2015

The Majorization Minimization Principle and Some Applications in Convex Optimization

Daniel Giles
Portland State University

The Majorization-Minimization Priniciple and Some Applications in Convex Optimization

by

Daniel Giles

An undergraduate honors thesis submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

University Honors

and

Mathematics

Thesis Advisor

Dr. Mau Nam Nguyen

Portland State University

2015

The Majorization-Minimization Priniciple and Some Applications in Convex Optimization

Daniel Giles

Advisor: Dr. Mau Nam Nguyen

Portland State University, 2015

ABSTRACT

The majorization-minimization (MM) principle is an important tool for developing algorithms to solve optimization problems. This thesis is devoted to the study of the MM principle and applications to convex optimization. Based on some recent research articles, we present a survey on the principle that includes the geometric ideas behind the principle as well as its convergence results. Then we demonstrate some applications of the MM principle in solving the feasible point, closest point, support vector machine, and smallest intersecting ball problems, along with sample MATLAB code to implement each solution. The thesis also contains new results on effective algorithms for solving the smallest intersecting ball problem.

Contents

Introduction

Optimization problems seek the 'best' solution when minimizing or maximizing a function—the point at which the function value is either minimal or maximal. Problems of this class have far-reaching practical applications, and the mathematics behind them is a rich and still-developing field. Convex optimization focuses on a subset of these problems in which the function to be minimized exhibits a specific structure. The properties which result from convexity bestow many benefits to our efforts. These include accelerating the algorithms which find our solution, calculating explicit bounds on any error in our approximations, as well as proving that our solutions are indeed optimal, to name but a few.

We hope to contribute to the subject by discussing known results in convex optimization at a level appropriate for undergraduate students, and by presenting new algorithms for solving the smallest intersecting ball problem. This paper relies on Mordukhovich & Nam (2014), and is in part a compilation of results found in Chi, Zhou & Lange (2014), and Mairal (2013). Their inclusion here is intended to support this text as a self-contained introduction to the subject, guiding the readers from the foundational concepts to computing numerical solutions for common problems. The paper is organized as follows. Chapter 1 is an introduction to the terminology and primary results in the field of convex optimization. Chapter 2 is devoted to presenting the majorization-minimization principle and providing sufficient conditions for its convergence. We see here that the gradient method is a special case of the MM principle, and discuss Nesterov's accelerated gradient method (1983). Chapter 3 surveys four problems in optimization and demonstrates how the MM principle is used to find their solutions. Chapter 4 implements the resulting algorithms in the *MATLAB* programming language, and demonstrates empirical results for sample problems.

Chapter 1

Fundamentals of Convex Optimization

An optimization problem asks, given some function $f : \Omega \subset \mathbb{R}^n \to \mathbb{R}$, at which argument $x \in \mathbb{R}^n$ does the function attain its minimum or maximum? The function value $f(x)$ at this point is called the optimal value of f. We refer to f as the *objective function*, or in some cases the *cost* or *loss* function. We will see that when the objective function satisfies a property we call convexity, any local minimum is a global minimum. In *constrained optimization* problems, we limit ourselves to finding the best solution that exists only in a given *constraint* set, for example:

$$\min_{x \in \mathcal{C}} \ell(x) \quad \text{where } \mathcal{C} = \bigcap_{i=1}^{m} \mathcal{C}_i \subset \mathbb{R}^n.$$

One way of dealing with a constraint is by the penalty method (see [4], [7]) wherein a new function $f(x)$ is defined as the sum of $\ell(x)$ and the $\mu > 0$ weighted sum of distances from x to all constraint sets. This allows us to rephrase the problem as

$$\min f(x) := \ell(x) + \mu \sum_{i=1}^{m} \operatorname{dist}(x, \mathcal{C}_i).$$

By minimizing f, we also minimize the distance from its argument to the constraint sets. If the intersection of these sets is non-empty, then the penalty term, and thus the distance to each constraint set, can be driven to zero.

Let us now familiarize ourselves with some important terminology and results from convex

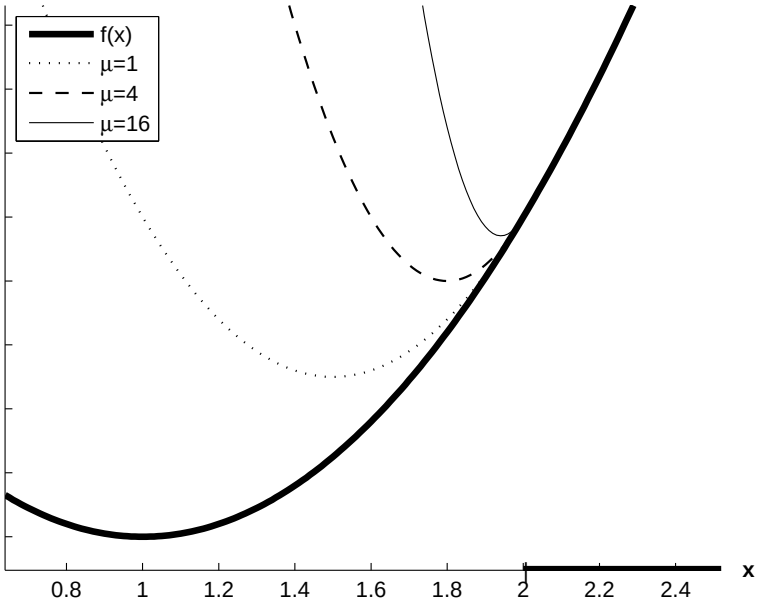


FIGURE 1.0.1: Using the penalty method to minimize $f(x)$ with the constraint that $x \in \Omega = \{x \mid x \geq 2\}$. As $\mu \to \infty$, minimizing $f(x) + \frac{\mu}{2}\text{dist}(x, \Omega)^2$ satisfies the constraint.

analysis. In all cases, $\|\cdot\|$ denotes the Euclidean norm, and $\langle \cdot, \cdot \rangle$ denotes the dot product in Euclidean space. All functions under consideration are defined on $\mathbb{R}^n$ unless otherwise stated.

Definition 1.0.1. A function $f : \mathbb{R}^n \to \mathbb{R}$ is called a **convex function** if for all $x, y \in \mathbb{R}^n$ and all $t \in [0, 1]$,

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y).$$

If strict inequality holds for all $x \neq y$, we say that f is **strictly convex**.

Remark 1.0.1. For $0 \leq t \leq 1$, the term $tx + (1-t)y$ lies on the line segment connecting x and y, and it is sometimes helpful to see that as $tx + (1-t)y = y + t(x-y)$, the function f is convex if

$$f(y + t(x-y)) \leq f(y) + t(f(x) - f(y))$$

for all $t \in [0, 1]$ and $x, y \in \mathbb{R}^n$.

Let us demonstrate convexity with a simple example, and derive sufficient conditions for convexity for the sum and composition of functions.

Example 1.0.2. The function $f(x) = \|x\|$ is convex. Indeed, using the triangle inequality we have, for $t \in (0, 1)$ and $x, y \in \mathbb{R}^n$,

$$f(tx + (1-t)y) = \|tx + (1-t)y\| \leq t\|x\| + (1-t)\|y\| = tf(x) + (1-t)f(y).$$

Proposition 1.0.3. If f and g are convex, then $f + g$ is convex.

Proof. Take any x, y and $t \in [0, 1]$. Then we have

$$\begin{aligned}[f + g](x + t(y - x)) &= f(x + t(y - x)) + g(x + t(y - x)) \\ &\leq f(x) + t(f(y) - f(x)) + g(x) + t(g(y) - g(x)) \\ &= [f + g](x) + t([f + g](y) - [f + g](x)),\end{aligned}$$

which satisfies the definition of convexity for $f + g$. □

Example 1.0.4. Let $f(x) = \langle x, c\rangle$, be a function from $\mathbb{R}^n$ to $\mathbb{R}$, with $c \in \mathbb{R}^n$. Then f is convex. Indeed, for any $t \in [0, 1]$ and $x, y \in \mathbb{R}^n$,

$$f(x + t(y - x)) = \langle x + t(y - x), c\rangle = \langle x, c\rangle + t\big(\langle y, c\rangle - \langle x, c\rangle\big) = f(x) + t(f(y) - f(x)).$$

This satisfies the definition of convexity.

Definition 1.0.5. A function $B : \mathbb{R}^n \to \mathbb{R}^k$ is **affine** if $B(x) = A(x) + b$, where A is a linear mapping from $\mathbb{R}^n$ to $\mathbb{R}^k$ and $b \in \mathbb{R}^k$.

Proposition 1.0.6. If a mapping $B : \mathbb{R}^n \to \mathbb{R}^k$ is affine, then

$$B(y + t(x - y)) = B(y) + t(B(x) - B(y))$$

for all $x, y \in \mathbb{R}^n$ and all $t \in \mathbb{R}$.

Proof. Suppose B is affine, so there exist some $b \in \mathbb{R}^k$ and a linear map A such that $B(x) = A(x) + b$. Take any $t \in \mathbb{R}$ and $x, y \in \mathbb{R}^n$. Then

$$B(y + t(x - y)) = A(y + t(x - y)) + b.$$

Since A is linear, we have

$$B(y + t(x - y)) = A(y) + t\big(A(x) - A(y)\big) + b.$$

Using $A(x) = B(x) - b$, we find that

$$B(y + t(x - y)) = B(y) + t\big(B(x) - B(y)\big),$$

which is the desired equality. □

Proposition 1.0.7. If $f : \mathbb{R}^n \to \mathbb{R}$ is convex and $B : \mathbb{R}^k \to \mathbb{R}^n$ is affine, then $f \circ B$ is convex.

Proof. Let $x, y \in \mathbb{R}^k$ and $t \in [0,1]$. Then using Proposition 1.0.6,

$$\begin{aligned}[f \circ B](x + t(y - x)) &= f\big(B(x + t(y - x))\big) \\ &= f\big(B(x) + t(B(y) - B(x))\big).\end{aligned}$$

Since as f is convex, we have

$$\begin{aligned}[f \circ B](x + t(y - x)) &\leq f(B(x)) + t\big(f(B(y)) - f(B(x))\big) \\ &= [f \circ B](x) + t\big([f \circ B](y) - [f \circ B](x)\big),\end{aligned}$$

which satisfies the definition of convexity for $f \circ B$. □

Definition 1.0.8. A set $\Omega \subset \mathbb{R}^n$ is a **convex set** if for all $x, y \in \Omega$ and all $t \in [0,1]$,

$$tx + (1 - t)y \in \Omega.$$

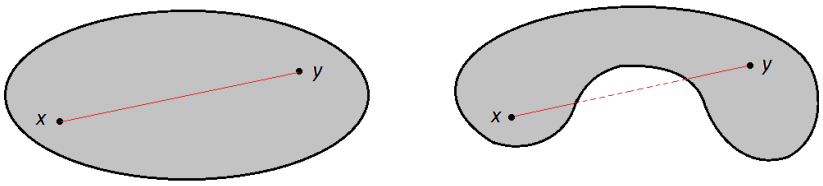


FIGURE 1.0.2: A convex (*left*) and non-convex (*right*) set. A set is convex if the line segment connecting any two elements is contained in the set.

Definition 1.0.9. For $x \in \mathbb{R}^n$, its **Euclidean distance to a set** $\Omega \subset \mathbb{R}^n$ is defined by

$$\operatorname{dist}(x, \Omega) := \inf\{\|\omega - x\| \;\big|\; \omega \in \Omega\}.$$

Definition 1.0.10. The **Euclidean projection** of $x \in \mathbb{R}^n$ onto a set $\Omega \subset \mathbb{R}^n$ is

$$P_\Omega(x) := \{\omega \in \Omega \;\big|\; \|\omega - x\| = \operatorname{dist}(x, \omega)\}.$$

In [10] it is shown that the Euclidean projection onto a nonempty closed convex set is a singleton.

Remark 1.0.2. It follows from the definitions of distance and the Euclidean projection that

$$\operatorname{dist}(x, \Omega) = \|x - P_\Omega(x)\|$$

for a nonempty closed convex set Ω. Here and thereafter we identify the set $P_\Omega(x)$ with its unique element.

Definition 1.0.11. The **closed ball** centered at $\bar{x}$ with radius $r > 0$ is the set

$$\mathbb{B}(\bar{x}, r) := \{x \in \mathbb{R}^n \mid \|x - \bar{x}\| \leq r\}.$$

Definition 1.0.12. A **hyperplane** H in $\mathbb{R}^n$ is the set

$$H := \{x \in \mathbb{R}^n \mid \langle n, x\rangle = b\}.$$

Here $b \in \mathbb{R}$ is a constant, and $n \neq 0$ is a normal vector to the hyperplane. Further, each hyperplane defines a **halfspace** as the set of all $x \in \mathbb{R}^n$ where $\langle n, x\rangle \leq b$. The second halfspace is defined as the set of all $x \in \mathbb{R}^n$ such that $\langle n, x\rangle \geq b$.

Definition 1.0.13. A function $f : \mathbb{R}^n \to \mathbb{R}$ has a **local minimum** at $\bar{x} \in \mathbb{R}^n$ if there exists $r > 0$ such that

$$f(\bar{x}) \leq f(x) \text{ for all } x \in \mathbb{R}^n \text{ such that } \|\bar{x} - x\| < r.$$

Definition 1.0.14. A function $f : \mathbb{R}^n \to \mathbb{R}$ has an **absolute minimum** at $\bar{x} \in \mathbb{R}^n$ if

$$f(\bar{x}) \leq f(x) \text{ for all } x \in \mathbb{R}^n.$$

Proposition 1.0.15. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. If f has a local minimum at x^*, then it has an absolute minimum at x^*.

Proof. Let x^* be a local minimizer of f. Then there is some $r > 0$ such that $\|x^* - z\| < r$ implies $f(z) \geq f(x^*)$. Now take $y \in \mathbb{R}^n$ and choose $z = x^* + t(y - x^*)$ with $t \in (0, 1)$ sufficiently small that $\|z - x^*\| < r$. Then by convexity we have

$$f(z) = f(x^* + t(y - x^*)) \leq f(x^*) + t\big(f(y) - f(x^*)\big).$$

Since $f(z) - f(x^*) \geq 0$, we have $t(f(y) - f(x^*)) \geq 0$, so $f(y) \geq f(x^*)$. With y chosen arbitrarily, x^* is indeed an absolute minimizer of f. □

As we discuss differentiable functions, let C^1 denote the set of all functions whose partial derivatives exist and are continuous. We refer to $f \in C^1$ as a C^1 function.

Definition 1.0.16. For a C^1 function $f : \mathbb{R}^n \to \mathbb{R}$, the **gradient** of f at a point $x = [x_1, ..., x_n]$ is

$$\nabla f(x) = \left[\tfrac{\partial f}{\partial x_1}(x), \quad \cdots \quad , \tfrac{\partial f}{\partial x_n}(x)\right].$$

For C^1 functions f, g from $\mathbb{R}^n$ to $\mathbb{R}$, a scalar c, and a C^1 function $F : \mathbb{R} \to \mathbb{R}$, we recall the following rules from elementary calculus:

(a) $\nabla(f + g) = \nabla f + \nabla g$
(b) $\nabla(cf) = c\nabla f$
(c) $\nabla[F \circ f](x) = F'(f(x))\nabla f(x)$.

