       $B := B(I - \frac{1}{2}\tilde{a}_j \tilde{a}_j^{\mathrm{T}})$;  $\tilde{A} := (I - \frac{1}{2}\tilde{a}_j \tilde{a}_j^{\mathrm{T}})\tilde{A}$
       normalize the columns of $\tilde{A}$

4. Go back to Step 2.
```

**Theorem 8** *Assume $A \in \mathbb{R}^{m \times n}$ satisfies Assumption 1 and the problem* (2.3) *is feasible. Then the smooth re-scaled perceptron algorithm terminates with a solution to $A^{\mathrm{T}} y > 0$ after at most*

$$\frac{1}{\log(1.5)} \cdot \left( (m-1) \log \left( \frac{1}{\rho(A)\sqrt{1-\rho(A)^2}} \right) + \log(m) + \frac{1}{2} \log(\pi) \right) = \mathcal{O}\left( m \log \left( \frac{1}{\rho(A)} \right) \right)$$

*rescaling steps. Since the algorithm performs $\mathcal{O}(n\sqrt{m\log(n)})$ perceptron updates between rescaling steps, the algorithm terminates after at most*

$$\mathcal{O}\left(mn\sqrt{m\log(n)}\log\left(\frac{1}{\rho(A)}\right)\right)$$

*perceptron updates.*

**Proof:** This proof is a modification of the proof of Theorem 7. It suffices to show that if the smooth perceptron phase in the rescaled perceptron algorithm does not find a solution to $\tilde{A}^{\mathrm{T}}y > 0$ then the vector $\tilde{a}_j$ in the first step of the rescaling phase satisfies

$$\{y : \tilde{A}^{\mathrm{T}}y \geq 0\} \subseteq \left\{y : 0 \leq \tilde{a}_j^{\mathrm{T}}y \leq \frac{1}{\sqrt{6m}}\|y\|\right\}. \tag{6.6}$$

Indeed, from Lemma 5 it follows that if the perceptron phase does not find a solution to $\tilde{A}^{\mathrm{T}}y > 0$, then $\|\tilde{A}x_N\|^2 \leq \frac{8\log(n)}{(N+1)^2} \leq \frac{8}{49mn^2} \leq \frac{1}{6mn^2}$. Since $x_N \geq 0$ and $\|x_N\|_1 = 1$, the index $j$ in the rescaling phase satisfies $\langle e_j, x_N\rangle \geq \frac{1}{n}$. Therefore, if $\tilde{A}^{\mathrm{T}}y \geq 0$ then

$$0 \leq \frac{1}{n}\tilde{a}_j^{\mathrm{T}}y \leq \langle e_j, x_N\rangle\,\tilde{a}_j^{\mathrm{T}}y \leq x_N^{\mathrm{T}}\tilde{A}^{\mathrm{T}}y \leq \|\tilde{A}x_N\|\,\|y\| \leq \frac{1}{\sqrt{6m}n}\|y\|.$$

So (6.6) follows. ■

## 6.4 General case

We next extend our re-scaled perceptron algorithm for finding a feasible solution in a more general convex cone $F$. The gist of the algorithm for the general case is the same as that of the polyhedral case presented above. We just need a bit of extra work to identify a suitable direction for the rescaling phase. To do so, we maintain a collection of $2m$ index sets $S_j,\ j = \pm1, \pm2, \ldots, \pm m$. This collection of sets helps us determine a subset of update steps that align with each other. The sum of these steps in turn defines the appropriate direction for rescaling.

**Assumption 4** *There is an available separating oracle for the cone $F$: Given $y \in \mathbb{R}^m$ the oracle either determines that $y \in F$ or else it finds a non-zero vector $u \in F^* := \{u : \langle u, v\rangle > 0$ for all $v \in F\}$ such that $\langle u, y\rangle \leq 0$.*

Observe that for a non-singular matrix $B \in \mathbb{R}^{m\times m}$, we have $(B^{-1}F)^* = B^{\mathrm{T}}F^*$. Thus a separation oracle for $\tilde{F} := B^{-1}F$ is readily available provided one for $F$ is: Given $y \in \mathbb{R}^m$, apply the separation oracle for $F$ to the point $By$. If $By \in F$ then $y \in B^{-1}F = \tilde{F}$. If $By \notin F$, then let $u \in F^*$ be a non-zero vector such that $\langle u, By\rangle \leq 0$. Thus $\left\langle B^{\mathrm{T}}u, y\right\rangle = \langle u, By\rangle \leq 0$ with $B^{\mathrm{T}}u \in (B^{-1}F)^* = \tilde{F}^*$. Consequently, throughout the algorithm below we assume that a separation oracle for the re-scaled cone $\tilde{F}$ is available.

**General Re-scaled Perceptron Algorithm**

```
1. let $B := I$; $\tilde{F} := F$; $N := 24m^4$

2. for $j = \pm 1, \pm 2, \ldots, \pm m$
       $S_j := \emptyset$
   end for

3. (Perceptron Phase)
   $u_0 := 0 \in \mathbb{R}^m$; $y_0 := 0 \in \mathbb{R}^m$;
   for $k = 0, 1, \ldots, N-1$
       if $y_k \in \tilde{F}$ then Halt and output $By_k$
       else
           let $u_k \in \tilde{F}^*$ be such that $\langle u_k, y_k \rangle \leq 0$ and $\|u_k\| = 1$
           $y_{k+1} = y_k + u_k$
           $j := \operatorname*{argmax}_{i=1,\ldots,m} |\langle e_i, u_k \rangle|$
           if $\langle e_j, u_k \rangle > 0$ then $S_j := S_j \cup \{k\}$
           else $S_{-j} := S_{-j} \cup \{k\}$
           end if
       end if
   end for

4. (Rescaling Phase)
       $i = \operatorname*{argmax}_{j=\pm 1,\ldots,\pm m} |S_j|$
       $d := \frac{\sum_{k \in S_i} u_k}{\|\sum_{k \in S_i} u_k\|}$
       $B := B(I - \frac{1}{2}dd^{\mathrm{T}})$;  $\tilde{F} := (I + dd^{\mathrm{T}})\tilde{F}$

5. Go back to Step 2.
```

The general re-scaled perceptron algorithm changes the initial cone $F$ to $\tilde{F} = B^{-1}F$. Thus when $y \in \tilde{F}$, we have $By \in F$. Notice that although the above algorithm implicitly performs this transformation, its steps do not involve inverting any matrices or solving any system of equations.

We analyze the performance of the general re-scaled perceptron in terms of the *width* of the cone $F$:

$$\tau_F := \sup_{\|y\|=1} \{r \in \mathbb{R}_+ : \mathbb{B}(y, r) \subseteq F\}. \qquad (6.7)$$

Notice that when $F = \{y \in \mathbb{R}^m : A^{\mathrm{T}}y > 0\}$ the parameter $\rho(A)$ defined in (2.4) is the same as $\tau_F$. For a general convex cone of the form $\{y \in \mathbb{R}^m : A^{\mathrm{T}}y \in \mathcal{K}^*\}$, we have $\rho(A) \leq \tau_F$, where $\rho(A)$ is defined in (4.5).

Now we can state the general version of our main theorem.

**Theorem 9** *Assume $F \subseteq \mathbb{R}^m$ is such that Assumption 4 holds. Then the general re-scaled perceptron algorithm terminates with a solution to $y \in F$ after at most*

$$\frac{1}{\log(1.5)} \cdot \left( (m-1)\log\left(\frac{1}{\tau_F\sqrt{1-\tau_F^2}}\right) + \log(m) + \frac{1}{2}\log(\pi) \right) = \mathcal{O}\left( m \log\left(\frac{1}{\tau_F}\right)\right)$$

*rescaling steps. Since the algorithm performs $\mathcal{O}(m^4)$ perceptron updates between rescaling steps, the algorithm terminates after at most*

$$\mathcal{O}\left( m^5 \log\left(\frac{1}{\tau_F}\right)\right)$$

*perceptron updates.*

The proof of Theorem 9 is almost identical to the proof of Theorem 7. All we need is the following analog of Lemma 6. Lemma 7 and Lemma 8 in Section 6.2 also hold here after replacing $\rho(A)$ with $\tau_F$.