Example 1.0.17. If $f(x) = \|x - c\|^2$, where $x, c \in \mathbb{R}^n$ and c is a constant, then

$$\nabla f(x) = 2(x - c).$$

To verify this, from $f(x) = \|x - c\|^2 = (x_1 - c_1)^2 + \cdots + (x_n - c_n)^2$, we easily see

$$\nabla f(x) = [2(x_1 - c_1), \ldots, 2(x_n - c_n)] = 2(x - c).$$

We will now see that the gradient of a function allows us to characterize convex functions as those functions which are bound below by their linearization at any point.

Proposition 1.0.18. A function $f \in C^1$ is convex if and only if for all $x, y \in \mathbb{R}^n$,

$$f(x) \geq f(y) + \langle \nabla f(y), x - y \rangle.$$

Proof. Let $f \in C^1$. First suppose f is convex. Then for all x, y and any $t \in (0, 1)$, we have

$$f(x) + t\big(f(y) - f(x)\big) \geq f(x + t(y - x)),$$

and simple rearrangement gives us

$$f(y) \geq f(x) + \frac{f(x + t(y - x)) - f(x)}{t}.$$

We may now take the limit as $t \to 0$, and find

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle.$$

For the converse, suppose f satisfies the given inequality. Let $x, y \in \mathbb{R}^n$, and let $z = tx + (1 - t)y$ be some convex combination thereof, with $0 \leq t \leq 1$. By assumption we have

$$f(x) \geq f(z) + \langle \nabla f(z), x - z \rangle$$

and

$$f(y) \geq f(z) + \langle \nabla f(z), y - z \rangle.$$

Multiplying these inequalities by t and $(1 - t)$, respectively, we obtain

$$tf(x) \geq tf(z) + \langle \nabla f(z), tx - tz \rangle$$

and

$$(1 - t)f(y) \geq (1 - t)f(z) + \langle \nabla f(z), (1 - t)y - z + tz \rangle.$$

Taking the sum of these gives us

$$tf(x) + (1 - t)f(y) \geq f(z) + \langle \nabla f(z), tx + (1 - t)y - z \rangle = f(z) = f(tx + (1 - t)y),$$

which satisfies the definition of convexity. □

Definition 1.0.19. A function $g : \mathbb{R}^n \to \mathbb{R}$ is $\sigma > 0$-**strongly convex** if for all $x, y \in \mathbb{R}^n$ and all $t \in [0, 1]$,

$$g(tx + (1-t)y) \leq tg(x) + (1-t)g(y) - \frac{\sigma}{2}t(1-t)\|x-y\|^2.$$

We now demonstrate some necessary and sufficient conditions of strong convexity, including a useful lower bound on strongly convex functions.

Proposition 1.0.20. A function $f : \mathbb{R}^n \to \mathbb{R}$ is σ-strongly convex if for all $c \in \mathbb{R}^n$, the function

$$g(x) := f(x) - \frac{\sigma}{2}\|x-c\|^2$$

is convex.

Proof. Let f be a function from $\mathbb{R}^n$ to $\mathbb{R}$ and suppose for all $c \in \mathbb{R}^n$, the function $g(x) := f(x) - \frac{\sigma}{2}\|x-c\|^2$ is convex. Let $x, y \in \mathbb{R}^n$ and $t \in [0, 1]$. Now by the convexity of g, we have

$$g(y + t(x-y)) \leq g(y) + t(g(x) - g(y)).$$

Applying the definition of g, we have for any c that

$$f(y+t(x-y)) - \frac{\sigma}{2}\|y+t(x-y)-c\|^2 \leq f(y) - \frac{\sigma}{2}\|y-c\|^2 + t\big(f(x) - \frac{\sigma}{2}\|x-c\|^2 - f(y) + \frac{\sigma}{2}\|y-c\|^2\big).$$

Now consider the case where $c = y$. This gives us

$$f\big(y + t(x-y)\big) - \frac{\sigma}{2}\|t(x-y)\|^2 \leq f(y) + t\big(f(x) - \frac{\sigma}{2}\|x-y\|^2 - f(y)\big),$$

which is equivalent to

$$\begin{aligned} f(y + t(x-y)) &\leq f(y) + t\big(f(x) - f(y)\big) - t\frac{\sigma}{2}\|x-y\|^2 + \frac{\sigma}{2}t^2\|x-y\|^2 \\ &= f(y) + t\big(f(x) - f(y)\big) - t(1-t)\frac{\sigma}{2}\|x-y\|^2. \end{aligned}$$

This is the strong convexity of f. □

Proposition 1.0.21. Let $f : \mathbb{R}^n \to \mathbb{R}$ and $\sigma > 0$. If

$$f(x) - \frac{\sigma}{2}\|x\|^2$$

is convex, then f is σ-strongly convex.

Proof. Suppose $f(x) - \frac{\sigma}{2}\|x\|^2$ is convex. Take any $c \in \mathbb{R}^n$, and consider that

$$\begin{aligned} f(x) - \frac{\sigma}{2}\|x-c\|^2 &= f(x) - \frac{\sigma}{2}\big(\|x\|^2 - 2\langle x, c\rangle + \|c\|^2\big) \\ &= f(x) - \frac{\sigma}{2}\|x\|^2 + \sigma\big(\langle x, c\rangle - \frac{1}{2}\|c\|^2\big). \end{aligned}$$

Now $f(x) - \frac{\sigma}{2}\|x\|^2$ is convex by assumption, and $\sigma\big(\langle x, c\rangle - \frac{1}{2}\|c\|^2\big)$ is convex, so their sum $f(x) - \frac{\sigma}{2}\|x - c\|$ is also convex, regardless of c. So (by Proposition 1.0.20) f is strongly convex. □

Proposition 1.0.22. A C^1 function $f : \mathbb{R}^n \to \mathbb{R}$ is σ-strongly convex if and only if for all $x, y \in \mathbb{R}^n$,

$$f(y) \geq f(x) + \langle \nabla f(x), y - x\rangle + \frac{\sigma}{2}\|y - x\|^2.$$

Proof. Suppose f is strongly convex with parameter σ. Then by Proposition 1.0.20, $g(y) = f(y) - \frac{\sigma}{2}\|x - y\|^2$ is convex for any x. By Proposition 1.0.18, $g(y) \geq g(x) + \langle \nabla g(x), y - x\rangle$ and using our definition of g, we obtain

$$f(y) - \frac{\sigma}{2}\|y - x\|^2 \geq f(x) - \frac{\sigma}{2}\|x - x\|^2 + \langle \nabla(f(x) - \frac{\sigma}{2}\|x - x\|^2), y - x\rangle.$$

This however simplifies to

$$f(y) \geq f(x) + \langle \nabla f(x), y - x\rangle + \frac{\sigma}{2}\|y - x\|^2,$$

which is our desired inequality.

To demonstrate the converse, suppose f satisfies the given inequality. For any x, the function $g(y) = f(y) - \frac{\sigma}{2}\|y - x\|^2$ gives us

$$\begin{aligned} g(y) = f(y) - \frac{\sigma}{2}\|y - x\|^2 &\geq f(x) + \langle \nabla f(x), y - x\rangle \\ &= g(x) + \langle \nabla g(x), y - x\rangle, \end{aligned}$$

which satisfies the convexity of g by Proposition 1.0.18, and so by Proposition 1.0.20, f is σ-strongly convex. □

Definition 1.0.23. A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is L-**Lipschitz continuous** on $\mathbb{R}^n$ if there exists an $L > 0$ such that for all x, y,

$$\|f(x) - f(y)\| \leq L\|x - y\|.$$

Example 1.0.24. Given a nonempty closed convex set $\Omega \subset \mathbb{R}^n$, the distance function $\text{dist}(x, \Omega)$ is 1-Lipschitz. To verify, let $x, y \in \mathbb{R}^n$. First suppose $\text{dist}(x, \Omega) \geq \text{dist}(y, \Omega)$.

Then

$$\text{dist}(x,\Omega) = \|x - P_\Omega(x)\| \leq \|x - y\| + \|y - P_\Omega(y)\| = \|x - y\| + \text{dist}(y,\Omega),$$

and because $\text{dist}(x,\Omega) \geq \text{dist}(y,\Omega)$ we have $|\text{dist}(x,\Omega) - \text{dist}(y,\Omega)| \leq \|x - y\|$. If instead $\text{dist}(y,\Omega) > \text{dist}(x,\Omega)$ we proceed symmetrically to obtain $|\text{dist}(y,\Omega) - \text{dist}(x,\Omega)| \leq \|y - x\|$. In either case, the distance function is 1-Lipschitz.

Definition 1.0.25. The **directional derivative** of f at the point x with respect to v is the limit (if it exists)

$$D_v f(x) = \lim_{t \to 0} \frac{f(x + tv) - f(x)}{t}.$$

We have also, for $f \in C^1$, that $D_v f(x) = \langle \nabla f(x), v \rangle$.

Proposition 1.0.26. If $f \in C^1$ is L-Lipschitz continuous, then for all x,

$$\|\nabla f(x)\| \leq L.$$

Proof. Take x and $x + tu$, where $t > 0$ and u is the unit vector in the direction of x. Then $\|f(x + tu) - f(x)\| \leq L\|(x + tu) - x\|$, and so

$$\frac{|f(x + tu) - f(x)|}{t} \leq L,$$

where taking the limit as $t \to 0$ gives us $L \geq |D_u f(x)| = |\langle \nabla f(x), u \rangle| = |\cos\theta| \|\nabla f(x)\| \|u\| = \|\nabla f(x)\|$. (This assumes $\nabla f(x) \neq 0$. If it is zero, then the result is trivial.) □

We now demonstrate an important upper bound which exists for a function whose gradient is Lipschitz continuous.

Proposition 1.0.27. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a C^1 function. If ∇f is L-Lipschitz continuous, then for all $x, y \in \mathbb{R}^n$,

$$f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2}\|y - x\|^2.$$

Proof. Using $\frac{d}{dt} f(y + t(x - y)) = \nabla f(y + t(x - y))^T (x - y)$, and the fundamental theorem of calculus, we first note that

$$\int_0^1 \nabla f(y + t(x - y)^T (x - y) dt = f(y + t(x - y))\Big|_0^1 = f(x) - f(y).$$

Now let $x, y \in \mathbb{R}^n$, and begin with

$$\big[f(y) - f(x)\big] - \big[\nabla f(x)^T(y-x)\big] = \int_0^1 \nabla f(x + t(y-x))^T(y-x)dt - \int_0^1 \nabla f(x)^T(y-x)dt$$
$$= \int_0^1 \big(\nabla f(x + t(y-x)) - \nabla f(x)\big)^T(y-x)dt.$$

Using the Cauchy Schwarz inequality and the Lipschitz continuity of ∇f, we have

$$\big[f(y) - f(x)\big] - \big[\nabla f(x)^T(y-x)\big] \leq \int_0^1 \big\|\nabla f(x + t(y-x)) - \nabla f(x)\big\| \|y-x\| dt$$
$$\leq \int_0^1 L\big\|(x + t(y-x)) - x)\big\| \|y-x\| dt$$
$$= \int_0^1 Lt\|y-x\|^2 dt.$$

From here, integration gives us

$$\big[f(y) - f(x)\big] - \big[\nabla f(x)^T(y-x)\big] \leq \frac{L}{2}\|y-x\|^2,$$

and adding $f(x)$ and $\nabla f(x)^T(y-x)$ to both sides gives us the desired inequality. □

Definition 1.0.28. A **subgradient** of $f : \mathbb{R}^n \to \mathbb{R}$ at the point c is any vector $g \in \mathbb{R}^n$ such that

$$f(x) \geq f(c) + \langle g, x - c\rangle$$

for all $x \in \mathbb{R}^n$. The set of all subgradients is called the **subdifferential** $\partial f(c)$ of f at c.

It follows from the definition that f has an absolute minimum at c is and only if $0 \in \partial f(c)$. By Proposition 1.0.18, if f is C^1, then $\partial f(c) = \{\nabla f(c)\}$ for any $c \in \mathbb{R}^n$.

Definition 1.0.29. For a convex set $\Omega \subset \mathbb{R}^n$, let $y \in \Omega$. The **normal cone** to Ω at y is the set

$$N(y, \Omega) = \{v \in \mathbb{R}^n \mid \langle v, x - y\rangle \leq 0 \text{ for all } x \in \Omega\}.$$

For $y \notin \Omega$, we define $N(y, \Omega) = \varnothing$.

Proposition 1.0.30. If f is strongly convex, then f is strictly convex. In particular, f does not have more than one absolute minimizer.

Proof. Let f be strongly convex with parameter $\sigma > 0$. Take any $x, y \in \mathbb{R}^n$ with $x \neq y$. For any $t \in (0, 1)$ we have

$$f(x + t(y-x)) \leq f(x) + t(f(y) - f(x)) - \frac{\sigma}{2}t(1-t)\|x-y\|^2.$$

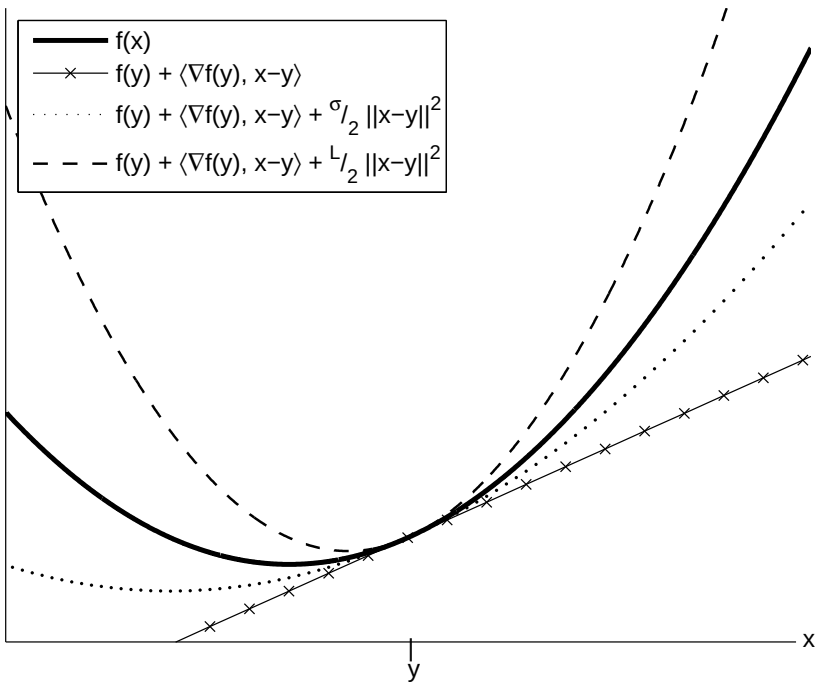


FIGURE 1.0.3: The various bounds which result from convexity and Lipschitz continuity. Any convex function is at least as great as its linearization (hashed curve). Strong convexity implies a lower bound (dotted), and a Lipschitz continuous gradient provides an upper bound (dashed).

Since $\|x - y\| > 0$, we obtain the strict inequality

$$f(x + t(y - x)) < f(x) + t(f(y) - f(x)).$$

This implies the strict convexity of f. Now suppose by contradiction that x and y are absolute minimizers of f with $x \neq y$, so $f(x) = f(y)$. Then $f(x + t(y - x)) < f(x)$. This contradicts our assumption that x is an absolute minimizer, by which we must have that $x = y$. Thus f has a unique absolute minimizer. □

Proposition 1.0.31. For a nonempty closed convex set Ω, we have $c = P_\Omega(x)$ if and only if $c \in \Omega$ and

$$\langle x - c, \omega - c \rangle \leq 0$$

for all $\omega \in \Omega$. The proof can be found in [10].

Proposition 1.0.32. For a nonempty closed convex set $\Omega \subseteq \mathbb{R}^n$ and all $x, y \in \mathbb{R}^n$,

$$\|P_\Omega(x) - P_\Omega(y)\| \leq \|x - y\|.$$

Proof. Let $x, y \in \mathbb{R}^n$ and denote $P(x)$ and $P(y)$ as their projections onto Ω. By Proposition 1.0.31, $\langle x - P(x), P(y) - P(x)\rangle \leq 0$, and so $\langle P(x) - x, P(y) - P(x)\rangle \geq 0$. Similarly we obtain $\langle P(y) - y, P(x) - P(y)\rangle \geq 0$. Taking the sum of these inequalities gives us

$$\langle P(x) - P(y), P(y) - P(x)\rangle + \langle y - x, P(y) - P(x)\rangle \geq 0,$$

from which it follows that

$$\langle P(y) - P(x), P(y) - P(x)\rangle \leq \langle y - x, P(y) - P(x)\rangle.$$

Using $\langle x, x\rangle = \|x\|^2$, we find

$$\|P(y) - P(x)\|^2 \leq \|y - x\| \cdot \|P(y) - P(x)\|$$

as a result of the Cauchy Schwarz inequality, and Lipschitz continuity follows. □

Proposition 1.0.33. If $f : \mathbb{R}^n \to \mathbb{R}$ is convex and differentiable at $\bar{x}$, then

$$\partial f(\bar{x}) = \{\nabla f(\bar{x})\}.$$

A proof of this result is given in [10].

Proposition 1.0.34. Given a nonempty closed convex set $\Omega \subset \mathbb{R}^n$ and $d(x)$ =dist(x, Ω), with $\mathbb{B}$ denoting the unit ball in $\mathbb{R}^n$, we have

$$\partial d(x) = \begin{cases} N(x, \Omega) \cap \mathbb{B} & \text{if } x \in \Omega \\ \left\{ \frac{x - P_\Omega(x)}{\|x - P_\Omega(x)\|} \right\} & \text{if } x \notin \Omega. \end{cases}$$

Proof. Let us first suppose some y lies in Ω. We will show that any subgradient of d at y lies in both the normal cone and the unit ball, and then that any element in their intersection is also a subgradient of d at y.

Let $v \in \partial d(y)$. Then v is a subgradient, so for any x, by definition we have $\langle v, x - y\rangle \leq d(x) - d(y)$. Since the distance function is 1-Lipschitz continuous, we have

$$\langle v, x - y\rangle \leq \|x - y\|.$$

Now this holds for any x, so it holds also for $x = y + v$, which gives us

$$\|v\|^2 \;=\; \langle v, v\rangle \;=\; \langle v, x - y\rangle \;\leq\; \|x - y\| \;=\; \|v\|.$$

so $\|v\|$ must be no greater than 1, thus $v \in \mathbb{B}$.

Now we also had $\langle v, x - y\rangle \leq d(x) - d(y)$. As this holds for arbitrary x; choose $x \in \Omega$. With y also in Ω by assumption, we have $d(x) = 0 = d(y)$, so $\langle v, x - y\rangle \leq 0$, which satisfies the

inclusion of v in the normal cone of Ω at y. So for $y \in \Omega$, we find $v \in N(y,\Omega) \cap \mathbb{B}$, thus $\partial d(y) \subseteq N(y,\Omega) \cap \mathbb{B}$.

Now take some $z \in N(y,\Omega) \cap \mathbb{B}$. We must show that $z \in \partial d(y)$. Let $u \in \Omega$, wherein by the inclusion of z in the normal cone, $\langle z, u-y\rangle \leq 0$. Then for any $x \in \mathbb{R}^n$,

$$\begin{aligned}\langle z, x-y\rangle &= \langle z, x-u\rangle + \langle z, u-y\rangle \\ &\leq \langle z, x-u\rangle,\end{aligned}$$

but by the Cauchy-Schwarz inequality, $\langle z, x-u\rangle \leq \|z\|\|x-u\|$, and $z \in \mathbb{B}$, so that $\langle z, x-y\rangle \leq \|x-u\|$ for $x \in \mathbb{R}^n$ and $u \in \Omega$. Let us then consider $u = P_\Omega(x)$. This gives us $\langle z, x-y\rangle \leq \|x - P_\Omega(x)\| = d(x)$, and as $d(y) = 0$, we may say that $\langle z, x-y\rangle \leq d(x) - d(y)$. By this, z satisfies the definition of a subgradient to f at y, thus $N(y,\Omega) \cap \mathbb{B} \subseteq \partial d(y)$.

It has been shown that for $y \in \Omega$, the subdifferential of f at y is the intersection of the normal cone to Ω at y and the unit ball. Let us take $y \notin \Omega$, and show that $\partial d(y) = \left\{\frac{y-P(y)}{\|y-P(y)\|}\right\}$.

To begin, take some some subgradient $v \in \partial d(y)$. For any $x \in \mathbb{R}^n$, we have that

$$\langle v, x-y\rangle \leq d(x) - d(y) = \|x - P_\Omega(x)\| - \|y - P_\Omega(y)\|.$$

Letting $z = P_\Omega(y)$ and defining $p(t) = \|t-z\|$, we have $\langle v, x-y\rangle \leq p(x) - p(y)$. This implies that v is a subgradient of p at y. However, p is differentiable with gradient $\nabla p(t) = \frac{t-z}{\|t-z\|}$ for $t \neq z$. So by Proposition 1.0.33, $\partial p(y) = \{\nabla p(y)\}$ by which we must have that $v = \frac{y-z}{\|y-z\|} = \frac{y-P_\Omega(y)}{\|y-P_\Omega(y)\|}$, as claimed.

Still considering $y \notin \Omega$ and denoting $z = P_\Omega(y)$, let us demonstrate that $v = \frac{y-z}{\|y-z\|}$ is a subgradient of d at y. Fixing some $x \in \mathbb{R}^n$, we see

$$\begin{aligned}\langle v, x-y\rangle &= \langle v, x-z\rangle + \langle v, z-y\rangle \\ &= \langle v, x-z\rangle - \|y-z\| \\ &= \langle v, x - P_\Omega(x)\rangle + \langle v, P_\Omega(x) - z\rangle - \|y-z\|.\end{aligned}$$

Now using Proposition 1.0.31, (noting that $P_\Omega(x) \in \Omega$,) we have

$$\langle v, P_\Omega(x) - z\rangle = \frac{1}{\|y-z\|}\langle y - P_\Omega(y), P_\Omega(x) - P_\Omega(y)\rangle \leq 0.$$

Then from the Cauchy Schawrz inequality, we have

$$\begin{aligned}\langle v, x-y\rangle &\leq \|v\|\|x-P_\Omega(x)\| - \|y-z\| \\ &\leq \|x-P_\Omega(x)\| - \|y-P_\Omega(y)\| \\ &= d(x) - d(y),\end{aligned}$$

which by definition implies $v = \frac{y-P_\Omega(y)}{\|y-P_\Omega(y)\|}$ is a subgradient of d at y. □

Proposition 1.0.35. For a nonempty closed and convex set $\Omega \subset \mathbb{R}^n$,

$$\nabla \text{dist}(x,\Omega)^2 = 2(x - P_\Omega(x)).$$

This follows from Proposition 1.0.34 and the chain rule for subdifferentials, a discussion of which is given in [10].

Chapter 2

Methods of Optimization

This chapter provides effective techniques for minimization of convex functions. The majorization minimization (MM) principle is introduced, and the convergence rates of some of its resulting algorithms are proven. The gradient method is shown to result from a specific application of the MM principle, which leads us to describe Nesterov's accelerated gradient method, and prove its second order convergence.

2.1 The Majorization-Minimization Principle

Originally the work of Ortega and Rheinboldt [14], the majorization-minimization principle is a simple concept with powerful applications in convex optimization. The idea has since appeared in applications ranging from statistical analysis to computational biology. The introduction is a partial summary of work by Chi, Zhou, & Lange in their paper *Distance Majorization and its Applications*, with proof of convergence following Mairal's *Optimization with First Order Surrogates.*

Introduction

Given a function $f : \mathbb{R}^n \to \mathbb{R}$ to minimize, the majorization-minimization (MM) principle uses an iterative approach to find an absolute minimizer. First, a surrogate function in two variables $g(x, y) : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is constructed to satisfy two conditions:

$$\begin{array}{lll} \textbf{Dominancy:} & g(x,\kappa) \geq f(x) & \forall x, \kappa \in \mathbb{R}^n. \\ \textbf{Tangency:} & g(\kappa,\kappa) = f(\kappa) & \forall \kappa \in \mathbb{R}^n. \end{array}$$

Choosing any x_0 and defining $x_{k+1} := \operatorname{argmin}_x g(x, x_k)$ ensure a monotonically decreasing sequence of function values, as

$$\underbrace{f(x_{k+1}) \leq \overbrace{g(x_{k+1}, x_k)}^{x_{k+1}:=\operatorname{argmin} g}}_{\text{by dominancy}} \leq \underbrace{g(x_k, x_k) = f(x_k)}_{\text{by tangency}}. \tag{2.1.1}$$

So we have $f(x_{k+1}) \leq f(x_k)$ for all k.

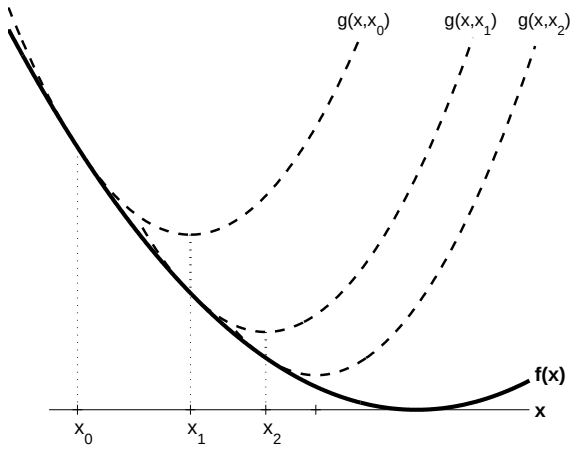


FIGURE 2.1.1: The majorization minimization principle. A surrogate function $g(x, x_i)$ is 'anchored' at $f(x_i)$. The absolute minimizer of $g(x, x_i)$ determines the next anchor point.

When minimizing the function $f(x) := \ell(x) + \operatorname{dist}(x, \Omega)^2$, where Ω is a nonempty closed convex set, it is typical to utilize the 'distance majorization' surrogate $g(x, \kappa) := \ell(x) + \|x - P_\Omega(\kappa)\|^2$, which squares the distance from x—not to its projection onto Ω—but to the projection of κ. This satisfies dominance because for any x,

$$g(x, \kappa) \;=\; \ell(x) + \|x - P_\Omega(\kappa)\|^2 \;\geq\; \ell(x) + \operatorname{dist}(x, \Omega)^2 \;=\; f(x),$$

and tangency, as

$$g(x, x) \;=\; \ell(x) + \|x - P_\Omega(x)\|^2 \;=\; \ell(x) + \operatorname{dist}(x, \Omega)^2 = f(x).$$

It should be noted that distance majorization is not the only useful surrogate. We will later see another, which follows from the upper bound induced by the Lipschitz continuity of the gradient of a convex function.

Convergence Rate

We now demonstrate that if we apply the MM principle with certain strongly convex surrogate functions, the resulting sequence of arguments is guaranteed to converge to an absolute minimizer. The proof here follows that given in [9]. It will be referenced to ensure the convergence of algorithms used for the problems in chapter 3, and is thus repeated here for convenience. For a thorough treatment of a variety of surrogate functions, the reader is encouraged to see Mairal (2013).

Lemma 2.1.1. *Let f be convex, and $g_\kappa(x) := g(x, \kappa)$ be a surrogate function of f (satisfying dominance and tangency, and 'anchored' at κ) with an approximation error function $h_\kappa := g_\kappa - f$ such that ∇h_κ is L-Lipschitz continuous and $\nabla h_\kappa(\kappa) = 0$. Let x be any element in the $\mathbb{R}^n$ and x_m be an absolute minimizer of $g_\kappa(x)$. Then*

$\{a\}$ $$|h_\kappa(x)| \leq \tfrac{L}{2}\|x - \kappa\|^2,$$

and if g is σ-strongly convex,

(b) $$f(x_m) + \tfrac{\sigma}{2}\|x - x_m\|^2 \leq f(x) + \tfrac{L}{2}\|x - \kappa\|^2.$$

Proof (a) Let f and g_κ satisfy the given conditions, and suppose x_m is an absolute minimum of g_κ. By Proposition 1.0.27, we have $h_\kappa(x) \leq h_\kappa(y) + \nabla h_\kappa(y)^T(x - y) + \frac{L}{2}\|x - y\|^2$ for all x, y. Letting $y = \kappa$, we have $\nabla h_\kappa(\kappa) = 0$ by assumption, and $h_\kappa(\kappa) = g(\kappa, \kappa) - f(\kappa) = 0$ by tangency. Substitution of these terms gives us $h_\kappa(x) \leq \frac{L}{2}\|x - \kappa\|^2$. By dominance, $h_\kappa(x) = g_\kappa(x) - f(x) \geq 0$, so $h_\kappa(x) = |h_\kappa(x)|$, and the result follows. □

Proof (b) Let f and g_κ satisfy the given conditions, and suppose further that g_κ is strongly convex. By Proposition 1.0.22 we have $g_\kappa(x) \geq g_\kappa(y) + \langle \nabla g_\kappa(y), x - y\rangle + \frac{\sigma}{2}\|x - y\|^2$, for all x, y. Let x_m be an absolute minimum of g_κ, so $\nabla g_\kappa(x_m) = 0$, and then choosing $y = x_m$ gives us $g_\kappa(x) \geq g_\kappa(x_m) + \frac{\sigma}{2}\|x - x_m\|^2$. Then combining this with the dominance property we have,

$$f(x_m) + \frac{\sigma}{2}\|x - x_m\|^2 \leq g_\kappa(x_m) + \frac{\sigma}{2}\|x - x_m\|^2 \leq g_\kappa(x).$$

Noting that $g_\kappa(x) = f(x) + h_\kappa(x)$ and utilizing part (a), we find

$$f(x_m) + \frac{\sigma}{2}\|x - x_m\|^2 \leq f(x) + \frac{L}{2}\|x - \kappa\|^2,$$

which is the desired inequality. □

Theorem 2.1.1. Let f be a convex function with an absolute minimum at x^*, and let $g_\kappa(x) := g(x, \kappa)$ be a surrogate of f (satisfying dominance and tangency, and 'anchored' at κ.) Suppose that the error function $h_\kappa := g_\kappa - f$ has an L-Lipschitz continuous gradient, $\nabla h_\kappa(\kappa) = 0$, and g_κ is σ-strongly convex with $\sigma \geq L$. For $\{x_k\}$ the sequence of iterations

generated by the MM algorithm,

$$f(x_k) - f(x^*) \leq \frac{L\|x_0 - x^*\|^2}{2k}.$$

Proof. Let x_k an absolute minimum of $g(x, x_{k-1})$. Then from part(b) of Lemma 2.1.1, we have for any x,

$$f(x_k) + \frac{\sigma}{2}\|x - x_k\|^2 \leq f(x) + \frac{L}{2}\|x - x_{k-1}\|^2.$$