**Lemma 9** *If the perceptron phase in the general rescaled perceptron algorithm does not find a solution to $y \in \tilde{F}$ then the vector $d$ in the rescaling phase satisfies*

$$\tilde{F} \subseteq \left\{ y : 0 \le \langle d, y\rangle \le \frac{1}{\sqrt{6m}}\|y\| \right\}. \tag{6.8}$$

**Proof:** Proceeding as in the proof of Lemma 6, it is easy to see that if the perceptron phase does not find a solution to $y \in \tilde{F}$ then the last iterate $y_N = \sum_{k=0}^{N-1} u_k$ satisfies $\|y_N\|^2 \le N = 24m^4$. Since $\{e_1, \ldots, e_m\}$ is an orthonormal basis and each $u_k$ satisfies $\|u_k\| = 1$, we have $|\langle e_j, u_k\rangle| \ge 1/\sqrt{m}$ for $j = \underset{i=1,\ldots,m}{\mathsf{argmax}} |\langle e_i, u_k\rangle|$. Furthermore, since $|\bigcup_{j=\pm1,\ldots,\pm m} S_j| = N = 24m^4$ it follows that the set $S_i$ in the rescaling phase must have at least $12m^3$ elements. Thus

$$\left\| \sum_{k\in S_i} u_k \right\| \ge \left| \left\langle e_{|i|}, \sum_{k\in S_i} u_k \right\rangle \right| = \sum_{k\in S_i} |\langle e_{|i|}, u_k\rangle| \ge \frac{|S_i|}{\sqrt{m}} \ge 12m^{5/2}. \tag{6.9}$$

On the other hand, for all $y \in \tilde{F}$ we have

$$0 \le \sum_{k\in S_i} \langle u_k, y\rangle \le \sum_{k=0}^{N-1} \langle u_k, y\rangle = \langle y_N, y\rangle \le \|y_N\|\|y\| \le \sqrt{24}m^2\|y\|. \tag{6.10}$$

Putting (6.9) and (6.10) together, it follows that for all $y \in \tilde{F}$

$$0 \le \langle d, y\rangle \le \frac{\sqrt{24}m^2\|y\|}{12m^{5/2}} = \frac{1}{\sqrt{6m}}\|y\|.$$

Hence (6.8) holds. ■

## 6.5 Rescaled perceptron as an online learning algorithm

An online learning algorithm [56, 61] observes a stream of data points from the instance domain $\mathcal{Z}$ and makes a prediction $p_k$ for each element $z_k \in \mathcal{Z}$ on the stream at time $k$. The algorithm receives immediate feedback $w_k$ from a target domain $\mathcal{W}$ about the correctness of each prediction and uses this feedback to improve the current and the future predictions. The algorithm suffers a loss $\ell(p_k, w_k)$ every time it predicts an instance incorrectly, i.e. $p_k \neq w_k$. The ultimate goal of an online learning algorithm is to minimize the cumulative loss suffered along its run.

```
Online Learning
For $k = 1, 2, \ldots$
Receive the instance $z_k \in \mathcal{Z}$
Predict $p_k$
Receive the true label $w_k \in \mathcal{W}$
Suffer loss $\ell(p_k)$
```

Online learning algorithms do not make stochastic assumptions about the data they observe. The data source is allowed to be deterministic, stochastic, or even adversarially adaptive to the algorithm's behavior, that is, the input may be selected in the worst possible way for the learning algorithm. However, if the data were chosen randomly, no online learning algorithm could be correct more than half the time. In addition, an adversary can make the cumulative loss of the online algorithm arbitrarily large by providing an instance, waiting for the algorithm's prediction, and then stating the opposite label as the correct prediction. Thus, non-trivial statements require some restrictions. One common restriction is to assume that the correct label associated to each instance is determined by some function $h : \mathcal{Z} \to \mathcal{W}$ [16, 61]. The mapping $h$ is taken from a hypothesis class $\mathcal{H}$ which is known to the algorithm. The goal of an online learning algorithm is to make as few mistakes as possible, assuming that both the choice of $h$ and the choice of instances are under the control of an adversary. It is desirable to design online algorithms for which the maximum number of mistakes made by the algorithm is minimal relative to $\mathcal{H}$.

The online learning algorithm $\mathcal{M}$ has the hypothesis $h_k \in \mathcal{H}$ at the beginning of time $k$. The algorithm observes $z_k$ and predicts $h_k(z_k)$. At the end of this round, $w_k$ is revealed and $\mathcal{M}$ makes a mistake if $h_k(z_k) \neq w_k$. The algorithm then updates its hypothesis to $h_{k+1}$. Suppose that the labels were actually produced by some function $f$ in a given hypothesis class $\mathcal{H}$. The number of mistakes that the algorithm commits can be bounded based on the loss function as

$$\text{mistake}(\mathcal{M}, \mathcal{H}) = \max_{f \in \mathcal{H}} \sum_k \mathbf{1}_{[h_k(z_k) \neq f(z_k)]},$$

where $\mathbf{1}$ is an indicator function. We can now define what it means for an algorithm to learn a class in the mistake bound model.

**Definition 1** *An online algorithm $\mathcal{M}$ learns a class $\mathcal{H}$ with mistake bound $M$ if*

$$\textit{mistake}(\mathcal{M}, \mathcal{H}) \leq M.$$

In the rest of this section, we focus on online learning algorithms for solving binary classification problems. In the context of classification, the target domain has values -1 and 1 indicating the membership of instances to the two different classes. The instance and its target in this context are a point and its label, respectively. We represent the point $z_k$ received at time $k$ and its label $w_k$ by a column vector $a_k := w_k z_k \in \mathbb{R}^m$. Thus at time $k$, the available data and the corresponding labels define a matrix $A_k := \begin{bmatrix} a_1 & \dots & a_k \end{bmatrix} \in \mathbb{R}^{m \times k}$. We denote by $A \in \mathbb{R}^{m \times n}$ the matrix of all points and labels. Thus the end classification of interest can be written as $A^{\mathrm{T}} y > 0$ (See Chapter 5).

The classical perceptron algorithm is an example of an online learning algorithm for solving binary classification problems [16, 61] with the hypothesis class of half-spaces. Each half-space hypothesis $h_k \in \mathcal{H} = \{a \mapsto \operatorname{sign}(a^{\mathrm{T}} y) : y \in \mathbb{R}^m\}$ can be describe using a vector $y_k$. The perceptron algorithm maintains the hyperplane defined by $y_k$. Every time this algorithm receives a new instance $a_{k+1}$, it predicts $p_{k+1} = h_k(a_{k+1}) = \operatorname{sign}(a_{k+1}^{\mathrm{T}} y_k)$. The algorithm incurs a loss 1 if $p_{k+1} < 0$ and 0 otherwise. In particular, we can write the loss function as $\ell(p_k) = \mathbf{1}_{[p_k < 0]}$. The main idea of the perceptron algorithm is that as long as it does not make a mistake, i.e. $p_k > 0$, its current hyperplane $y_k$ remains unchanged. When the algorithm does make a mistake, it moves the current hyperplane towards the instance that is predicted incorrectly.

Assuming the instances are linearly separable, Block [15] and Novikoff [50] have shown that the mistake bound for the online perceptron algorithm on the instance matrix $A$ is at most $1/\rho(A)^2$. In other words, after at most $1/\rho(A)^2$ mistakes, it is assured that from this point onwards, the classifier generated by the algorithm will classify all observations correctly.

We show that when the matrix $A$ is well-conditioned, precisely when $1/\rho(A)^2 \leq m$, the perceptron algorithm's mistake bound, $1/\rho(A)^2$, is a lower bound on the number of mistakes made by any online algorithm. In particular, we show that there is an instance matrix $A$ with $1/\rho(A)^2 \leq m$ on which there is no deterministic online learning algorithm can find a solution to $A^{\mathrm{T}} y > 0$ before making at least $\lfloor 1/\rho(A)^2 \rfloor$ mistakes. There exists a similar lower bound in [42].

**Example 1** *Consider an online algorithm $\mathcal{M}$. Assume the matrix $A = \begin{bmatrix} a_1 & \dots & a_m \end{bmatrix} \in \mathbb{R}^{m \times m}$ is constructed as follows: If $y_0 \in \mathbb{R}^m$ is the first solution returned by $\mathcal{M}$, then let $a_1 = \pm e_1$ be such that $a_1^{\mathrm{T}} y_0 < 0$, where $e_1$ is a $m$-dimensional unit vector that has one in its first element and zero in the rest. Similarly, if $y_i$ is the solution returned by $\mathcal{M}$ at time $i \in \{1, \dots, m-1\}$, then let $a_{i+1} = \pm e_{i+1}$ be such that $a_{i+1}^{\mathrm{T}} y_i < 0$. Clearly, $\rho(A) = 1/\sqrt{m}$ and the algorithm $\mathcal{M}$ incurs a mistake every time a new instance is received which means that this algorithm makes $m$ mistakes.*

Our deterministic rescaled perceptron algorithm is also an elementary online learning algorithm. This algorithm includes some momentum information in its updates; that is it incorporates some influence of the past instances through a rescaling update . This is a key difference between this accelerated algorithm and the classical perceptron algorithm, since the perceptron algorithm only uses the current data point to update the current solution. As shown in Section 6.4, our rescaled perceptron algorithm finds a classifier after making no more than $\mathcal{O}\left(m^5 \log\left(1/\rho(A)\right)\right)$ mistakes.

An interesting question is whether the rescaled perceptron algorithm's complexity bound is tight. In order to answer this question, we study the lower bound complexity on the number mistakes made by online algorithms, which is of course a lower bound on the number of mistakes made by the best online learning algorithm.