Let consider the case where x is x^*, and rearrange the expression to find

$$f(x_k) - f(x^*) \leq \frac{L}{2}\|x^* - x_{k-1}\|^2 - \frac{\sigma}{2}\|x^* - x_k\|^2.$$

Because $\sigma \geq L$, this allows

$$f(x_k) - f(x^*) \leq \frac{L}{2}\Big(\|x^* - x_{k-1}\|^2 - \|x^* - x_k\|^2\Big).$$

We may take the sum of this inequality over j iterations to find useful cancellations:

$$\begin{array}{rcl}
f(x_1) - f(x^*) & \leq & \frac{L}{2}\Big(\|x^* - x_0\|^2 - \|x^* - x_1\|^2\Big) \\
+f(x_2) - f(x^*) & \leq & +\frac{L}{2}\Big(\|x^* - x_1\|^2 - \|x^* - x_2\|^2\Big) \\
\vdots & \vdots & \vdots \\
+f(x_{j-1}) - f(x^*) & \leq & +\frac{L}{2}\Big(\|x^* - x_{j-2}\|^2 - \|x^* - x_{j-1}\|^2\Big) \\
+f(x_j) - f(x^*) & \leq & +\frac{L}{2}\Big(\|x^* - x_{j-1}\|^2 - \|x^* - x_j\|^2\Big) \\
\hline
\displaystyle\sum_{i=1}^{j} f(x_i) - f(x^*) & \leq & \frac{L}{2}\big(\|x_0 - x^*\|^2 - \|x^* - x_j\|^2\big)
\end{array}$$

where each $f(x_i)$ in the sum is no less than $f(x_j)$ because of the monotonicity of the MM-updates, so that $\sum f(x_i) - f(x^*) \geq j\big(f(x_j) - f(x^*)\big)$. Dropping the $\|x^* - x_j\|^2$ term gives us

$$f(x_j) - f(x^*) \leq \frac{L}{2j}\|x_0 - x^*\|^2,$$

which is the claimed convergence. □

2.2 Lipschitz Based Surrogates and the Gradient Method

We were previously introduced to the distance majorization surrogate, where $g(x, \kappa) := \ell(x) + \frac{1}{2}\|x - P_\Omega(\kappa)\|^2$ acts as a surrogate to $f(x) = \ell(x) + \frac{1}{2}\text{dist}(x, \Omega)^2$. In this section we explore another surrogate (as described in [9]) which utilizes the Lipschitz continuity based

upper bound given in Proposition 1.0.27. For a C^1 function with L-Lipschitz continuous gradient, define the surrogate

$$g_\kappa(x) := f(\kappa) + \langle \nabla f(\kappa), x - \kappa \rangle + \frac{L}{2}\|x - \kappa\|^2.$$

Note that this is precisely the Lipschitz continuity based upper bound given in Proposition 1.0.27, so g_κ satisfies dominance, and verifying tangency is trivial by evaluating $g(\kappa, \kappa)$.

To find a minimal argument of $g_\kappa(x)$ for the update of the MM algorithm, we compute the gradient and find where it is zero. With the gradient given by

$$\nabla g_\kappa(x) = \nabla f(\kappa) + L(x - \kappa),$$

we can solve for $\nabla g_\kappa(x) = 0$ at

$$x = \kappa - \frac{1}{L}\nabla f(\kappa),$$

so that using this to define our next iteration gives us

$$x_{k+1} := x_k - \frac{1}{L}\nabla f(x_k).$$

Note that this is exactly the update used in the gradient method, with step size $\frac{1}{L}$.

A sophisticated proof that if ∇f is L-Lispschitz then so too is ∇h is given in [9](Lemmas B.2,B.9), which, coupled with the strong convexity of g, gives us the convergence rate given in theorem 2.1.1. This is however the same result as from the conventional (and more approachable) proof of gradient method convergence, which we find below.

Theorem 2.2.1. Let $f \in C^1$ be a convex function with an absolute minimum at x^*, and suppose ∇f is L-Lipschitz continuous. The gradient descent update $x_{k+1} := x_k - \alpha \nabla f(x_k)$, with $\alpha \leq \frac{1}{L}$ converges to x^* with order $\mathcal{O}(1/k)$. Further,

$$f(x_k) - f(x^*) \leq \frac{\|x_0 - x^*\|^2}{2\alpha k}.$$

Proof. Let us start by utilizing Proposition 1.0.27 to guarantee the upper bound on $f(x_{k+1})$,

$$f(x_{k+1}) \leq f(x_k) + \langle \nabla f(x_k), x_{k+1} - x_k \rangle + \frac{L}{2}\|x_{k+1} - x_k\|^2.$$

Now because $x_{k+1} := x_k - \alpha \nabla f(x_k)$, we have the difference $x_{k+1} - x_k = -\alpha \nabla f(x_k)$, and

substitution gives us

$$\begin{aligned} f(x_{k+1}) &\leq f(x_k) - \alpha\|\nabla f(x_k)\|^2 + \frac{\alpha^2 L}{2}\|\nabla f(x_k)\|^2 \\ &= f(x_k) + \left(\frac{\alpha^2 L}{2} - \alpha\right)\|\nabla f(x_k)\|^2. \end{aligned}$$

Taking a moment to confirm that, with $\alpha \leq 1/L$, we have $\frac{\alpha^2 L}{2} - \alpha \leq -\alpha/2$ this gives us

$$f(x_{k+1}) \leq f(x_k) - \frac{\alpha}{2}\|\nabla f(x_k)\|^2.$$

From here, the first order convexity condition ensures that we have $f(x_k) \leq f(x^*) - \langle \nabla f(x_k), x^* - x_k \rangle$, which provides

$$f(x_{k+1}) \leq f(x^*) + \langle \nabla f(x_k), x_k - x^* \rangle - \frac{\alpha}{2}\|\nabla f(x_k)\|^2. \tag{2.2.1}$$

If we express $\alpha\nabla f(x_k)$ as $(x_k - x^*) - (x_k - x^* - \alpha\nabla f(x_k))$, it can be shown that

$$\|\alpha\nabla f(x_k)\|^2 = -\|x_k - x^*\|^2 - 2\langle x_k - x^*, -\alpha\nabla f(x_k)\rangle + \|x_{k+1} - x^*\|^2,$$

and multiplying by $\frac{-1}{2\alpha}$ gives us

$$-\frac{\alpha}{2}\|\nabla f(x_k)\|^2 = \frac{1}{2\alpha}\big(\|x_k - x^*\|^2 - \|x_{k+1} - x^*\|^2\big) - \langle x_k - x^*, \nabla f(x_k)\rangle.$$

Using this identity in 2.2.1 gives us

$$f(x_{k+1}) \leq f(x^*) + \frac{1}{2\alpha}\big(\|x_k - x^*\|^2 - \|x_{k+1} - x^*\|^2\big),$$

which brings us to our last step. If we sum the above inequality over some j consecutive k-values, starting with k_0, many terms cancel, leaving us with

$$\sum_{i=1}^{j} f(x_i) \leq jf(x^*) + \frac{1}{2\alpha}\big(\|x_0 - x^*\|^2 - \|x_j - x^*\|^2\big).$$

The monotonicity of the MM algorithm ensures that $f(x_i) \geq f(x_j)$ for all $i < j$, so that $jf(x_k) \leq \sum f(x_i)$. In this way we have

$$f(x_j) - f(x^*) \leq \frac{\|x_0 - x^*\|^2}{2\alpha j}$$

which concludes the proof. □

2.3 Nesterov's Accelerated Gradient Method

In this section we introduce a famous technique known as Nesterov's accelerated gradient method (see [11], [12],[13]) which has a convergence rate of $\mathcal{O}(1/k^2)$. This proof was given by Beck & Teboulle (2009) in their development of *FISTA* methods, and written here with the help of Bubeck (2013). Starting with the sequence:

$$\lambda_0 = 0, \quad \lambda_k = \frac{1+\sqrt{1+4\lambda_{k-1}^2}}{2}, \quad \gamma_k = \frac{1-\lambda_k}{\lambda_{k+1}},$$

and choosing an arbitrary $y_0 = x_0$, let

$$y_{k+1} := x_k - \frac{1}{L}\nabla f(x_k), \quad x_{k+1} := (1-\gamma_k)y_{k+1} + \gamma_k y_k.$$

Theorem 2.3.1. If $f \in C^1$ is convex and ∇f is L-Lipschitz continuous, with x^* a minimizer of f, then for the sequence $x_0, x_1, ..., x_k$ defined above,

$$f(y_k) - f(x^*) \leq \frac{2L\|x_1 - x^*\|^2}{k^2}.$$

Proof. For all $x, z \in \mathbb{R}^n$, the convexity of f ensures $f(z) \geq f(x) + \nabla f(x)^T(z-x)$. Multiplying by -1 and adding $f(x - \nabla f(x)/L)$ allows

$$f\left(x - \frac{\nabla f(x)}{L}\right) - f(z) \leq f\left(x - \frac{\nabla f(x)}{L}\right) - f(x) + \nabla f(x)^T(x-z). \tag{2.3.1}$$

Given that ∇f is L-Lipschitz continuous, by Proposition 1.0.27 we express the upper bound

$$\begin{aligned} f(x - \nabla f(x)/L) &\leq f(x) + \nabla f(x)^T((x - \nabla f(x)/L) - x) + \frac{L}{2}\|(x - \nabla f(x)/L) - x\|^2 \\ &= f(x) - \frac{1}{2L}\|\nabla f(x)\|^2, \end{aligned}$$

so that 2.3.1 reduces to

$$f\left(x - \frac{\nabla f(x)}{L}\right) - f(z) \leq -\frac{1}{2L}\|\nabla f(x)\|^2 + \nabla f(x)^T(x-z).$$

As this holds for any $x, z \in \mathbb{R}^n$; let us consider $x = x_k$. We also note $y_{k+1} := x_k - \nabla f(x_k)/L$, and thus $\nabla f(x_k) = -L(y_{k+1} - x_k)$. This gives us

$$\begin{aligned} f(y_{k+1}) - f(z) &\leq -\frac{1}{2L}\|\nabla f(x_k)\|^2 + \nabla f(x_k)^T(x_k - z) \\ &= \frac{-L}{2}\|y_{k+1} - x_k\|^2 - L(y_{k+1} - x_k)^T(x_k - z). \end{aligned}$$

We now note that this inequality is true for both $z = y_k$ and $z = x^*$, from which we obtain

$$f(y_{k+1}) - f(y_k) \leq \frac{-L}{2}\big\|y_{k+1} - x_k\big\|^2 - L(y_{k+1} - x_k)^T(x_k - y_k) \tag{2.3.2}$$

$$f(y_{k+1}) - f(x^*) \leq \frac{-L}{2}\big\|y_{k+1} - x_k\big\|^2 - L(y_{k+1} - x_k)^T(x_k - x^*). \tag{2.3.3}$$

We will multiply 2.3.2 by $\lambda_k - 1$ and add it to 2.3.3 so that, on the left hand side we have,

$$\begin{aligned}(\lambda_k - 1)\big[f(y_{k+1}) - f(y_k)\big] + f(y_{k+1}) - f(x^*)& \\ = \lambda_k\big[f(y_{k+1}) - f(x^*)\big] - (\lambda_k - 1)\big[f(y_k) - f(x^*)\big],&\end{aligned}$$

which is less than or equal to the right hand side:

$$\begin{aligned}(\lambda_k - 1)\left[\frac{-L}{2}\big\|y_{k+1} - x_k\big\|^2 - L(y_{k+1} - x_k)^T(x_k - y_k)\right] + \frac{-L}{2}\big\|y_{k+1} - x_k\big\|^2 + L(y_{k+1} - x_k)^T(x_k - x^*)& \\ = \lambda_k\frac{-L}{2}\|y_{k+1} - x_k\|^2 - L(y_{k+1} - x_k)^T(\lambda_k x_k - (\lambda_k - 1)y_k - x^*).&\end{aligned}$$

Using δ_k to denote $f(y_k) - f(x^*)$, our sum of inequalities is

$$\lambda_k\delta_{k+1} - (\lambda_k - 1)\delta_k \leq \lambda_k\frac{-L}{2}\|y_{k+1} - x_k\|^2 - L(y_{k+1} - x_k)^T(\lambda_k x_k - (\lambda_k - 1)y_k - x^*). \tag{2.3.4}$$

Elementary algebra shows that $\lambda_k^2 - \lambda_k = \lambda_{k-1}^2$, so that multiplying 2.3.4 by λ_k (and factoring out $-L/2$) gives us

$$\lambda_k^2\delta_{k+1} - \lambda_{k-1}^2\delta_k \leq \frac{-L}{2}\Big(\|\lambda_k(y_{k+1} - x_k)\|^2 + 2\lambda_k(y_{k+1} - x_k)^T(\lambda_k x_k - (\lambda_k - 1)y_k - x^*)\Big). \tag{2.3.5}$$

Following directly from the expansion of $\|\lambda_k(y_{k+1} - x_k) + \lambda_k x_k - (\lambda_k - 1)y_k - x^*\|^2$ we find that

$$\begin{aligned}\|\lambda_k(y_{k+1} - x_k)\|^2 + 2\lambda_k(y_{k+1} - x_k)^T(\lambda_k x_k - (\lambda_k - 1)y_k - x^*)& \\ = \|\lambda_k y_{k+1} - (\lambda_k - 1)y_k - x^*\|^2 - \|\lambda_k x_k - (\lambda_k - 1)y_k - x^*\|^2,&\end{aligned}$$

which we substitute into 2.3.5 to obtain

$$\lambda_k^2\delta_{k+1} - \lambda_{k-1}^2\delta_k \leq \frac{-L}{2}\Big(\|\lambda_k y_{k+1} - (\lambda_k - 1)y_k - x^*\|^2 - \|\lambda_k x_k - (\lambda_k - 1)y_k - x^*\|^2\Big). \tag{2.3.6}$$

It follows from the definition of x_{k+1} that

$$x_{k+1} = y_{k+1} + \gamma_k(y_k - y_{k+1}),$$

and multiplying this by λ_{k+1}, and using by its definition that $\gamma_k := \frac{1-\lambda_k}{\lambda_{k+1}}$, we find

$$\lambda_{k+1}x_{k+1} = (\lambda_{k+1} - 1)y_{k+1} + (1 - \lambda_k)y_k + \lambda_k y_{k+1}.$$

Utilizing the equivalent form

$$\lambda_k y_{k+1} - (\lambda_k - 1)y_k = \lambda_{k+1}x_{k+1} - (\lambda_{k+1} - 1)y_{k+1},$$

we may restate 2.3.6 as

$$\lambda_k^2\delta_{k+1} - \lambda_{k-1}^2\delta_k = \frac{-L}{2}\Big(\|\lambda_{k+1}x_{k+1} - (\lambda_{k+1} - 1)y_{k+1} - x^*\|^2 - \|\lambda_k x_k - (\lambda_k - 1)y_k - x^*\|^2\Big).$$

For the sake of notation, express $\lambda_k x_k - (\lambda_k - 1)y_k - x* = u_k$, so that we have

$$\lambda_k^2\delta_{k+1} - \lambda_{k-1}^2\delta_k \leq \frac{L}{2}(\|u_k\|^2 - \|u_{k+1}\|^2). \tag{2.3.7}$$

Now summing 2.3.7 over $k = 1$ to $k = t - 1$ allows cancellation of terms, giving us

$$\lambda_{t-1}^2\delta_t - \lambda_0^2\delta_1 \leq \frac{L}{2}(\|u_1\|^2 - \|u_t\|^2).$$

If we note that $\lambda_0 = 0$ by definition, $\lambda_1 = 1$ by simple computation, and $\|u_t\|^2 \geq 0$ we have

$$\begin{aligned}
\lambda_{t-1}^2\delta_t &\leq \frac{L}{2}\|u_1\|^2 \\
&= \frac{L}{2}\|\lambda_1 x_1 - (\lambda_1 - 1)y_1 - x^*\|^2 \\
&= \frac{L}{2}\|x_1 - x^*\|^2 \\
\delta_t &\leq \frac{L}{2\lambda_{t-1}^2}\|x_1 - x^*\|^2.
\end{aligned}$$

From here we note that for all k, $\lambda_k \geq \frac{k+1}{2}$ follows by simple induction, so $\frac{1}{\lambda_{t-1}} \leq \frac{2}{t}$, and thus

$$f(y_t) - f(x^*) = \delta_t \leq \frac{2L}{t^2}\|x_1 - x^*\|^2.$$

which concludes the proof of second order convergence. □

Chapter 3

Optimization Problems

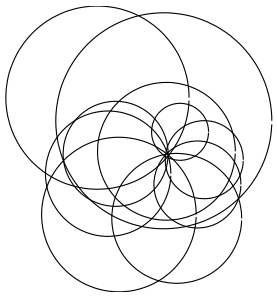
In this chapter we present four problems in convex optimization: the feasible point, closest point, support vector machine, and smallest intersecting ball problems. For each, the MM principle is shown to produce algorithms which lead to their optimal solutions. The derivation and approach for the feasible and closest point problems and the support vector machine problem is given in Chi, Zhou, & Lange (2014), and we apply the results from Mairal (2013) to determine their convergence rates. The log-exponential smoothing approach to the smallest intersecting ball problem is given in An et al. (2015), and to the author's best knowledge, the 2 additional methods of solving the smallest intersecting ball problem are original.