### 6.5.1 Lower bound complexity on the number of mistakes

To get a lower bound on the number of mistakes made by online algorithms, we show that for any online learning algorithm there exists an instance matrix $A$ under the control of an adversary for which the algorithm makes at least $\mathcal{O}\left(m \log\left(1/\rho(A)\right)\right)$ mistakes to solve $A^{\mathrm{T}}y > 0$. The crux of our lower complexity bound is similar to the resisting oracle introduced by Nesterov in [48] and the ideas in [72].

**Theorem 10** *For any online learning algorithm $\mathcal{M}$, there exists $A$ such that the number of mistakes made by $\mathcal{M}$ to solve $A^{\mathrm{T}}y > 0$ is at least $(m-1)\left(\log\left(1/\rho(A)\right) - \log(m) - 1\right)$.*

**Proof:** Assume data is being received in an online fashion. Let $A_k = \begin{bmatrix} a_1 & \cdots & a_k \end{bmatrix} \in \mathbb{R}^{m\times k}$ is the matrix of all instances that have been observed so far and $F_k := \{y \in \mathbb{R}^m : A_k^{\mathrm{T}} y > 0\}$. Without loss of generality assume all the columns of $A_k$ are normalized. We construct a sequence of boxes $\{B_k\}_{k=0}^{\infty}$ defined by $B_k = F_k \cap \{y \in \mathbb{R}^m : y^{(m)} = 1\} = \{y = \begin{bmatrix} \bar{y} \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m : b_k \leq \bar{y} \leq d_k, y^{(m)} = 1\}$ for some $b_k, d_k \in \mathbb{R}^m$ such that $B_{k+1} \subseteq B_k$. Let $c_k := \frac{1}{2}(b_k + d_k)$ denote the center of the box $B_k$. We initialize $A_0$ to be the $2(m-1)$ vectors of the dual cone to the cone $-F_0$ such that $B_0 := \left\{y \in \mathbb{R}^m : -\bar{e} \leq \bar{y} \leq \bar{e}, y^{(m)} = 1\right\}$, where $\bar{e} \in \mathbb{R}^{m-1}$ is the $(m-1)$-dimensional vector of all ones. Let $i$ denote the active coordinate.

**Adversarially adaptive example**

```
1. let $i = 1$; $b_0 := -\bar{e}$; $d_0 := \bar{e}$; $k = 0$

2. given $A_k$, let $\hat{y}$ be the test solution generated by algorithm $\mathcal{M}$.

3. if $\hat{y} \notin F_k$ then
       $a_{k+1} := w$ where $w \in \mathbb{R}^m$ with $\|w\| = 1$ is the separator of $\hat{y}$ from $F_k$.

4. if $\hat{y} \in F_k$ then
       if $\hat{y}^{(i)} \leq c_k^{(i)}$ then
           $a_{k+1} := -\bar{e}_i + c_k^{(i)}\bar{e}_m$;
           $b_{k+1} := b_k + (c_K^{(i)} - b_k^{(i)})\bar{e}_i$;
```

```
        $d_{k+1} := d_k$;
    end if;
    if $\hat{y}^{(i)} > c_k^{(i)}$ then
        $a_{k+1} := \bar{e}_i - c_k^{(i)}\bar{e}_m$;
        $b_{k+1} := b_k$;
        $d_{k+1} := d_k + (c_k^{(i)} - d_k^{(i)})\bar{e}_i$;
    end if;
end if;
```

5. let $\bar{a}_{k+1} := a_{k+1}/\|a_{k+1}\|$ and $A_{k+1} = \begin{bmatrix} A_k & \bar{a}_{k+1} \end{bmatrix}$
6. $i = i + 1$; if $i > m - 1$, then $i = 1$; k := k+1.
7. Go to Step 2.

Notice that the box $B_{k+1} \subseteq B_k$ is always half of the last box $B_k$. In particular, the box $B_k$ is divided into two parts by the hyperplane defined by $a_{k+1}$ which passes through $c_k$ and corresponds to the active coordinate $i$. Therefore, $V_{m-1}(B_{k+1}) = \frac{1}{2}V_{m-1}(B_k)$. In addition, by construction, $d_k - b_k = \left(\frac{1}{2}\right)^{k/(m-1)}(d_0 - b_0) = 2\left(\frac{1}{2}\right)^{k/(m-1)}\bar{e}$. Hence for all $k \geq 0$, we have the inclusion

$$\mathbb{B}_{m-1}(c_k, r_k) \subseteq B_k \text{ with } r_k := \left(\frac{1}{2}\right)^{k/(m-1)}. \tag{6.11}$$

Let $z_k = \begin{bmatrix} c_k \\ 1 \end{bmatrix}$. Consider an arbitrarily point $v = \begin{bmatrix} \bar{v} \\ v^{(m)} \end{bmatrix} \in \mathbb{R}^m$ with $\|v\| \leq \dfrac{r_k}{2\|z_k\|^2}$. It is easy to see that $\|\frac{c_k + \|z_k\|\bar{v}}{1+\|z_k\|v^{(m)}} - c_k\| \leq r_k$. Therefore by (6.11), we have $\frac{c_k + \|z_k\|\bar{v}}{1+\|z_k\|v^{(m)}} \in B_k$ and hence $\begin{bmatrix} \frac{c_k + \|z_k\|\bar{v}}{1+\|z_k\|v^{(m)}} \\ 1 \end{bmatrix} \in F_k$.

This indicates that $\dfrac{z_k}{\|z_k\|} + v = \dfrac{1}{\|z_k\|}\begin{bmatrix} c_k + \|z_k\|\bar{v} \\ 1 + \|z_k\|v^{(m)} \end{bmatrix} \in F_k$. Since $v$ with $\|v\| \leq \dfrac{r_k}{2\|z_k\|^2}$ is chosen arbitrarily, we have $\mathbb{B}\left(\frac{z_k}{\|z_k\|}, \frac{r_k}{2\|z_k\|^2}\right) \subseteq F_k$ for all $k$. Therefore,

$$\rho(A) > \frac{1}{2\|z_k\|^2}\left(\frac{1}{2}\right)^{k/(m-1)} \geq \frac{1}{2m}\left(\frac{1}{2}\right)^{k/(m-1)}.$$

The last inequality in the above expression holds since $\|z_k\| \leq \sqrt{m}$ by construction.

It is clear that the number of generated boxes does not exceed the number of mistakes made by the online algorithm $\mathcal{M}$. Therefore, the number of mistakes made by $\mathcal{M}$ is at least $(m-1)(\log(1/\rho(A)) - \log(m) - 1)$. ■

# Chapter 7

# Preconditioning

(Joint work with Javier Peña and Vera Roshchina)

## 7.1 Introduction

Condition numbers play an important role in numerical analysis. The condition number of a problem is a key parameter in the computational complexity of iterative algorithms as well as in issues of numerical stability. A related challenge of paramount importance is to *precondition* a given problem instance, that is, perform some kind of data preprocessing to transform a given problem instance into an equivalent one that is better conditioned. An advantage of preconditioning is that it is intrinsic to the problem instances but not to the solution methods. Therefore, it can be applied to any class of algorithms. Preconditioning has been extensively studied in numerical linear algebra [37, 67] and is an integral part of the computational implementation of numerous algorithms for solving linear systems of equations. In the more general optimization context, the task of designing preconditioners has been studied by Epelman and Freund [32] as well as by Belloni and Freund [10]. In a similar spirit, we first show the existence of optimal preconditining procedures for homogeneous systems of linear inequalities. An optimal preconditioner makes the condition number of the preconditioned system to be bounded above by $\mathcal{O}(n^c)$ which could result in obtaining a polynomial time algorithm and would behave robustly over large problem classes. The existence of such preconditioners is theoretically very attractive. However, these preconditioners rely on some unkown information and are therefore impractical. To address this issue, we propose two simple preconditioning procedures which lead to improvements in three types of condition numbers for homogeneous systems of linear inequalities, namely Renegar's [57], Goffin-Cucker-Cheung's [21, 36], and the Grassmann condition number [5, 11]. Both Renegar's and Goffin-Cheung-Chucker's condition numbers are key parameters in the analyses of algorithmic schemes and other numerical

This chapter is based on [53] published in Optimization Letters.

properties of constraint systems [31, 33, 52, 57, 58, 64]. The more recently developed Grassmann condition number is especially well-suited for probabilistic analysis [6].

We recall the definition of the above condition numbers in Section 7.2 below. These condition numbers quantify certain properties associated to the homogenous systems of inequalities (2.2) and (2.3) defined by a matrix $A \in \mathbb{R}^{m\times n}$, where $m \leq n$. Renegar's condition number is defined in terms of the distance from $A$ to a set of *ill-posed* instances in the space $\mathbb{R}^{m\times n}$. Goffin-Cucker-Cheung's condition number is defined in terms of the *best conditioned* solution to the system defined by $A$. Alternatively, it can be seen as the reciprocal of a measure of thickness of the cone of feasible solutions to the system defined by $A$. Goffin-Cucker-Cheung's condition number is intrinsic to certain geometry in the column space of $A$. The more recent Grassmann condition number, introduced in a special case by Belloni and Freund [11] and subsequently generalized by Amelunxen-Burgisser [5] is based on the projection distance from the linear subspace spanned by the rows of $A$ to a set of *ill-posed* subspaces in $\mathbb{R}^n$. The Grassmann condition number is intrinsic to certain geometry in the row space of $A$.

The Goffin-Cucker-Cheung's condition number and the Grassmann condition numbers are respectively invariant under column scaling and elementary row operations on the matrix $A$ respectively. For suitable choices of norms, each of them is also always smaller than Renegar's condition number (see (7.2) and (7.3) below). We observe that these properties have an interesting parallel and naturally suggest two preconditioning procedures, namely *column normalization* and *row balancing.* Our main results, presented in Section 7.4, discuss some interesting properties of these two preconditioning procedures. In particular, we show that a combination of them would improve the values of the three condition numbers.

## 7.2 Condition numbers and their relations

Let $F_P$ and $F_D$ denote the sets of matrices $A$ such that (2.2) and (2.3) are respectively feasible. The sets $F_P$ and $F_D$ are closed and $F_P \cup F_D = \mathbb{R}^{m\times n}$. The set $\Sigma := F_P \cap F_D$ is the set of *ill-posed* matrices. For a given $A \in \Sigma$, arbitrary small perturbations on $A$ can lead to a change with respect to the feasibility of (2.2) and (2.3).

Renegar's condition number is defined as

$$\mathcal{C}_R(A) := \frac{\|A\|}{\min\{\|A - A'\| : A' \in \Sigma\}}.$$

Here $\|A\|$ denotes the operator norm of $A$ induced by a given choice of norms in $\mathbb{R}^n$ and $\mathbb{R}^m$. When the Euclidean norms are used in both $\mathbb{R}^n$ and $\mathbb{R}^m$, we shall write $\mathcal{C}_R^{2,2}(A)$ for $\mathcal{C}_R(A)$. On the other hand, when the one-norm is used in $\mathbb{R}^n$ and the Euclidean norm is used in $\mathbb{R}^m$ we shall write $\mathcal{C}_R^{1,2}(A)$ for $\mathcal{C}_R(A)$. These are the two cases we will consider in the sequel.

Assume $A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} \in \mathbb{R}^{m\times n}$ with $a_i \neq 0$ for $i = 1, \ldots, n$. Goffin-Cheung-

Cucker's condition number is defined as

$$\mathcal{C}_{GCC}(A) := \min_{\|y\|_2=1} \max_{i\in\{1,\ldots,n\}} \frac{\|a_i\|_2}{a_i^{\mathrm{T}} y}.$$

Here $\|\cdot\|_2$ denotes the Euclidean norm in $\mathbb{R}^m$. As mentioned in Section 2.1, the quantity $\rho(A) = 1/\mathcal{C}_{GCC}(A)$ is the Euclidean distance from the origin to the boundary of the convex hull of the set of normalized vectors $\left\{\frac{a_i}{\|a_i\|_2}, i = 1,\ldots,n\right\}$. Furthermore, when $A \in F_D$, this distance coincides with the thickness of the cone of feasible solutions to (2.3).

Let $\mathsf{Gr}_{n,m}$ denote the set of $m$-dimensional linear subspaces in $\mathbb{R}^n$, that is, the $m$-dimensional Grassmann manifold in $\mathbb{R}^n$. Let

$$P_m := \{W \in \mathsf{Gr}_{n,m} : W^{\perp} \cap \mathbb{R}^n_+ \neq \{0\}\},$$

$$D_m := \{W \in \mathsf{Gr}_{n,m} : W \cap \mathbb{R}^n_+ \neq \{0\}\},$$

and

$$\Sigma_m = P_m \cap D_m = \left\{W \in \mathsf{Gr}_{m,n} : W \cap \mathbb{R}^n_+ \neq \{0\}, W \cap \mathrm{int}(\mathbb{R}^n_+) = \emptyset\right\}.$$

The sets $P_m, D_m$, and $\Sigma_m$ are analogous to the sets $F_P, F_D$, and $\Sigma$ respectively: If $A \in \mathbb{R}^{m\times n}$ is full-rank then $A \in F_P \Leftrightarrow \mathrm{span}(A^{\mathrm{T}}) \in P_m$ and $A \in F_D \Leftrightarrow \mathrm{span}(A^{\mathrm{T}}) \in D_m$. In particular, $A \in \Sigma \Leftrightarrow \mathrm{span}(A^{\mathrm{T}}) \in \Sigma_m$. The Grassmann condition number is defined as [5]:

$$\mathcal{C}_{Gr}(A) := \frac{1}{\min\{d(\mathrm{span}(A^{\mathrm{T}}), W) : W \in \Sigma_m\}}.$$

Here $d(W_1, W_2) = \sigma_{\max}(\Pi_{W_1} - \Pi_{W_2})$, where $\Pi_{W_i}$ denotes the orthogonal projection onto $W_i$ for $i = 1, 2$. In the above expression and in the sequel, $\sigma_{\max}(M)$ and $\sigma_{\min}(M)$ denote the largest and smallest singular values of the matrix $M$ respectively.

We next recall some key properties of the condition numbers $\mathcal{C}_R^{1,2}$, $\mathcal{C}_R^{2,2}$, $\mathcal{C}_{GCC}$ and $\mathcal{C}_{Gr}$. Throughout the remaining part of the chapter assume $A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} \in \mathbb{R}^{m\times n}$ is full-rank and $a_i \neq 0$ for $i = 1,\ldots,n$. From the relationship between the one-norm and the Euclidean norm, it readily follows that

$$\frac{\mathcal{C}_R^{2,2}(A)}{\sqrt{n}} \leq \mathcal{C}_R^{1,2}(A) \leq \sqrt{n}\,\mathcal{C}_R^{2,2}(A), \tag{7.1}$$

The following property is a consequence of [21, Theorem 1]:

$$\mathcal{C}_{GCC}(A) \leq \mathcal{C}_R^{1,2}(A) \leq \frac{\max\limits_{i=1,\ldots,n} \|a_i\|_2}{\min\limits_{i=1,\ldots,n} \|a_i\|_2}\mathcal{C}_{GCC}(A). \tag{7.2}$$

The following property was established in [5, Theorem 1.4]:

$$\mathcal{C}_{Gr}(A) \leq \mathcal{C}_R^{2,2}(A) \leq \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} \mathcal{C}_{Gr}(A). \tag{7.3}$$

The inequalities (7.2) and (7.3) reveal an interesting parallel between the pairs $\mathcal{C}_{GCC}(A), \mathcal{C}_R^{1,2}(A)$ and $\mathcal{C}_{Gr}(A), \mathcal{C}_R^{2,2}(A)$. This parallel becomes especially striking by observing that the fractions in the right hand sides of (7.2) and (7.3) can be written respectively as

$$\frac{\max\limits_{i=1,\ldots,n} \|a_i\|_2}{\min\limits_{i=1,\ldots,n} \|a_i\|_2} = \frac{\max\{\|Ax\|_2 : \|x\|_1 = 1\}}{\min\{\|Ax\|_2 : \|x\|_1 = 1\}},$$

and

$$\frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} = \frac{\max\{\|A^{\mathrm{T}}y\|_2 : \|y\|_2 = 1\}}{\min\{\|A^{\mathrm{T}}y\|_2 : \|y\|_2 = 1\}}.$$

## 7.3 Optimal preconditioning

The main source of difficulty in solving the canonical feasibility problems (2.2) and (2.3) is that the conditions number of the problem instance $A$ can be arbitrarily large even though $A$ is well-posed and its condition number is finite. This happens because $A$ can be poorly conditioned. Therefore, the ultimate goal of preconditioning is to reduce the condition number of the problem instance.

A good preconditioner is an approximation for $A$ which can be efficiently inverted, and chosen in a way that the preconditioned matrix has a better condition number. There are three general types of preconditioning for the original linear system $Ax = b$:

- Left preconditioning by a matrix $P$:

$$PAx = Pb.$$

- Right preconditioning by a matrix $D$:

$$ADz = b, \;\; x = Dz.$$

  The latter involves a substitution $z$ for the original variable $x$.

- Split (two-sided) preconditioning:

$$PADz = Pb, \;\; x = Dz.$$

Split preconditioning encompasses both the left and the right methods by setting $D = I$ or $P = I$, respectively.

An optimal preconditioning is a polynomial time procedure that makes the condition number of preconditioned matrix to be bounded above by $\mathcal{O}(n^c)$. Such preconditioning is called optimal because if $A$ is preconditioned in this way, we can automatically obtain an algorithm that solves the feasibility problems in polynomially many steps.

We next exhibit an optimal preconditioning procedures to solve (2.2) and (2.3) for which the condition number of the preconditioned matrix is bounded above by $\mathcal{O}(\sqrt{n})$.

**Theorem 11** *Assume* $A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} \in \mathbb{R}^{m\times n}$ *is full-rank and* $a_i \neq 0$ *for* $i = 1, \ldots, n$.