3.1 Feasible Point

In a constrained optimization problem, the only elements which we allow to be an optimal solution are those which lie in the constraint set. For this reason, we refer to the points which satisfy all constraints as *feasible points*. The feasible point problem asks to locate such a point in an intersection of sets in $\mathbb{R}^n$. It is in essence solely a constraint problem, because there is no corresponding loss function to minimize. For some collection of nonempty and closed sets $\{C_i \subset \mathbb{R}^n \mid i = 1, ..., m\}$, the feasible point problem can be phrased, using the penalty method, as the minimization problem:

$$\text{minimize: } f(x) = \frac{1}{2} \sum_{i}^{m} \text{dist}(x, C_i)^2. \tag{3.1.1}$$

This function is zero if and only if the distance to each set C_i is zero, which occurs if and only if $x \in C_i$ for all $i = 1, ..., m$. The square term allows differentiation, and the $\frac{1}{2}$ is a convention for simple notation in differentiation.



4	4	2	1	1	8	1	4	5	4	2
1	6	0	1	1	7	6	6	8	4	7
1	1	6	0	0	5	9	1	5	1	11
7	9	4	0	1	1	1	3	3	0	10
2	1	4	4	1	2	2	1	2	2	9
6	2	6	6	1	0	3	7	4	3	11
9	7	0	7	3	9	0	8	4	6	11
4	4	3	5	3	7	6	3	3	9	8
6	7	7	1	2	5	4	6	5	9	9
7	3	6	6	2	3	9	2	7	4	12

FIGURE 3.1.1: In higher dimensions, finding a feasible point is non-trivial.
(left) 10 Balls in $\mathbb{R}^2$ with non empty intersection. The feasible point problem asks for a point which lies in each and every ball.
(right) 10 Balls in $\mathbb{R}^{10}$. Each row of data contains the coordinates of their centers, with radii in the last column.

3.1.1 Distance Majorization

If the sets are closed and convex, we may, as described in [5], employ the majorization minimization technique with the conventional distance majorization surrogate

$$g(x, x_k) = \frac{1}{2} \sum_{i=1}^{m} \|x - P_{C_i}(x_k)\|^2.$$

To find each update x_{k+1}, we compute the gradient of g and find for which values $\nabla g(x, x_k) = 0$. In this case the gradient is explicitly given for some fixed x_k as

$$\nabla g(x, x_k) = \sum_{i=1}^{m} \Big(x - P_{C_i}(x_k) \Big).$$

Then $\nabla g(x, x_k) = 0$ when

$$x = \frac{1}{m} \sum_{i=1}^{m} P_{C_i}(x_k),$$

which gives us each update

$$x_{k+1} := \frac{1}{m} \sum_{i=1}^{m} P_{C_i}(x_k)$$

as the average of the projections onto each set. This is known as the simultaneous projection algorithm, originally developed by Gianfranco Cimmino in [6] in the 1930's.

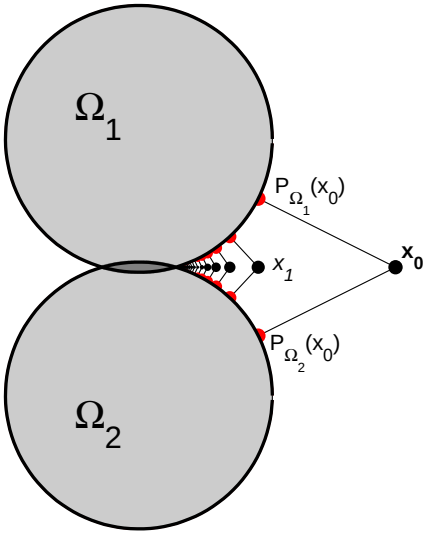


FIGURE 3.1.2: Using distance majorization to find a feasible point in $\Omega_1 \cap \Omega_2$. Each iteration is the average of projections of the last; this update scheme is known as the simultaneous projection algorithm.

We now demonstrate that the above feasible point algorithm converges with order $\mathcal{O}(1/k)$.

Proposition 3.1.1. Let $f(x) = \frac{1}{2}\sum_i^m \|x - P_{C_i}(x)\|^2$ and $g(x, x_k) = \frac{1}{2}\sum_i^m \|x - P_{C_i}(x_k)\|^2$. Let $x_0 \in \mathbb{R}^n$ and for $j = 0, 1, 2, ...$ define x_{j+1} as an optimal solution of $g(x, x_j)$. Then

$$f(x_j) - f(x*) \leq \frac{m\|x_0 - x^*\|^2}{2j}.$$

Proof. We will prove (a) the gradient of the error $h_\kappa := g(x, \kappa) - f(x)$ is m-Lipschitz continuous with $\nabla h_\kappa(\kappa) = 0$, and (b) that g is m-strongly convex, which will satisfy the conditions of theorem 2.1.1.
(a) Let us compute the difference $h_\kappa(x) = g(x, \kappa) - f(x)$, where κ is fixed,

$$h_\kappa(x) := g(x, \kappa) - f(x) = \frac{1}{2}\sum_i^m \|x - P_{\Omega_i}(\kappa)\|^2 - \frac{1}{2}\sum_i^m \|x - P_{\Omega_i}(x)\|^2$$

$$= \frac{1}{2}\sum_i^m \big(\|x - P_{\Omega_i}(\kappa)\|^2 - \|x - P_{\Omega_i}(x)\|^2\big).$$

Then its gradient is

$$\begin{aligned}\nabla h_\kappa(x) &= \sum_i^m \big((x - P_{\Omega_i}(\kappa)) - (x - P_{\Omega_i}(x))\big) \\ &= \sum_i^m \big(P_{\Omega_i}(x) - P_{\Omega_i}(\kappa)\big).\end{aligned}$$

It is clear here that $\nabla h_\kappa(\kappa) = 0$. Then consider the norm of the difference

$$\begin{aligned}\|\nabla h_\kappa(x) - \nabla h_\kappa(y)\| &= \big\|\sum_i^m \big(P_{\Omega_i}(x) - P_{\Omega_i}(\kappa)\big) - \sum_i^m \big(P_{\Omega_i}(y) - P_{\Omega_i}(\kappa)\big)\big\| \\ &= \big\|\sum_i^m \big(P_{\Omega_i}(x) - P_{\Omega_i}(y)\big)\big\| \\ &\leq \sum_i^m \|P_{\Omega_i}(x) - P_{\Omega_i}(y)\|.\end{aligned}$$

Using Proposition 1.0.32, each term in the sum is less than or equal to $\|x - y\|$. This gives us

$$\|\nabla h_\kappa(x) - \nabla h_\kappa(y)\| \leq m\|x - y\|,$$

which demonstrates that ∇h is m-Lipschitz.

(b) To see that $g(x, \kappa)$ is m-strongly convex, we use

$$\begin{aligned}g(x,\kappa) - \frac{m}{2}\|x\|^2 &= \frac{1}{2}\sum_i^m \|x - P_{\mathcal{C}_i}(\kappa)\|^2 - \frac{m}{2}\|x\|^2 \\ &= \frac{1}{2}\sum_i^m \Big(\|(x - P_{\mathcal{C}_i}(\kappa))\|^2 - \|x\|^2\Big) \\ &= \frac{1}{2}\sum_i^m \Big(\|x\|^2 + \|P_{\mathcal{C}_i}(\kappa)\|^2 - 2\langle x, P_{\mathcal{C}_i}(\kappa)\rangle - \|x\|^2\Big) \\ &= \frac{1}{2}\sum_i^m \Big(\|P_{\mathcal{C}_i}(\kappa)\|^2 - 2\langle x, y - P_{\mathcal{C}_i}(\kappa)\rangle\Big),\end{aligned}$$

which is a sum of convex functions, and is thus convex, so by Proposition 1.0.21, $g(x, \kappa)$ is m-strongly convex.

This is sufficient by Theorem 2.1.1 to justify the given convergence. □

3.1.2 Lipschitz Continuous Gradient Surrogate

To find a feasible point by minimizing equation 3.1.1, we might also consider the surrogate given in section 2.2, with $g(x, \kappa)$ defined as the upper bound on f induced by the Lipschitz constant of its gradient. Let us first see that the gradient of the feasible point loss function 3.1.1 is indeed Lipschitz.

Proposition 3.1.2. For a collection of m closed convex sets, and the function

$$f(x) = \frac{1}{2}\sum_{i=1}^{m} \|x - P_{\Omega_i}(x)\|^2,$$

the gradient ∇f is $2m$-Lipschitz.

Proof. It is straightforward to compute the gradient of f to be

$$\nabla f(x) = mx - \sum_{i}^{m} P_{\Omega_i}(x),$$

and then for any x, y we can use the triangle inequality and the Lipschitz continuity of the projection (Proposition 1.0.32) to find

$$\begin{aligned}
\|\nabla f(x) - \nabla f(y)\| &= \|m(x-y) + \sum_{i}^{m}\big(P_{\Omega_i}(y) - P_{\Omega_i}(x)\big)\| \\
&\leq m\|x-y\| + \sum_{i}^{m}\|P_{\Omega_i}(y) - P_{\Omega_i}(x)\| \\
&\leq m\|x-y\| + \sum_{i}^{m}\|x-y\| \\
&= 2m\|x-y\|,
\end{aligned}$$

which demonstrates that ∇f is $2m$-Lipschitz. □

If we then define the surrogate

$$g(x, z) := f(z) + \langle \nabla f(z), x - z\rangle + \frac{L}{2}\|x - z\|^2$$

as discussed in section 2.2, minimization leads to the gradient method update. Using theorem 2.2.1 with $L = 2m$ the convergence error is given by

$$f(x_k) - f(x^*) \leq \frac{m\|x_0 - x^*\|^2}{k}.$$

This is twice that of the distance majorization approach, but because ∇f is $2m$-Lipschitz

continuous, it is eligible for Nesterov acceleration. By Theorem 2.3.1 then, we may obtain

$$f(y_k) - f(x^*) \leq \frac{4m\|x_1 - x^*\|^2}{k^2}$$

by using the accelerated gradient method. The implementation and comparison of these methods is given in section 4.2.

3.2 The Closest Point Problem

The closest point problem is a modification of the feasible point problem. Given a collection of nonempty closed convex sets $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_m$ and a point $y \in \mathbb{R}^n$, we seek not simply a point in the intersection $\cap \mathcal{C}_i$, but the closest such point to y. This introduces a simple loss function to be minimized, with constraint, as follows:

$$\min_{x \in \mathbb{R}^n} \; f(x) = \|x - y\|^2 \qquad \text{such that } x \in \bigcap_{i=1}^{m} \mathcal{C}_i.$$

Using the penalty method, we instead minimize

$$f(x) = \frac{1}{2}\|x - y\|^2 + \frac{\mu}{2}\sum_{i}^{m} \|x - P_{\mathcal{C}_i}(x)\|^2,$$

with $\mu > 0$ a variable weight which allows us to scale penalization of the constraint violation. We again use distance majorization to construct the surrogate

$$g(x, x_k) = \frac{1}{2}\|x - y\|^2 + \frac{\mu}{2}\sum_{i}^{m} \|x - P_{\mathcal{C}_i}(x_k)\|^2,$$

whose gradient is given by

$$\begin{aligned} \nabla g(x, x_k) &= x - y + \mu \sum_{i}^{m} x - P_{\mathcal{C}_i}(x_k) \\ &= x(1 + \mu m) - y - \mu \sum_{i}^{m} P_{\mathcal{C}_i}(x_k). \end{aligned}$$

Where solving for $\nabla g(x, x_k) = 0$ gives us the update

$$x_{k+1} := \frac{y}{1 + \mu m} + \frac{\mu}{1 + \mu m} \sum_{i}^{m} P_{\mathcal{C}_i}(x_k).$$

Note that if we denote $t = \frac{m\mu}{1+m\mu}$, which lies in the interval $(0,1)$, we may express this update as

$$x_{k+1} = y + t\Big(\frac{1}{m}\sum_{i}^{m} P_{\mathcal{C}_i}(x_k) - y\Big).$$

The update is unchanged, but this demonstrates that at each iteration we take as our next iterate a point which lies on the chord between y and the average of projections of x_k. If we start with some small μ (which corresponds to a small t) our updates will fall near y, and as μ increases, t approaches 1, the updates are weighted toward the average of projections, and the algorithm tends towards the feasible point algorithm. It is unknown what method of increasing μ values optimizes the algorithm.
In computing the error function $h_\kappa := g(x,\kappa) - f(x)$, the $\|x-y\|^2$ terms cancel, to give

$$h_\kappa(x) = \frac{\mu}{2}\sum_{i}^{m}\big(\|x - P_{\mathcal{C}_i}(\kappa)\|^2 - \|x - P_{\mathcal{C}_i}(x)\|^2\big),$$

which is the same function h in the feasible point problem except scaled by μ, so it is simple to verify that ∇h is μm-Lipschitz. To see that g is $(1+\mu m)$-strongly convex for a fixed point y, we take

$$\begin{aligned}
g(x,\kappa) - \frac{1+\mu m}{2}\|x-y\|^2 &= \frac{1}{2}\|x-y\|^2 + \frac{\mu}{2}\sum_{i}^{m}\|x - P_{\mathcal{C}_i}(\kappa)\|^2 - \frac{1}{2}\|x-y\|^2 - \frac{\mu m}{2}\|x-y\|^2 \\
&= \frac{\mu}{2}\sum_{i}^{m}\big(\|x - P_{\mathcal{C}_i}(\kappa)\|^2 - \|x-y\|^2\big) \\
&= \frac{\mu}{2}\sum_{i}^{m}\big(\|x - y + y + P_{\mathcal{C}_i}(\kappa)\|^2 - \|x-y\|^2\big) \\
&= \frac{\mu}{2}\sum_{i}^{m}\big(\|x-y\|^2 + 2\langle x-y, y + P_{\mathcal{C}_i}(\kappa)\rangle + \|y - P_{\mathcal{C}_i}(\kappa)\|^2 - \|x-y\|^2\big) \\
&= \frac{\mu}{2}\sum_{i}^{m}\big(2\langle x-y, y + P_{\mathcal{C}_i}(\kappa)\rangle + \|y - P_{\mathcal{C}_i}(\kappa)\|^2\big),
\end{aligned}$$

where $\langle x-y, y + P_{\mathcal{C}_i}(\kappa)\rangle + \|y - P_{\mathcal{C}_i}(\kappa)\|^2$ is convex for any i, so by Proposition 1.0.21 we have $g(x,\kappa)$ is $1+\mu m$ strongly convex, and and by theorem 2.1.1 we obtain for a given μ penalty,

$$f(x_k) - f(x^*) \leq \frac{m\mu\|x_0 - x^*\|^2}{2k}.$$

Unfortunately, the closest point algorithm requires that μ be very large to sufficiently penalize distance from the constraint.