*(a) Assume* (2.2) *is feasible, i.e. there exists* $\bar{x} \in \mathbb{R}^n$ *such that* $A\bar{x} = 0$, $\bar{x} \geq 0$, $\bar{x} \neq 0$. *Let* $\hat{A}$ *be the matrix obtained after scaling the columns of* $A$ *by the diagonal matrix* $D = Diag(\bar{x})$ *and then balancing the rows of the resulting matrix – that is applying the QR-factorization to the rows of* $AD$ *such that the preconditioned matrix* $\hat{A}$ *satisfies* $\sigma_{max}(\hat{A}) = \sigma_{min}(\hat{A})$*. Then*

$$\mathcal{C}_R^{2,2}(\hat{A}) \leq \sqrt{n} + 1. \tag{7.4}$$

*(b) Assume* (2.3) *is feasible, i.e. there exists* $\bar{y} \in \mathbb{R}^m$ *with* $\|\bar{y}\| = 1$ *such that* $A^{\mathrm{T}}\bar{y} > 0$*. Let* $\hat{A}$ *be the matrix obtained after scaling the columns of* $A$ *by the diagonal matrix* $D = Diag\left(A^{\mathrm{T}}\bar{y}\right)^{-1}$ *and then scaling the resulting matrix by* $(I + \lambda\bar{y}\bar{y}^{\mathrm{T}})$ *for a sufficiently large* $\lambda > 0$*. Then*

$$\mathcal{C}_R^{2,2}(\hat{A}) \leq \sqrt{n} + 1. \tag{7.5}$$

**Proof:**

(a) Let $e \in \mathbb{R}^n$ denote the $n$-dimensional vector of all ones and $\tilde{A} = AD$. Clearly $\tilde{A}e = 0$. To prove (7.4), its sufficient to show if $v \in \mathbb{R}^m$ satisfies $\|v\| \leq \frac{1}{\sqrt{n}+1}$ then there exists $x \in \mathbb{R}^n_+$ with $\|x\| \leq 1$ such that $v = \hat{A}x$. Since if this condition holds, by Proposition 4.12 we have $\mathcal{C}_R^{2,2}(\hat{A}) \leq \sqrt{n} + 1$.

Let $v \in \mathbb{R}^m$ be an arbitrarily point with $\|v\| \leq \frac{1}{\sqrt{n}+1}$. Since the matrix $\hat{A}$ is balanced, there exists $u \in \mathbb{R}^n$ with $\|u\| \leq \|v\|$ that satisfies $\hat{A}u = v$. Let $x := u + \frac{1}{\sqrt{n}+1}e$. It can be easily verified that $x \geq 0$, $\|x\| \leq 1$, and $\hat{A}x = v$. Hence $v \in \left\{\hat{A}x \,:\, x \in \mathbb{R}^n_+,\ \|x\| \leq 1\right\}$ and the proof is complete.

(b) Let $e \in \mathbb{R}^n$ denote the $n$-dimensional vector of all ones and $\tilde{A} = AD$. Clearly $\tilde{A}^{\mathrm{T}}\bar{y} = e$. As $\lambda \to \infty$, a simple calculation shows that $\rho(\hat{A})$ tends to one. This can be easily verified by substituting $\hat{A} = \tilde{A}(I + \lambda\bar{y}\bar{y}^{\mathrm{T}})$ into (2.4). Therefore, $\mathcal{C}_{GCC}(\hat{A}) = \rho(\hat{A})^{-1} = 1$ for sufficiently large $\lambda$. The inequality (7.5) now follows from (7.1) and (7.2). Notice that the inequalities in (7.2) are tight for sufficiently large $\lambda$. ■

Although the above mentioned preconditioning procedures significantly improve the condition numbers of the matrix $A$, they are impractical since these preconditioners require the knowledge of the solutions $\bar{x}$ and $\bar{y}$ to systems (2.2) and (2.3) which we are trying to find and are unknown. To overcome this problem, we next show that a suitable combination of column normalization and row balancing precedures would always bring $\mathcal{C}_R$, $\mathcal{C}_{GCC}$, and $\mathcal{C}_{Gr}$ to roughly the same value without increasing the value of any of them. Hence if one of them is much larger than the other, in which case $A$ is certainly poorly conditioned, this simple combined preconditioner would rectify the poor conditioning.

## 7.4 Preconditioning via normalization and balancing

Inequalities (7.2) and (7.3) suggest two preconditioning procedures for improving Renegar's condition number $\mathcal{C}_R(A)$. The first one is to *normalize,* that is, scale the columns of $A$ so that the preconditioned matrix $\tilde{A}$ satisfies $\max\limits_{i=1,\dots,n} \|\tilde{a}_i\| = \min\limits_{i=1,\dots,n} \|\tilde{a}_i\|$. The second one is to *balance,* that is, apply an orthogonalization procedure to the rows of $A$ such as Gram-Schmidt or QR-factorization so that the preconditioned matrix $\tilde{A}$ satisfies $\sigma_{\max}(\tilde{A}) = \sigma_{\min}(\tilde{A})$. Observe that if the initial matrix $A$ is full rank and has non-zero columns, these properties are preserved by each of the above two preconditioning procedures. The following proposition formally states some properties of these two procedures.

**Proposition 5** *Assume $A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} \in \mathbb{R}^{m\times n}$ is full-rank and $a_i \neq 0$ for $i = 1,\dots,n$.*

**(a)** *Let $\tilde{A}$ be the matrix obtained after normalizing the columns of $A$, that is, $\tilde{a}_i = \frac{a_i}{\|a_i\|}$, $i = 1,\dots,n$. Then*

$$\mathcal{C}_R^{1,2}(\tilde{A}) = \mathcal{C}_{GCC}(\tilde{A}) = \mathcal{C}_{GCC}(A).$$

*This transformation is optimal in the following sense*

$$\mathcal{C}_R^{1,2}(\tilde{A}) = \min\{\mathcal{C}_R^{1,2}(AD) : D \text{ is diagonal positive definite}\}.$$

**(b)** *Let $\tilde{A}$ be the matrix obtained after balancing the rows of $A$, that is, $\tilde{A} = Q^{\mathrm{T}}$, where $QR = A^{\mathrm{T}}$ with $Q \in \mathbb{R}^{n\times m}, R \in \mathbb{R}^{m\times m}$ is the QR-decomposition of $A^{\mathrm{T}}$. Then*

$$\mathcal{C}_R^{2,2}(\tilde{A}) = \mathcal{C}_{Gr}(\tilde{A}) = \mathcal{C}_{Gr}(A).$$

*This transformation is optimal in the following sense*

$$\mathcal{C}_R^{2,2}(\tilde{A}) = \min\{\mathcal{C}_R^{2,2}(PA) : P \in \mathbb{R}^{m\times m} \text{ is non-singular}\}.$$

**Proof:** Parts (a) and (b) follow respectively from (7.2) and (7.3). ■

Observe that the normalization procedure transforms the solutions to the original problem (2.2) when this system is feasible. More precisely, observe that $Ax = 0,\ x \geq 0$ if and only if $\tilde{A}D^{-1}x = 0,\ D^{-1}x \geq 0$, where $D$ is the diagonal matrix whose diagonal entries are the norms of the columns of $A$. Thus a solution to the original problem (2.2) can be readily obtained from a solution to the column-normalized preconditioned problem: premultiply by $D$. At the same time, observe that the solution to (2.3) does not change when the columns of $A$ are normalized.

Similarly, the balancing procedure transforms the solutions to the original problem (2.3). In this case $A^{\mathrm{T}}y \geq 0,\ y \neq 0$ if and only if $\tilde{A}^{\mathrm{T}}R^{-1}y \geq 0,\ R^{-1}y \neq 0$, where $QR = A^{\mathrm{T}}$ is the QR-decomposition of $A^{\mathrm{T}}$. Hence a solution to the original problem (2.3) can be readily obtained from a solution to the row-balanced preconditioned problem: premultiply by $R$. At the same time, observe that the solution to (2.2) does not change when the rows of $A$ are balanced.

**Theorem 12** *Assume* $A = \begin{bmatrix} a_1 & \cdots & a_n \end{bmatrix} \in \mathbb{R}^{m\times n}$ *is full-rank and* $a_i \neq 0$ *for* $i = 1, \ldots, n$.