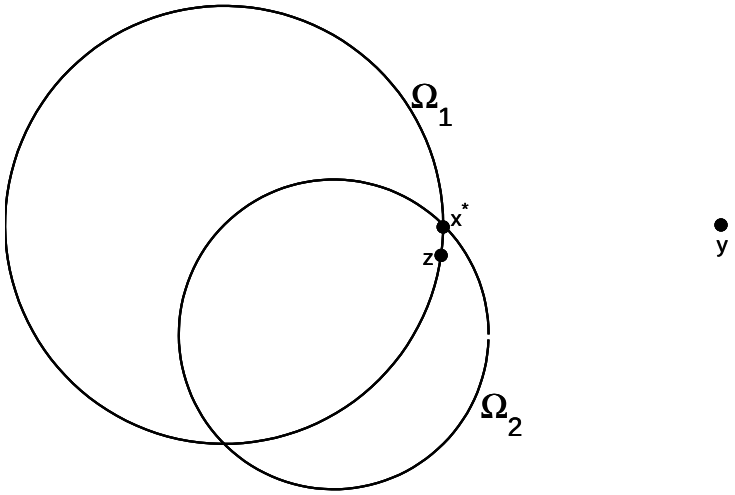


FIGURE 3.2.1: The closest point problem in $\mathbb{R}^2$. The optimal solution x^* minimizes distance to y while lying in the intersection of Ω_1 and Ω_2. A feasible point algorithm returns z, which satisfies the set constraint but does not minimize distance.

3.3 The Support Vector Machine Problem

Support vector machines are one method of solving binary classification problems, which seek to determine which of two classes some element belongs to, based on some known collection of observations, or training data. The support vector is a hyperplane, or 'decision boundary,' which separates two sets of data with the largest margin. Each datapoint $x_i \in \mathbb{R}^n$ is associated with a $y_i \in \{-1, 1\}$ to designate which data class it lies in. Any hyperplane can be described by its normal vector ω and a point on the plane p, by all $x \in \mathbb{R}^n$ such that $\omega \cdot x = \omega \cdot p$. Thus a hyperplane is expressed as the set of all x such that $\langle \omega, x \rangle + b = 0$, where $b = -n \cdot p$. A hyperplane partitions $\mathbb{R}^n$ into two halfspaces, where for any x, the halfspace in which it lies is determined by whether $\langle \omega, x \rangle + b$ is greater or less than zero. To see how the MM principle allows us to construct an optimal hyperplane, let us first derive the optimization problem.

For each data type ($y_i = 1$ or $y_i = -1$) we can construct a parallel hyperplane H such that

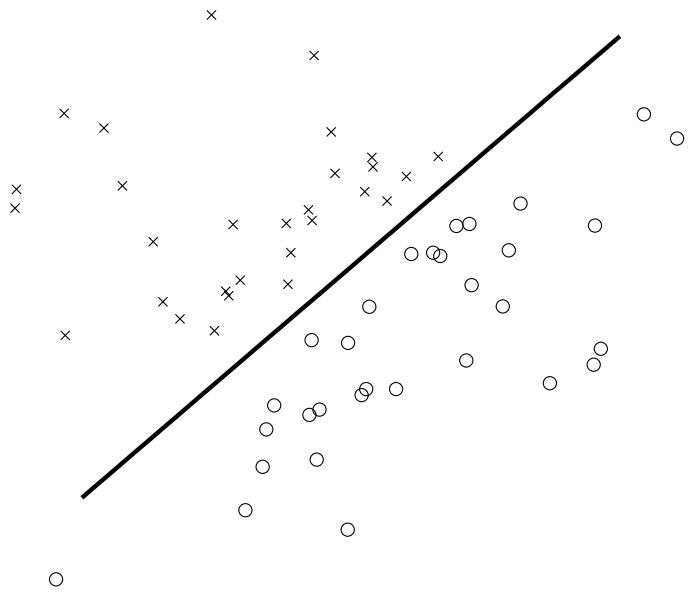


FIGURE 3.3.1: Two classes of data in $\mathbb{R}^2$ separated by a hyperplane (line). This hyperplane is optimal because it maximizes the margin between the two sets of data. We predict the class of any new datum based on which side of the hyperplane it lies.

all of that type data lies to one side. Let us begin with

$$\begin{aligned} H_1 &= \{x \in \mathbb{R}^n \mid \omega' \cdot x = \omega' \cdot p_1 \text{ for some } p_1 \in H_1\} \\ &= \{x \in \mathbb{R}^n \mid \omega' \cdot x + b_1 = 0\}, \\ H_2 &= \{x \in \mathbb{R}^n \mid \omega' \cdot x = \omega' \cdot p_2 \text{ for some } p_2 \in H_2\} \\ &= \{x \in \mathbb{R}^n \mid \omega' \cdot x + b_2 = 0\}, \end{aligned}$$

(where b_1 and b_2 denote $-\omega' \cdot p_1$ and $-\omega' \cdot p_2$, respectively)

as our marginal hyperplanes. Our decision boundary will lie between them. It is desired that our decision hyperplane lies parallel to and equidistant from each marginal hyperplane. For any hyperplane H_i determined by ω and b_i, we will use the formula for the distance from a point to that hyperplane:

$$\text{dist}(x, H_i) = \frac{|\langle \omega, x \rangle + b_i|}{\|\omega\|}.$$

To find the distance between two parallel hyperplanes (with equal ω), we see that for any

point $\bar{x} \in H_1$, its distance to H_2 is

$$\text{dist}(\bar{x}, H_2) = \frac{|\langle \omega', \bar{x} \rangle + b_2|}{\|\omega\|}.$$

Since $\bar{x}$ lies on H_1, we have $\langle \omega', \bar{x} \rangle = -b_1$, and so the distance from H_1 to H_2 is $\frac{|b_2 - b_1|}{\|\omega'\|}$. Note that if a third hyperplane has a $b' = \frac{b_1 + b_2}{2}$, it will be equidistant to both H_1 and H_2. We use that b' value for our decision hyperplane H' and express our hyperplanes as

$$\begin{aligned}
H' &= \{x \in \mathbb{R}^n \mid \omega' \cdot x + b' = 0\}, \\
H_1 &= \{x \in \mathbb{R}^n \mid \omega' \cdot x + b' + (b_1 - b') = 0\} \\
&= \{x \in \mathbb{R}^n \mid \omega' \cdot x + b' = c\}, \\
H_2 &= \{x \in \mathbb{R}^n \mid \omega' \cdot x + b' + (b_2 - b') = 0\} \\
&= \{x \in \mathbb{R}^n \mid \omega' \cdot x + b' = -c\},
\end{aligned}$$

$$\text{where } c = \tfrac{b_2 - b_1}{2}.$$

We now unitize by dividing by c (assuming $H_1 \neq H_2$), so that with $b = b'/c$ and $\omega = \omega'/c$, we have

$$\begin{aligned}
H' &= \{x \in \mathbb{R}^n \mid \omega \cdot x + b = 0\}, \\
H_1 &= \{x \in \mathbb{R}^n \mid \omega \cdot x + b = 1\}, \\
H_2 &= \{x \in \mathbb{R}^n \mid \omega \cdot x + b = -1\}.
\end{aligned}$$

Returning to the distance formula, it is simple to check that the distance between H_1 and H_2 is now expressed as $\frac{2}{\|\omega\|}$. For greatest margin, we maximize this distance or, equivalently, minimize $\frac{1}{2}\|\omega\|^2$.

We of course want the data of each type to lie on opposite sides of these hyperplanes, and this is achieved if, for all x of type $y_i = 1$ we have $\langle \omega, x \rangle + b \geq 1$ and for all x of type $y_i = -1$ we have $\langle \omega, x \rangle + b \leq -1$. Note that if a datapoint lies 'above' H_1 or 'below' H_2 then it also lies on that same side of H'. We seek a decision hyperplane for which all datapoints of a given class lie on opposite sides, so that for a given x_i, $\langle \omega, x_i \rangle + b \geq 1$ if $i = 1$ and $\langle \omega, x_i \rangle + b \leq -1$ if $i = -1$. Note that regardless of the value of y_i, this is equivalent to $y_i(\langle \omega, x_i \rangle + b) \geq 1$. For any point/class pair (x_i, y_i), let us denote C_i as the set of all hyperplanes (each denoted $H_{\omega,b} = \{x | \omega \cdot x + b = 0\}$) such that this equation holds. That is,

$$C_i = \{H_{\omega,b} \mid y_i(\langle \omega, x_i \rangle + b) \geq 1\}.$$

We seek a hyperplane which partitions all the data into opposite halfspaces, so our optimal hyperplane must lie in the intersection of all C_i. This serves as our constraint for the

support vector machine problem. At this stage, we seek to minimize $\frac{1}{2}\|\omega\|^2$ such that ω is a normal vector to a hyperplane which lies in the intersection of all C_i.

We now apply a common technique in optimization, which relies on the following equivalency:

$$\langle w, x\rangle + b = w_1x_1 + \cdots + w_nx_n + b = \langle \theta, \mathbf{x}\rangle,$$

where θ is an $n+1$ coordinate vector with entries $\omega_1, ..., \omega_n, b$ and $\mathbf{x}$ is identical to x, except with a 1 concatenated as its $(n+1)^{\text{th}}$ coordinate. This effectively raises the dimension in which we work to $\mathbb{R}^{n+1}$. In this way, for any x_i, y_i, we express the corresponding constraint set as

$$C_i = \{\theta \in \mathbb{R}^{n+1} \mid y_i\langle \theta, \mathbf{x}_i\rangle \geq 1\}.$$

In this higher dimensional space, with coordinate system $\omega_1, ..., \omega_n, b$, each point θ determines a hyperplane back in $\mathbb{R}^n$. Meanwhile, any point $x_i \in \mathbb{R}^n$ along with its y_i, determines a halfspace in $\mathbb{R}^{n+1}$. These halfspaces in $\mathbb{R}^{n+1}$ are our constraint sets C_i, which ensure our decision boundary is on the correct side of $x_i \in \mathbb{R}^n$. Our problem is now stated as

$$\text{minimize } \frac{1}{2}\|\theta\|^2 \qquad \text{such that } \theta \in \bigcap_i C_i.$$

Note that this is simply a closest point problem: find the closest $\theta \in \mathbb{R}^{n+1}$ to 0 of all θ in an intersection. In the same way then, we may express it, for N data points $x_1, ..., x_N$, as

$$\text{minimize } f(\theta) = \frac{1}{2}\|\theta\|^2 + \frac{\mu}{2}\sum_i^N \text{dist}(\theta, C_i)^2.$$

Applying a distance majorization surrogate

$$g(\theta, \theta_k) = \frac{1}{2}\|\theta\|^2 + \frac{\mu}{2}\sum_i^N \|\theta - P_{C_i}(\theta_k)\|^2,$$

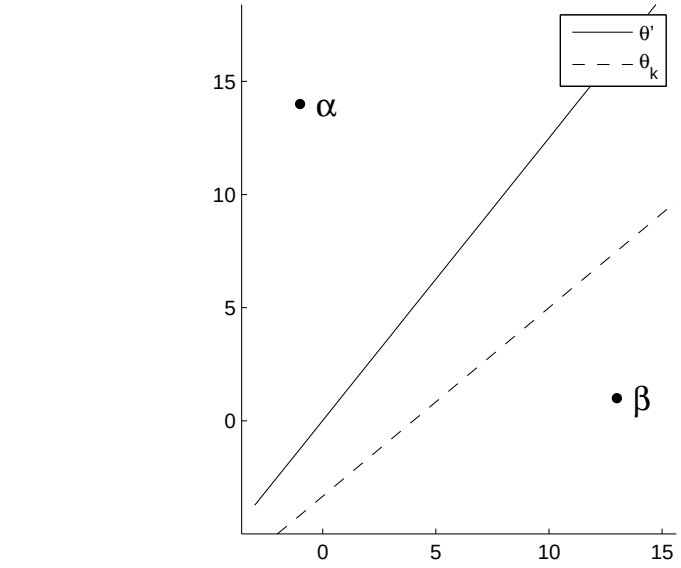
we obtain the updates

$$\theta_{k+1} := \frac{\mu}{1+N\mu}\sum_i^N P_{C_i}(\theta_k).$$

Because this is now reduced to a closest point problem, we obtain the same convergence rate, for some optimal solution θ^*:

$$f(\theta_k) - f(\theta^*) \leq \frac{m\mu\|\theta_0 - \theta^*\|^2}{2k}.$$

In chapter 4, we will see how to implement this algorithm, accounting for feature types and half-space projections.