**(a)** *Let* $\hat{A}$ *be the matrix obtained after normalizing the columns of* $A$ *and then balancing the rows of the resulting matrix. Then*

$$\frac{1}{\sqrt{n}}\mathcal{C}_{GCC}(\hat{A}) \leq \mathcal{C}_R^{2,2}(\hat{A}) = \mathcal{C}_{Gr}(\hat{A}) \leq \sqrt{n}\mathcal{C}_{GCC}(A). \tag{7.6}$$

**(b)** *Let* $\hat{A}$ *be the matrix obtained after balancing the rows of* $A$ *and then normalizing the columns of the resulting matrix. Then*

$$\frac{1}{\sqrt{n}}\mathcal{C}_{Gr}(\hat{A}) \leq \mathcal{C}_R^{1,2}(\hat{A}) = \mathcal{C}_{GCC}(\hat{A}) \leq \sqrt{n}\mathcal{C}_{Gr}(A). \tag{7.7}$$

**Proof:**

**(a)** Let $\tilde{A}$ be the matrix obtained by normalizing the columns of $A$. Proposition 5(a) yields

$$\mathcal{C}_R^{1,2}(\tilde{A}) = \mathcal{C}_{GCC}(\tilde{A}) = \mathcal{C}_{GCC}(A). \tag{7.8}$$

Since $\hat{A}$ is obtained by balancing the rows of $\tilde{A}$, Proposition 5(b) yields

$$\mathcal{C}_R^{2,2}(\hat{A}) = \mathcal{C}_{Gr}(\hat{A}) = \mathcal{C}_{Gr}(\tilde{A}) \leq \mathcal{C}_R^{2,2}(\tilde{A}). \tag{7.9}$$

Inequality (7.6) now follows by combining (7.1), (7.2), (7.8), and (7.9).

**(b)** The proof this part is essentially identical to that of part (a) after using the parallel roles of the pairs $\mathcal{C}_{GCC}, \mathcal{C}_R^{1,2}$ and $\mathcal{C}_{Gr}, \mathcal{C}_R^{2,2}$ apparent from (7.2), (7.3), and Proposition 5.

■

Observe that both combined preconditioners in Theorem 12 transform $A$ to a $\hat{A} = PAD$ where $D$ is a diagonal matrix and $P$ is a square non-singular matrix. The particular $P$ and $D$ depend on what procedure is applied first. Hence each of the combined preconditioners transforms both sets of solutions to the original pair of problems (2.2) and (2.3). In this case, solutions to the original problems can be readily obtained from solutions to the preconditioned problems via premultiplication by $D$ for the solutions to (2.2) and by $P^{\mathrm{T}}$ for the solutions to (2.3).

As discussed in [5, Section 5], there is no general relationship between $\mathcal{C}_{Gr}$ and $\mathcal{C}_{GCC}$, meaning that either one can be much larger than the other in a dimension-independent fashion. Theorem 12(a) shows that when $\mathcal{C}_{GCC}(A) \ll \mathcal{C}_{Gr}(A)$ column normalization followed by row balancing would reduce both $\mathcal{C}_{Gr}$ and $\mathcal{C}_R^{1,2}$ while not increasing $\mathcal{C}_{GCC}$ beyond a factor of $\sqrt{n}$. Likewise, Theorem 12(b) shows that when $\mathcal{C}_{Gr}(A) \ll \mathcal{C}_{GCC}(A)$ row balancing followed by column normalization would reduce both $\mathcal{C}_{GCC}$ and $\mathcal{C}_R^{1,2}$ while not increasing $\mathcal{C}_{Gr}$ beyond a factor of $\sqrt{n}$. In other words, one of the two combinations of column normalization and by row balancing will always improve the values of all three condition numbers $\mathcal{C}_{GCC}(A)$, $\mathcal{C}_{Gr}(A)$ and $\mathcal{C}_R(A)$ modulo a factor of $\sqrt{n}$. The following two questions concerning a potential strengthening of Thorem 12 naturally arise:

(i) Does the order of row balancing and column normalization matter in each part of Theorem 12?

(ii) Are the leftmost and rightmost expressions in (7.6) and (7.7) within a small power of $n$? In other words, do the combined pre-conditioners make all $\mathcal{C}_{GCC}(A), \mathcal{C}_{Gr}(A)$ and $\mathcal{C}_R(A)$ the same modulo a small power of $n$?

The examples below, adapted from [5, Section 5], show that the answers to these questions are 'yes' and 'no' respectively. Hence without further assumptions, the statement of Theorem 12 cannot be strengthened along these lines.

**Example 2** Assume $\epsilon > 0$ and let $A = \begin{bmatrix} 2/\epsilon & 1 & 1 \\ 0 & -1 & 1 \end{bmatrix}$. It is easy to see that $\mathcal{C}_{GCC}(A) = \sqrt{2}$. After balancing the rows of $A$ and normalizing the columns of the resulting matrix we get

$$\hat{A} = \frac{1}{\sqrt{2(1+\epsilon^2)}} \begin{bmatrix} \sqrt{2(1+\epsilon^2)} & \epsilon & \epsilon \\ 0 & -\sqrt{2+\epsilon^2} & \sqrt{2+\epsilon^2} \end{bmatrix}.$$

It is easy to show that $\mathcal{C}_{GCC}(\hat{A}) = \frac{\sqrt{2(1+\epsilon^2)}}{\epsilon}$. Thus for $\epsilon > 0$ sufficiently small, $\mathcal{C}_{GCC}(\hat{A})$ can be arbitrarily larger than $\mathcal{C}_{GCC}(A)$. Therefore (7.6) does not hold for $A, \hat{A}$.

**Example 3** Assume $0 < \epsilon < 1/2$ and let $A = \begin{bmatrix} -\epsilon & -1 & 1 \\ 0 & -1 & 1+\epsilon \end{bmatrix}$. Using [5, Proposition 1.6], it can be shown that $\mathcal{C}_{Gr}(A) = \sqrt{2 + (1+\epsilon)^2}$ . After normalizing the columns of $A$ and balancing the rows of the resulting matrix, we obtain

$$\hat{A} = \frac{1}{\sqrt{\delta}} \begin{bmatrix} -\theta\delta & -\sqrt{2}\theta\epsilon\,(1+\epsilon) & -\theta\epsilon\sqrt{1+(1+\epsilon)^2} \\ 0 & -\sqrt{1+(1+\epsilon)^2} & \sqrt{2}\,(1+\epsilon) \end{bmatrix}.$$

where $\delta = 1 + 3(1+\epsilon)^2$ and $\theta = \frac{1}{\sqrt{\delta+\epsilon^2}}$. Using [5, Proposition 1.6] again, it can be shown that $\mathcal{C}_{Gr}(\hat{A}) = \sqrt{1+\frac{\delta}{\epsilon^2}}$. Therefore, $\mathcal{C}_{Gr}(\hat{A})$ can be arbitrarily larger than $\mathcal{C}_{Gr}(A)$. Hence (7.7) does not hold for $A, \hat{A}$.

**Example 4** Assume $\epsilon > 0$ and let $A = \begin{bmatrix} 1+\epsilon & 1+\epsilon & -1+\epsilon \\ -1 & -1 & 1 \end{bmatrix}$. In this case $\mathcal{C}_{GCC}(A) = \frac{\sqrt{2+2(1+\epsilon)^2}}{\epsilon}$. After normalizing the columns of $A$ and balancing the rows of the resulting matrix, we get

$$\hat{A} = \frac{1}{\sqrt{2\gamma^2+2\beta^2}} \begin{bmatrix} \beta & \beta & \sqrt{2}\gamma \\ -\gamma & -\gamma & \sqrt{2}\beta \end{bmatrix},$$

where $\beta = \sqrt{\frac{1+(1+\epsilon)^2}{2}}$ and $\gamma = \sqrt{1+(1-\epsilon)^2}$. It is easy to see that $\mathcal{C}_{GCC}(\hat{A}) = \sqrt{2}$. Therefore, $\mathcal{C}_{GCC}(\hat{A})$ can be arbitrarily smaller than $\mathcal{C}_{GCC}(A)$ in (7.6).

**Example 5** Assume $0 < \epsilon < 1/2$ and let $A = \begin{bmatrix} 2\epsilon & 1 & 1 \\ 0 & -1 & 1 \end{bmatrix}$. In this case $\mathcal{C}_{Gr}(A) = \sqrt{1+\frac{1}{2\epsilon^2}}$. After balancing the rows of $A$ and normalizing the columns of the resulting matrix we get

$$\hat{A} = \frac{1}{\sqrt{2(1+\epsilon^2)}} \begin{bmatrix} \sqrt{2(1+\epsilon^2)} & 1 & 1 \\ 0 & -\sqrt{1+2\epsilon^2} & \sqrt{1+2\epsilon^2} \end{bmatrix}.$$

Using [5, Proposition 1.6], it can be shown that $\mathcal{C}_{Gr}(\hat{A}) = \sqrt{2+\epsilon^2}$. Thus $\mathcal{C}_{Gr}(\hat{A})$ can be arbitrarily smaller than $\mathcal{C}_{Gr}(A)$ in (7.7).

# Chapter 8

# Future Research Directions

In this chapter, we discuss several future research that are related to the research involved in this thesis.

## 8.1 Re-scaled von Neumann algorithm

The re-scaled perceptron algorithm developed in Chapter 6 solves the feasibility problems $A^{\mathrm{T}}y > 0$ when it is feasible. The duality between the problems (2.2) and (2.3) naturally suggests the extension of this algorithm for the alternative system $Ax = 0$, $x \geq 0$, $x \neq 0$.