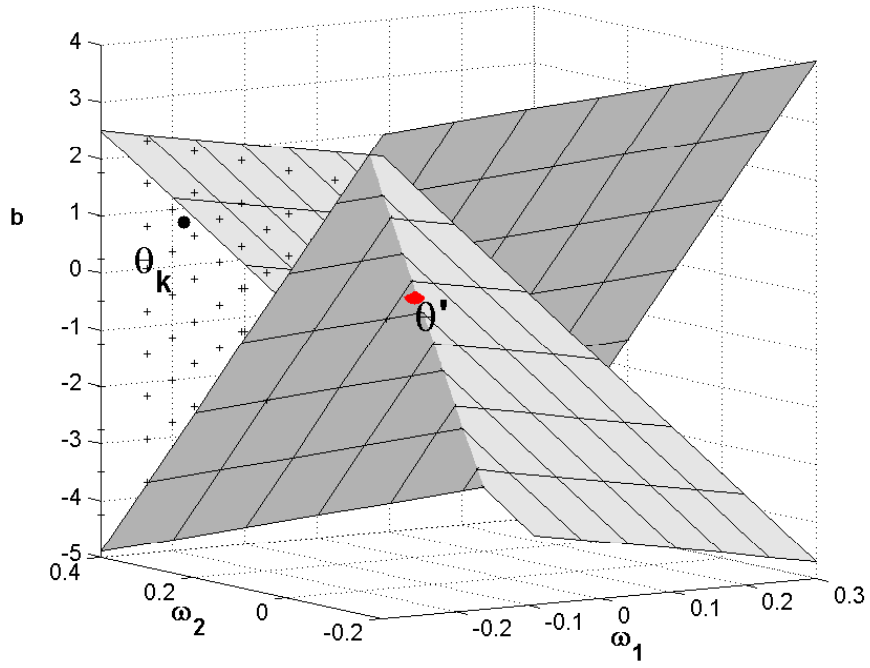


FIGURE 3.3.2: Support vector machine for data in $\mathbb{R}^2$.

top: Optimal hyperplane $\theta' = \{\mathbf{x} = (x, y) \in \mathbb{R}^2 \big| -.08x + .07y - 0.0008 = 0\}$ separates 2 pts $\alpha = (-1, 14)$ and $\beta = (13, 1)$ in $\mathbb{R}^2$.

bottom: In $\mathbb{R}^3$, each point $\theta = (\omega_1, \omega_2, b)$ corresponds to a line in $\mathbb{R}^2$. The point α defines a halfspace $C_1 = \{\theta \in \mathbb{R}^3 \big| -\omega_1 + 14\omega_2 + b \geq 1\}$ (above the dark gray plane) and β another, $C_2 = \{\theta \in \mathbb{R}^3 \big| 13\omega_1 + \omega_2 + b \leq -1\}$ (below the light gray plane.) If some point θ lies in their intersection, then its corresponding line in $\mathbb{R}^2$ separates α and β. The θ with minimal norm in this intersection (that is, closest to the origin) maximizes the margin between its hyperplane in $\mathbb{R}^2$ and the points it separates. Note θ_k lies in the intersection, but is not optimal, as can be seen by its hyperplane in $\mathbb{R}^2$.

3.4 The Smallest Intersecting Ball Problem

The smallest intersecting ball problem asks for the center of a ball with the smallest radius necessary to intersect a collection of closed sets $\Omega_1, ..., \Omega_m$. From any center $\bar{x}$, the ball $\mathbb{B}(\bar{x}, \max\{\text{dist}(x, \Omega_i)\})$ is guaranteed to intersect each Ω_i for $i = 1, ..., m$, so the problem can be expressed as the optimization problem

$$\min_{x \in \mathbb{R}^n} \ \mathcal{D}(x) := \max\{\text{dist}(x, \Omega_i) \big| i = 1, 2, ..., m\}. \tag{3.4.1}$$

This section will present three distinct methods of finding the solution which exploit the MM-principle, though in different ways.

3.4.1 Log Exponential Smoothing

For a detailed exposition of this approach, see [1], [16]; we present here an outline. The max distance function $\mathcal{D}(x)$ is a non-smooth function, and so to employ the MM principle, it can be approximated by the C^1 log-exponential smoothing function

$$\mathcal{G}_p(x) = p \ln \sum_{i=1}^{m} \mathrm{e}^{\frac{\sqrt{\text{dist}(x,\Omega_i)^2 + p^2}}{p}},$$

with an error given by

$$0 \leq \mathcal{G}_p(x) - \mathcal{D}(x) \leq p(1 + \ln(m)).$$

To minimize this now differentiable function, we may apply the MM principle. The surrogate function utilizes distance majorization in the same way as the feasible point problem, with each iterate given by

$$\mathcal{G}_p(x, \kappa) := p \ln \sum_{i=1}^{m} \mathrm{e}^{\frac{\sqrt{\|x - P_{\Omega_i}(\kappa)\|^2 + p^2}}{p}}. \tag{3.4.2}$$

For each update of the MM algorithm we must find the minimum of this function, but an explicit solution for x such that $\nabla \mathcal{G}(x) = 0$ is not available. However, because $\nabla \mathcal{G}(x, p)$ is $\frac{2}{p}$-Lipschitz continuous (see [15],Prop.2), we may make use of Nesterov's accelerated gradient method given in section 2.3 to minimize each iteration of the surrogate. Careful application of the chain rule provides the gradient of $\mathcal{G}_p(x, \kappa)$ to be

$$\nabla \mathcal{G}_p(x, \kappa) = \left(\sum_{j=1}^{m} \mathrm{e}^{\frac{g_j(x,\kappa,p)}{p}} \right)^{-1} \sum_{i=1}^{m} \frac{x - P_{\Omega_i}(\kappa)}{g_i(x, \kappa, p)} \mathrm{e}^{\frac{g_i(x,\kappa,p)}{p}},$$

where g_i is given by

$$g_i(x, \kappa, p) = \sqrt{\|x - P_{\Omega_i}(\kappa)\|^2 + p^2}.$$

To be clear, the surrogate function 3.4.2 is the log smoothing approximation for the max distance to the projections of the anchor κ onto each set. Nesterov's accelerated gradient method then solves the smallest 'intersecting' ball for those points, whose center serves as

the next 'anchor' point, the projections of which are then used to make the next set of points for which Nesterov's method finds the smallest ball.

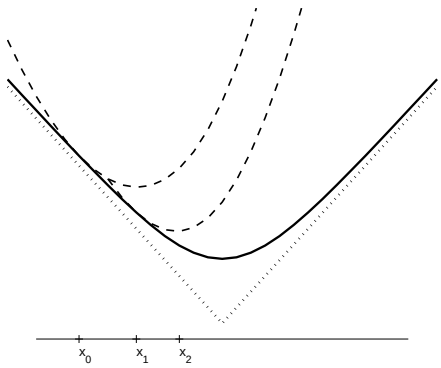


FIGURE 3.4.1: Layers of a smallest intersecting ball algorithm. The non-smooth max distance function is represented by the dotted line. It is approximated by a (solid) log-exponential smoothing function. Each (dashed) surrogate curve is minimized by Nesterov's accelerated gradient method, and the optimal solution serves as the tangent point for the next surrogate.

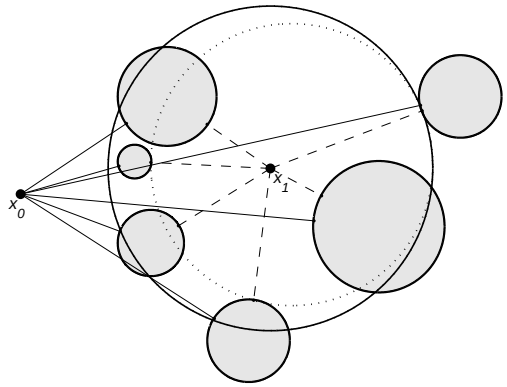


FIGURE 3.4.2: Solving the smallest intersecting ball with log-exponential/ MM algorithm in $\mathbb{R}^2$. The first surrogate function uses the projections of x_0. Minimizing it gives the optimal center x_1 whose ball (solid) intersects those projections. The next surrogate is minimized to find the smallest circle (dashed) which intersects the projections of x_1; its optimal center x_2 is not shown.

3.4.2 Expanding Sets

The expanding sets approach to the smallest intersecting ball problem 'inflates' each of the target sets just enough to give a nonempty intersection. We then find a point in this intersection as the optimal solution to 3.4.1. To see that this indeed minimizes the max distance function, we begin by defining a t-expansion of any closed set Ω_i as

$$\Omega_{i,t} = \{x \in \mathbb{R}^n | \text{dist}(x, \Omega_i) \leq t\},$$

and let

$$T = \inf \left\{ t \in \mathbb{R} \mid \bigcap_i^m \Omega_{i,t} \neq \varnothing \right\}.$$

That is, T is the smallest expansion term such that the intersection of all t-expanded sets is non-empty. We assume that each Ω_i is bound, so that T exists.

Proposition 3.4.1. Any element x lies in $\bigcap \Omega_{i,T}$ if and only if it is an optimal solution to $\mathcal{D}(x)$ (3.4.1).

Proof. Choose some $y \in \cap \Omega_{i,T}$. By its inclusion in the intersection, $\text{dist}(y, \Omega_i) \leq T$ for all $i = 1, 2, ..., m$. This includes its most distant set also, so $\mathcal{D}(y) \leq T$. Further, we find $y \in \bigcap \Omega_{i,\mathcal{D}(y)}$, so $\bigcap \Omega_{i,\mathcal{D}(y)}$ is non-empty. Because T is the infimal value for non-empty intersection, we must have $T \leq \mathcal{D}(y)$. This demonstrates that for arbitrary $y \in \cap \Omega_{i,T}$, the distance to the farthest set from it is T:

$$\max \{\text{dist}(y, \Omega_i), i = 1, 2, ..., m\} = T.$$

Now let x^* be an optimal solution to $\mathcal{D}(x)$. For any Ω_j, we have

$$\begin{aligned} \text{dist}(x^*, \Omega_j) &\leq \max\{\text{dist}(x^*, \Omega_i), i = 1, ..., m\} \\ &\leq \max\{\text{dist}(y, \Omega_i), i = 1, ..., m\} \\ &= T. \end{aligned}$$

With the distance from x^* to all target sets bound by T, we have that

$$x^* \in \bigcap_i \Omega_{i,T},$$

from which it follows that

$$\mathcal{D}(x^*) = \max \{\text{dist}(x^*, \Omega_i), i = 1, 2, ..., m)\} = T.$$

So for any $y \in \bigcap \Omega_{i,T}$, and all $x \in \mathbb{R}^n$,

$$\mathcal{D}(y) = T = \mathcal{D}(x^*) \leq \mathcal{D}(x),$$

so that y is an optimal solution. □

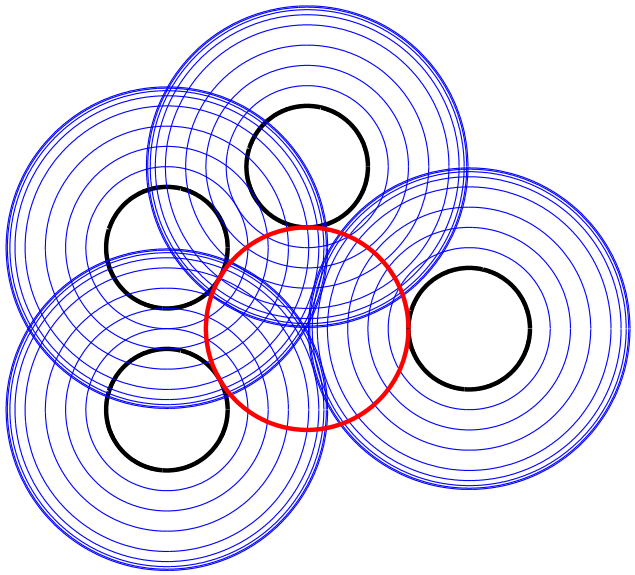


FIGURE 3.4.3: Expanding sets in $\mathbb{R}^2$. An element lies in the in the infimal intersection of expanded sets if and only if it is an optimal solution. We must find the infimal expansion for non-empty intersection, so if expanding the sets by τ causes non-empty intersection, then expand by only $\frac{\tau}{2}$.

With the above in mind, finding the smallest intersecting ball can be reduced to the following problem:

$$\min_t \bigcap_{i=1}^{m} \Omega_{i,t} \neq \varnothing.$$

Once the infimal nonempty intersection is established, a feasible point algorithm can locate a point therein.

The infimum T for the non-empty intersection of expanded sets can be approximated by incrementally expanding the sets by some $\tau > 0$, and searching at each increment for their intersection with a feasible point algorithm. If the algorithm converges with non-zero distance to the expanded sets, we can assume their intersection is empty. If instead the sum of distances is zero, then the expansion was too great. The radii are then decreased and expansion begins again with a smaller step size τ.

3.4.3 Weighted Projections

This approach to finding a smallest intersecting ball has not yet been proven to converge to the optimal solution but performs competively with other methods, and so is included as a work in progress. The weighted projection method seeks to minimize the max distance function by minimizing the distance to *all* target sets, but prioritizing those which are most distant from x at each iteration. The weight given to each set is adjusted until the distances from x to its most distant sets are both equal, and minimal.

Recall that the simultaneous projection algorithm minimizes the function

$$f(x) = \sum_{i=1}^{m} \gamma_i \text{dist}(x, \Omega_i)^2$$

with the update:

$$x_{n+1} = \sum_{i}^{m} \gamma_i P_{\Omega_i}(x_n).$$

We previously have implicitly assigned each $\gamma_i = 1/m$. The weighted projections approach is directly based on the idea, from *Distance Majorization and its Applications* by Chi, Zhou, & Lange (2014), that

> "Uniform γ_i weights equally penalize an iterate's violation of each constraint. Nonuniform weights will penalize constraint violations differently. This can be a useful mechanism if it is more important to satisfy some constraints over others in an application."

In this case, we seek only to penalize distance from the sets which are most distant. After the sum of distances to all sets are minimized for a given set of γ_i's, the γ_i for any sets which are not most distant donate some of their weight to that of the most distant set. Starting with an initial $\gamma_0^i = 1/m$ for all i, the weights are assigned as

$$\gamma_{k+1}^i = \begin{cases} \gamma_k^i + \sum_{j \neq i}^{m} s_k^j & \text{if } \Omega_i \text{ is the most distant set from } x_k \\ \gamma_k^i - s_k^i, & \text{otherwise} \end{cases}$$

$$s_k^i = \min\{c, \gamma_k^i\},$$

where c is a weight donation term which starts at $1/m$ and is eventually driven to zero. The shift in weight can be shown to cause each iteration to move towards whichever set was previously the most distant, and as c approaches zero, the distance between x_k and x_{k+1} approaches zero.

Chapter 4

Implementation in MATLAB

In this chapter, we implement our algorithms in *MATLAB* to solve the feasible and closest point problems, the support vector machine problem, and the smallest intersecting ball problem. We will proceed as though all constraint sets are closed balls (except for the halfspaces used for support vector machines,) but the functions can be modified to accommodate any convex sets, as long as one can compute their respective projections. Let us begin with a function to vectorize the computation of projections onto an arbitrary number of balls.

4.1 Supporting Functions

4.1.1 Projection onto a Ball

In many of our algorithms, we must compute the projection of $x \in \mathbb{R}^n$ onto a ball $\mathcal{B}$ with center c and radius r. The formula for this is given as

$$P_{\mathcal{B}}(x) = \begin{cases} \frac{x-c}{\|x-c\|}r + c, & x \notin \mathcal{B} \\ x, & x \in \mathcal{B}. \end{cases}$$

The projection is conditional on set inclusion, which invites the use of an *if* statement, nested in a *for* loop which repeats for each ball. We can avoid unnecessary computional costs by using logical indexing. Logical indexing returns a matrix whose entries are 0 or 1, depending on whether some condition is true. For example, if $A = \left(\begin{smallmatrix} 1 & 2 \\ 2 & 4 \end{smallmatrix}\right)$ then `B=(A==2)` returns the matrix `B=`$\left(\begin{smallmatrix} 0 & 1 \\ 1 & 0 \end{smallmatrix}\right)$. For the projection of $x \in \mathbb{R}^n$ onto m balls, we proceed as follows:

- Use `repmat` to construct an $m \times n$ matrix `X`, with a copy of x in each row.