When the problem $Ax = 0$, $x \geq 0$, $x \neq 0$ is feasible, zero can be written as a convex combination of the columns of $A$. Based on the definition of $\rho(A)$ in (2.4), the problem instance is poorly conditioned, if zero is close to the boundary of the convex-hull of the normalized columns of $A$ and it is well-conditioned if zero is far away from the boundary. If the problem is well-conditioned, the standard von Neumann algorithm finds an $\epsilon$-solution to $Ax = 0$, $x \geq 0$, $\|x\| = 1$ quickly. However, when this problem is poorly conditioned, the von Neumann algorithm requires many iterations to find an $\epsilon$-solution. The poor conditioning of this problem is caused by: $(i)$ the eigenvalues of the matrix $A$ being very small, i.e. the columns of $A$ being nearly linearly dependent, or $(ii)$ some columns of $A$ being almost perpendicular to the boundary of the convex-hull of the normalized columns of $A$ that are close to zero. It appears that some simple pre-processing (in addition to the normalization of the columns of $A$) discussed in Chapter 7, may separate these two types of ill-conditioning and allow us to mimic the steps in the re-scaling perceptron algorithm. In particular, we could eliminate the source $(i)$ of ill-conditioning by balancing the matrix $A$ and the source $(ii)$ of ill-conditioning by applying the re-scaling procedure.

## 8.2 Elementary algorithms for ill-posed instances

As discussed in Chapter 2, the von Neumann and the perceptron algorithms solve (2.2) and (2.3) respectively, when these problems are feasible. However, in many cases, the matrix $A$ is ill-posed and the feasibility pair (2.2) and (2.3) do not have strict feasible solutions and the condition measure $1/\rho(A)$ is infinite. In this case, instead, there exists a partition $\Omega(A) = (B, N)$ of $B \cup N = \{1, \dots, n\}$ such that both systems

$$A_B x_B = 0, \;\; x_B > 0, \;\; x_N = 0, \tag{8.1}$$

and

$$A_N^{\mathrm{T}} y < 0, \;\; A_B^{\mathrm{T}} y = 0, \tag{8.2}$$

are feasible. Here $A_B$ denotes the sub-matrix of $A$ that only contains the columns of A whose indices are in $B$. $A_N$, $x_B$ and $x_N$ have similar interpretations. The partition $\Omega(A)$ is called the Goldman and Tucker partition and its existence can be proved using the Farkas' Lemma [28]. When the matrix $A \notin \sum$, the partition $\Omega(A)$ is trivial, that is, $N = \emptyset$ or $B = \emptyset$.

A modification of interior-point algorithm [66] finds the partition $\Omega(A)$ and the corresponding solutions to (8.1) and (8.2), in at most $\mathcal{O}\left(\sqrt{n}[\log n + \log(1/\bar{\rho}(A))]\right)$ iterations where the parameter $\bar{\rho}(A)$ proposed by Cheung et al. [23] is an extension of $\rho(A)$. Unlike $1/\rho(A)$, the condition measure $1/\bar{\rho}(A)$ is finite for ill-posed instances. An interesting research direction is developing an elementary algorithm to solve (8.1) and (8.2). A possible avenue, is to thus supplement the perceptron-von Neumann template in Chapter 2 with a periodic "partition identification phase" as used in [66].

## 8.3 Chubanov's algorithm

The relaxation method discussed in Chapter 2 solves convex feasibility problems $y \in F$ for a convex set $F \subset \mathbb{R}^m$. When $F$ is a Polyhedral, i.e. $F = \left\{y : A^{\mathrm{T}} y \geq b\right\}$ where $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^n$, the separating oracle in the relaxation method finds an inequality $a_j^{\mathrm{T}} y \geq b_j$ violated by current solution $y$ and the algorithm projects $y$ onto this hyperplane i.e. $y_+ = y + \lambda(b_j - a_j^{\mathrm{T}} y)a_j$. Agmon [1], and Motzkin and Schoenberg [45] showed that, in this case, the relaxation method generates a sequence of points which convergences, in the limit, to a feasible point in $F$.

In late 2010, Sergei Chubanov [25] presented a variant of the relaxation algorithm for solving the polyhedral feasibility problem that was based on a divide-and-conquer (Chubanov's D&C) paradigm. This algorithm either finds a solution to $F = \{y : A^{\mathrm{T}} y \geq b, C^{\mathrm{T}} y = d\}$ with $A \in \mathbb{Z}^{m\times n}$, $C \in \mathbb{Z}^{m\times k}$, $b \in \mathbb{Z}^n$, and $d \in \mathbb{Z}^k$ or determines that this system has no integer solution. The key idea of Chubanov's algorithm is its use of new induced inequalities. Unlike Motzkin and Schoenberg's [45] algorithm that only projects the current solution onto the original violated hyperplane, Chubanov constructs new valid inequalities along the way and projects onto them as shown in Figure 8.1.

Chubanov's D&C procedure is the main ingredient in the Chubanov relaxation algorithm and its aim is to achieve the following. Given a current solution $y$, a radius $r$ and

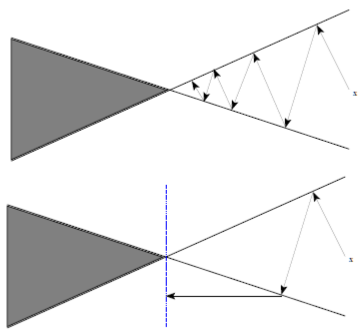


Figure 8.1: Chubanov's method generates new inequalities on the way.

an error bound $\epsilon > 0$, the algorithm will either:

(1) Find an $\epsilon$-solution $\bar{y} \in \mathbb{B}(y, r)$ to $F$, i.e. some $\bar{y}$ such that $A^{\mathrm{T}}\bar{y} \geq b + \epsilon\mathbf{1}$ and $C^{\mathrm{T}}\bar{y} = d$.

(2) Or find an induced hyperplane that separates $\mathbb{B}(y, r)$ from $F$.

When $r$ is very small, this task can be simply obtained by a stronger modification of separating oracle. However, for an arbitrary $r$, this task is achieved using a recursive algorithm. Since D&C returns $\epsilon$-solutions, it can be run on the system $A^{\mathrm{T}}y \geq b - \epsilon\mathbf{1}$, $C^{\mathrm{T}}y = d$; if the algorithm returns an $\epsilon$-solution, we will have an exact solution for the original system $A^{\mathrm{T}}y \geq b$, $C^{\mathrm{T}}y = d$.

Using these results, Chubanov showed that if D&C algorithm cannot find a feasible solution, either there exists an inequality in $A^{\mathrm{T}}y \geq b$ that is an implied equality, i.e., there exists $j \in \{1, \ldots, n\}$ such that $a_j^{\mathrm{T}}y = b_j$ for all feasible solutions, or there exists $j \in \{1, \ldots, n\}$ such that $a_j^{\mathrm{T}}y = b_j$ for all integer solutions to $F$.

The advantage of this algorithm is that it leads to a strongly polynomial algorithm for a certain class of linear feasibility problems. Chubanov showed that when the inequalities take a form $0 \leq y \leq 1$, his relaxation method either finds a solution to $F = \{y : C^{\mathrm{T}}y = d, 0 \leq y \leq 1\}$ or determines in strongly polynomial time that this system has no 0,1-solutions. Basu et al. [7] recently presented new proofs and interpretations of some of Chubanov's results. They extended this result and showed that when $F = \{y : C^{\mathrm{T}}y = d, 0 \leq y \leq 1\}$, if the matrix $C$ is totally uni-modular, then Chubanov's algorithm is strongly polynomial. In other words, this algorithm either finds a solution to this set or determines that this system has no solution in strongly polynomial time when $C$ is totally uni-modular.

Analysis in both Chubanov's [25] and Basu et al.'s [7] papers is restricted to integral data and relies on bit-length complexity bounds. The complexity of Chubanov's algorithm is based on the maximum determinant of all sub-matrices of $C$ which can be exponentially large. It would be interesting to analyze the performance of Chubanov's algorithm in terms of a more natural notion of real-number complexity based on condition numbers of the input data [18]. This would extend the ideas and results of Chubanov's algorithm for instances with real-numbers. Furthermore, it might shed new light into Chubanov's divide-and-conquer algorithm which is the driving idea behind Chubanov's results.

# References

[1] S. Agmon. The relaxation method for linear inequalities. *Canadian J. of Math.*, 6(3):382–392, 1954.

[2] F. Alizadeh and D. Goldfarb. Second-order cone programming. *Math. Program.*, 95(1):3–51, 2003.

[3] E. Amaldi, P. Belotti, and R. Hauser. A randomized algorithm for the maxfs problem. *In* IPCO, pages 249–264, 2005.

[4] E. Amaldi and R. Hauser. Boundedness theorems for the relaxation method. *Math. Oper. Res.*, 30(4).

[5] D. Amelunxen and P. Bürgisser. A coordinate-free condition number for convex programming. *SIAM J. Optim.*, 22(3):1029–1041, 2012.

[6] D. Amelunxen and P. Bürgisser. Probabilistic analysis of the Grassmann condition number. Technical report, University of Paderborn, Germany, 2013.

[7] A. Basu, J. De Loera, and M. Junod. On chubanov's method for linear programming. *arXiv preprint arXiv:1204.2031*, 2012.

[8] H. H. Bauschke and J. M. Borwein. Legendre functions and the method of random Bregman projections. *J. Convex Anal.*, 4:27–67, 1997.

[9] H. H. Bauschke, J. M. Borwein, and A. Lewis. The method of cyclic projections for closed convex sets in Hilbert space. *Contemporary Math.*, 4:1–38, 1997.

[10] A. Belloni and R. Freund. A geometric analysis of renegar's condition number, and its interplay with conic curvature. *Math. Program.*, 119:95–107, 2009.

[11] A. Belloni and R. Freund. Projective re-normalization for improving the behavior of a homogeneous conic linear system. *Math. Program.*, 118:279–299, 2009.

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

[13] A. Ben-Tal and A. Nemirovski. Non-euclidean restricted memory level method for large-scale convex optimization. *Math. Program.*, 102(3):407–456, 2005.

[14] U. Betke. Relaxation, new combinatorial and polynomial algorithms for the linear feasibility problem. *Discrete & Computational Geometry*, 32(3):317–338, 2004.

[15] H. D. Block. The perceptron: A model for brain functioning. *Reviews of Modern Physics*, 34:123–135, 1962.

[16] A. Blum. *On-line algorithms in machine learning.* Springer, 1998.

[17] A. Blum, A. Frieze, R. Kannan, and S. Vempala. A polynomial-time algorithm for learning noisy linear threshold functions. *Algorithmica*, 22:35–52, 1998.

[18] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation.* Springer-Verlag, 1998.

[19] P. Burgisser and F. Cucker. *Condition.* Forthcoming.

[20] C. Chang and CJ. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[21] D. Cheung and F. Cucker. A new condition number for linear programming. *Math. Program.*, 91(1):163–174, 2001.

[22] D. Cheung, F. Cucker, and J. Peña. A condition number for multifold conic systems. *SIAM J. Optim.*, 19(1):261–280, 2008.

[23] D. Cheung, F. Cucker, and J. Peña. On strata of degenerate polyhedral cones I: Condition and distance to strata. *Europea. J. Oper. Res.*, 198:23–28, 2009.

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

[25] S. Chubanov. A strongly polynomial algorithm for linear systems having a binary solution. *Math. Program.*, 134(2):533–570, 2012.

[26] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[27] B. Cox, A. Juditsky, and A. Nemirovski. Dual subgradient algorithms for large-scale nonsmooth learning problems. *Math. Program.*, pages 1–38, 2013.

[28] F. Cucker D. Cheung and J. Pe na. Unifying condition numbers for linear programming. *Math. Oper. Res.*, 28:609–624, 2003.

[29] G. B. Dantzig. An $\epsilon$-precise feasible solution to a linear program with a convexity constraint in $\frac{1}{\epsilon^2}$ iterations independent of problem size. Technical report, Technical Report, Stanford University, 1992.

[30] J. Dunagan and S. Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. *Math. Program.*, 114(1):101–114, 2006.

[31] M. Epelman and R. Freund. Condition number complexity of an elementary algorithm for computing a reliable solution of a conic linear system. *Math. Program.*, 88(3):451–485, 2000.

[32] M. Epelman and R. Freund. A new condition measure, preconditioners, and relations between different measures of conditioning for conic linear systems. *SIAM J. Optim.*, 12:627–655, 2002.

[33] R. Freund and J. Vera. Condition-based complexity of convex optimization in conic linear form via the ellipsoid algorithm. *SIAM J. Optim.*, 10:155–176, 1999.

[34] Y. Freund and R. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37:277–296, 1999.

[35] A. Gilpin, J. Peña, and T. Sandholm. First-order algorithm with $\mathcal{O}(\ln(1/\epsilon))$ convergence for $\epsilon$-equilibrium in two-person zero-sum games. *Math. Program.*, 133:279–298, 2012.

[36] J. L. Goffin. The relaxation method for solving systems of linear inequalities. *Math. Oper. Res.*, pages 388–414, 1980.

[37] G. Golub and C. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, third edition, 1996.

[38] M. Jaggi. Revisiting {Frank-Wolfe}: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 427–435, 2013.

[39] T. Joachims. Making large scale svm learning practical. 1999.

[40] T. Joachims. Training linear svms in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226. ACM, 2006.

[41] A. Juditsky and A. Nemirovski. First order methods for nonsmooth convex large-scale optimization, ii: utilizing problems structure. *Optim. for Machine Learning*, pages 149–183, 2011.

[42] S. Kakade. Perceptron lower bound & the winnow algorithm @http://stat.wharton.upenn.edu/skakade/courses/stat928/lectures/lecture19.pdf, 2011.

[43] G. Lan, Z. Lu, and R. Monteiro. Primal-dual first-order methods with $\mathcal{O}(1/\epsilon)$ iteration-complexity for cone programming. *Math. Program.*, Series A:DOI 10.1007/s10107–008–0261–6, 2009.

[44] D. Li and T. Terlaky. The duality between the perceptron algorithm and the von neumann algorithm. In *Modeling and Optimization: Theory and Applications (MOPTA) conference.*

[45] T. S. Motzkin and I. J. Schoenberg. The relaxation method for linear inequalities. *Canadian J. of Math.*, 6(3):393–404, 1954.

[46] A. Nemirovski. Prox-method with rate of convergence $\mathcal{O}(1/t)$ for variational inequalities with Lipschitz-continuous monotone operators and smooth convex-concave saddle point problems. *SIAM J. Optim.*, 15(1):229–251, 2004.

[47] Y. Nesterov. A method for unconstrained convex minimization problem with rate of convergence $\mathcal{O}(1/k^2)$. Doklady AN SSSR (in russian). *(English translation. Soviet Math. Dokl.)*, 269:543–547, 1983.

[48] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer, 2004.

[49] Y. Nesterov. Excessive gap technique in nonsmooth convex minimization. *SIAM J. Optim.*, 16(1):235–249, 2005.

[50] A. B. J. Novikoff. On convergence proofs on perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume XII, pages 615–622, 1962.

[51] B. O'Donoghue and E. J. Candès. Adaptive restart for accelerated gradient schemes. *Foundations of Computational Mathematics, To Appear.*

[52] J. Peña and J. Renegar. Computing approximate solutions for convex conic systems of constraints. *Math. Program*, 87:351–383, 2000.

[53] J. Peña, V. Roshchina, and N. Soheili. Some preconditioners for systems of linear inequalities. *Optimization Letters*, pages 1–8, 2013.

[54] J. Peña and N. Soheili. A deterministic rescaled perceptron algorithm. Forthcoming.

[55] J. C. Platt. Fast training of support vector machines using sequential minimal optimization. 1999.

[56] A. Rakhlin. Lecture notes on online learning @http://www-stat.wharton.upenn.edu/rakhlin/papers/onlinelearning.pdf. *Draft, April*, 2009.

[57] J. Renegar. Incorporating condition measures into the complexity theory of linear programming. *SIAM J. Optim.*, 5:506–524, 1995.

[58] J. Renegar. Linear programming, complexity theory and elementary functional analysis. *Math. Program.*, 70:279–351, 1995.

[59] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Cornell Aeronautical Laboratory, Psychological Review*, 65(6):386–408, 1958.

[60] B. Schölkopf, K. Tsuda, and JP. Vert. *Kernel methods in computational biology.* MIT press, 2004.

[61] S. Shalev-Shwartz. Online learning and online convex optimization. *Foundations and Trends in Machine Learning*, 4(2):107–194, 2011.

[62] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Math. program.*, 127(1):3–30, 2011.

[63] A. J. Smola, S. Vishwanathan, and Q. V. Le. Bundle methods for machine learning. In *NIPS*, volume 20, pages 1377–1384, 2007.

[64] N. Soheili and J. Peña. A smooth perceptron algorithm. *SIAM J. Optim.*, 22(2):728–737, 2012.

[65] N. Soheili and J. Peña. A primal–dual smooth perceptron–von neumann algorithm. In *Discrete Geometry and Optimization*, pages 303–320. Springer, 2013.

[66] N. Soheili and J. Peña. A condition-based algorithm for solving polyhedral feasibility problems. *J. of Complexity*, 2014.

[67] L. Trefethen and D. Bau. *Numerical Linear Algebra.* SIAM, 1997.

[68] P. Tseng. On accelerated proximal gradient methods for convex-concave optimization. *SIAM J. Optim. (Submitted)*, 2008.

[69] V. Vapnik. *The nature of statistical learning theory.* springer, New York, 1995.

[70] V. Vapnik and A. J Chervonenkis. Theory of pattern recognition [in russian]. 1974.

[71] S. J. Wright. *Primal-dual interior-point methods*, volume 54. Siam, 1997.

[72] D. Yudin and A. Nemirovskii. Informational complexity and efficient methods for solving complex extremal problems. *Matekon*, 13(3):25–45, 1977.