- Compute an $m \times n$ matrix `distToCenters`, holding distance to center i in all columns of i^{th} row.
- Create an $m \times n$ matrix `multiRad` holding radius of ball i in every column of i^{th} row.
- Use logical indexing to make two case-conditional $m \times n$ matrices,

  `XinB` holds 1's in row j if $x \in B_j$ and 0's if $x \notin B_j$.

  `XnotInB` holds 1's in row j if $x \notin B_j$ and 0's if $x \in B_j$.
- Compute matrix `P` whose i^{th} row holds the projection of x onto B_i in the case that x does not lie in B_i. (The matrix `X` holds the projections of x in the case that x does lie in some ball.)
- Multiply (entrywise) the projection matrices with their corresponding case-conditional matrices to send to zero any rows whose condition is false.
- The sum of the resulting products is an $m \times n$ matrix holding the projections of x onto m sets.

The function will work in $\mathbb{R}^n$, and accept as arguments a $1 \times n$ vector `x`, m centers in an $m \times n$ matrix `C`, with corresponding radii in an $m \times 1$ matrix `R`. It returns an $m \times n$ matrix whose i^{th} row holds the projection onto the i^{th} ball.

```
function [PROJECTIONS]=projectOntoBall(x,C,R)
numBalls=size(C,1);
dim=size(C,2);
X=repmat(x,numBalls,1);
distToCenters=sqrt(sum((X-C).^2,2));
XinB=repmat(distToCenters<=R,1,n);
XnotInB=repmat(distToCenters>R,1,n);
multiNorms=repmat(distToCenters,1,n);
multiRadii=repmat(R,1,n);
P=(X-C)./multiNorms.*multiRad+C;
PROJECTIONS=P.*XnotinB+X.*XinB;
end
```

Using the above function eliminates the need for *if* and *for* loops, which can decrease run times significantly, as seen in table 4.1.1.

Number of Balls	10	100	1,000	10,000
If/for loop	.0002	.0010	.0129	.59
Logical Indexing	.0001	.0002	.0005	.01

TABLE 4.1.1: Computation time in seconds for projection onto balls in $\mathbb{R}^3$. Vectorization with logical indexing is more efficient, especially when dealing with high numbers of sets.

4.1.2 Projection onto a Halfspace

The support vector machine algorithm requires projection onto half-spaces. For a halfspace $H = \{x \in \mathbb{R}^n \big| \langle \omega, x \rangle \geq b\}$, the projection of x onto H is given by

$$P_H(x) = \begin{cases} x + \frac{b - \langle \omega, x \rangle}{\|\omega\|^2}\omega, & x \notin H \\ x, & x \in H. \end{cases}$$

For an intuitive understanding of this, note that $\frac{\langle \omega, x \rangle - b}{\|\omega\|}$ is the distance from x to the hyperplane boundary H, and this scales the unit normal vector to the halfspace $\frac{\omega}{\|\omega\|}$ sufficiently to reach the halfspace from x. Care should be taken to ensure the vector ω is pointing to H, rather than away from it.

The following function is built specifically for use in support vector machine algorithms. Remember in our algorithm we are projecting $\theta \in \mathbb{R}^{n+1}$ onto halfspaces defined as $H_i = \{\theta \big| y_i \langle \theta, x_i \rangle \geq 1\}$, where $x_i \in \mathbb{R}^{n+1}$ is a data point in $\mathbb{R}^n$ with a 1 concatenated in its final entry. It takes as arguments a $1 \times (n+1)$ vector `z`, an $m \times 1$ column vector `Y` of class labels $\{-1, 1\}$, and an $m \times n$ matrix `W`, with each row holding the values which determine the halfspace: $x_1^i, x_2^i, ..., x_n^i, 1$. Using logical indexing for a loopless projection as we did for projections onto balls, our function follows.

```
function [PROJECTIONS]=projectOntoHalfspaces(z,Y,W)
numPlanes=size(W,1);
dim=size(W,2);
Z=repmat(z,numPlanes,1);
Wnorm= sqrt(sum(W.^2,2));
ZisInSpace= repmat(Y.*(W*z')>=1,1,dim);
ZnotInSpace= repmat(Y.*(W*z')<1,1,dim);
caseZnotInSpace= Z + repmat((Y-W*z')./(Wnorm.^2),1,dim).*W;
caseZisInSpace= Z;
PROJECTIONS= caseZisInSpace.*ZisInSpace + caseZnotInSpace.*ZnotInSpace;
end
```

4.1.3 Creating Balls with Non-Empty Intersection

So that we may test our feasible point algorithm, let us construct a collection of m balls in $\mathbb{R}^n$ with non-empty intersection. This function will return a matrix `C` containing a center in each row, and a vector `R` of their corresponding radii. The approach is outlined as follows:

- Assign m centers randomly.
- Choose the first two radii randomly.
- Increase the radii of these 1^{st} two balls until the sum of their radii is greater than the distance between their centers.

- Find a point in the intersection of these first two balls.
- Assign all other balls radii sufficiently large to enclose that point.

```
function [C,R] = intersectingSetOfBalls(numBalls,dim)
C=rand(numBalls,dim);
R=[rand(2,1); zeros(numBalls-2,1)];
while norm(C(2,:)-C(1,:)) > (R(1)+R(2))
    R(2)=R(2)+.01;
end
point = (((C(1,:)-C(2,:)) / norm(C(1,:)-C(2,:)) * R(2) + C(2,:)) + ...
         ((C(2,:)-C(1,:)) / norm(C(2,:)-C(1,:)) * R(1) + C(1,:))) / 2;
for k=3:numBalls
    R(k) = norm(C(k,:)-point) + .01;
end
end
```

4.1.4 Creating Balls with Empty Intersection

For the smallest intersecting ball problem (to be non-trivial), we need sets with empty intersection. Generating random balls is simple in *MATLAB* using the following function.

```
function [SETS,RADII] = randomBalls(numBalls,dim)
SETS=100*randn(numBalls,dim);
RADII=10*rand(numBalls,1);
end
```

We assume, due to the large variance of distribution of centers relative to radii, that a nonempty intersection is sufficiently improbable.

4.1.5 Generating Linearly Separable Data

For preliminary testing of our support vector machine algorithm, it is useful to use artificial data, perhaps for visual confirmation in $\mathbb{R}^2$ of a sensible solution. To do this, we can first generate a random normal vector $\omega = w_1, ..., w_n$, and a c value, from which we obtain a hyperplane; $H = \{x | \langle \omega, x \rangle = 0\}$ and two marginal hyperplanes; data will fall into class A if $\langle \omega, x \rangle \geq c$ and class B if $\langle \omega, x \rangle \leq -c$. We then generate a random point and depending on which class it falls into, concatenate either a 1 or -1. (If it falls in neither, it is ignored.) We continue in this way until we have an $m \times (n+1)$ matrix of m data points in $\mathbb{R}^n$, whose last column holds the $\{-1, 1\}$ classifer.

```
function [X] = PointGenerator( numPoints,dim )
w=10*randn(1,dim);
numAssigned=0;
c=30;
X=[];
```

```
while numAssigned < numPoints
    pt=10*randn(1,dim);
    if w*pt' > c
        X=[X; pt, 1];
        numAssigned=numAssigned+1;
    elseif w*pt' < -c
        X=[X; pt, -1];
        numAssigned=numAssigned+1;
    end
end
end
```

4.2 The Feasible Point Problem

With our `projectOntoBalls` function in hand, implementing a distance majorization algorithm to solve the feasible point problem is but a few lines. We choose $x_0 = 0$, and at each iteration compute the projections of x_k onto the m balls, and then define `x1` as the average of those projections. We use, as suggested in [5], `step`$= \frac{\|x_{k+1}-x_k\|}{\|x_k\|+1}$ as a measure of convergence. When `step` falls below some threshold (in this case 10^{-6}), the loop terminates. Taking as arguments a matrix `SETS` (with centers in each row) and `RADII` (a column vector of corresponding radii) this function returns an absolute minimum `x`:

```
function [x]=feasiblePoint(SETS,RADII)
numBalls=size(SETS,1);
dim=size(SETS,2);
x=zeros(1,dim);
step=999;
while step > 1e-6
    PROJ=projectOntoBalls(x,SETS,RADII);
    x1=sum(PROJ,1)/numBalls;
    step=norm(x - x1)/(norm(x)+1);
    x=x1;
end
end
```

To instead implement the Lipschitz surrogate/ gradient method, we use the update $x_{k+1} := x_k - \frac{1}{L}\nabla f(x_k)$, where $L = 2m$ and $\nabla f(x_k) = mx_k - \sum_i^m P_{\Omega_i}(x_k)$, so that

$$x_{k+1} = \frac{1}{2}x_k + \frac{1}{2m}\sum_i^m P_{\Omega_i}(x_k).$$

Using the same stopping criterion as the distance majorization algorithm, the function can be coded as

```
function [x] = feasibleGradient(x,SETS,RADII)
```

```
numSets=size(SETS,1);
dim=size(SETS,2);
gradient=ones(1,dim);
step=999;
while step > 1e-6
  PROJ=projectOntoBalls(x,SETS,RADII);
  x1= .5 * (x + sum(PROJ,1)/numSets);
  step=norm(x - x1)/(norm(x)+1);
  x=x1;
end
end
```

Let us now modify this code to produce the accelerated gradient method. Recall the accelerated update is $x_{k+1} := (1-\gamma_k)y_{k+1} + \gamma_k y_k$, where $y_{k+1} := x_k - \frac{1}{L}\nabla f(x_k)$. It is possible to compute a static vector of the required γ_k terms before the loop begins, but (because the algorithm runs until the iterations meet some measure of convergence) we do not know how many terms we will need. Instead we can dynamically update each term as needed. The sequences are given by

$$\lambda_0 = 0, \quad \lambda_k = \frac{1+\sqrt{1+4\lambda_{k-1}^2}}{2}, \quad \gamma_k = \frac{1-\lambda_k}{\lambda_{k+1}}.$$

Computing x_{k+1} requires y_k (`y`) and y_{k+1} (`yNext`) and γ_k (`gamma`), which in turn requires λ_k (`lambda`) and λ_{k+1} (`lambdaNext`.) After finding the projections, we compute `yNext` and `lambdaNext`, which we use with `lambda` to find γ_k (`gamma`). This allows the update $x_{k+1} = (1-\gamma_k)y_{k+1} + \gamma_k y_k$ which we code as `x = (1-gamma)*yNext + gamma*y`. We are now finished with y_k and λ_k (the next update will use the $k+1$ and $k+2$ terms), so we assign them to hold the $k+1$ terms. The next iteration of the loop will use them to determine the $k+2$ terms. Note that this sequence generates $x_1 = x_0$, and so to prevent the `while` loop from immediately terminating we start with $\lambda_0 = 1$.

```
function [x] = feasibleAccel(x,SETS,RADII)
numSets=size(SETS,1);
dim=size(SETS,2);
y = x;
lambda=1;
step=999;
while step > 1e-6
   PROJ=projectOntoBalls(x,SETS,RADII);
   yNext= .5*( x + sum(PROJ,1)/numSets);
   lambdaNext=(1+sqrt(1+4*lambda^2))/(2);
   gamma=(1-lambda)/lambdaNext;
   x1 = (1-gamma)*yNext + gamma*y;
   y=yNext;
```

```
   lambda=lambdaNext;
   step=norm(x - x1)/(norm(x)+1);
   x=x1;
end
end
```

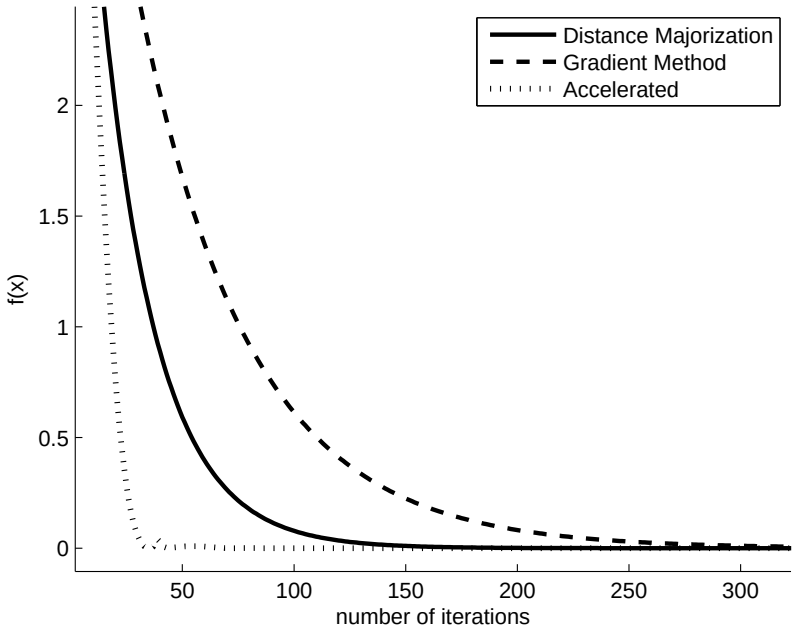


FIGURE 4.2.1: Finding a feasible point in the intersection of 50 balls in $\mathbb{R}^{50}$. Distance majorization converges faster than the gradient method, but Nesterov acceleration performs substantially better, as expected.

4.3 The Closest Point Problem

To implement the closest point algorithm, we will use an increasing sequence of penalty weights $\mu_i = 2^i - 1$, as suggested in [5]. As i ranges from 0 to 20, we have μ increase from zero–so that our first iteration is y itself—to over one million. To check convergence under a given μ penalty, we compute `step`, defined as in the feasible point algorithm. When `step` falls below some threshold, in this case 10^{-4} we increase μ to μ_{i+1}. Passing, as usual, a matrix of centers `C` and a vector `R` of their radii to the function, along with the fixed point y whose distance from our optimal solution we desire to minimize, we return the optimal solution x as follows:

```
function [x] = closestPoint(C,R,y)
numBalls=size(C,1);
dim=size(C,2);
```

```
x=zeros(1,dim);
for i=1:20
    mu=2^i-1;
    step=999;
    while step > 1e-4
        PROJ = projectOntoBalls(x,C,R);
        x1 = y ./(1+mu*numBalls) + mu/(1+mu*numBalls)*sum(PROJ);
        step=norm(x - x1)/(norm(x)+1);
        x=x1;
    end
end
end
```

4.4 The Support Vector Machine Problem

For support vector machine problems we will be given a set of m data points in n dimensions, with each datum x_i associated with a $y_i = 1$ or $y_i = -1$ to determine its type. We will pass to our function an $m \times n + 1$ matrix, with the y_i values in the $n + 1^{\text{th}}$ column. We again use the increasing sequence of μ penalties described in the closest point function.

In the code below, after accounting for the size of the data and its dimension, we isolate the x and y values from the input data– `X` holds the training data points and `Y` their $\{-1, 1\}$ class types. We use `dim2` as an index to concatenate 1's onto our x_i data. From here, the algorithm is the same as the closest point (to 0) problem, except that the constraint sets are halfspaces.

```
function [theta] = svm(DATA)
numData=size(DATA,1);
dim1=size(DATA,2)-1;
X=DATA(:,1:dim1);
Y=DATA(:,dim1+1);
dim2=dim1+1;
X(:,dim2)=1;
theta=zeros(1,dim2);
step=99;
for i=1:20
    mu=2^i-1;
    while step > 1e-4
        PROJECTIONS=projectOntoHalfspaces(theta,Y,X);
        theta1= mu/(1+numData*mu)*sum(PROJECTIONS,1);
        step=norm(theta - theta1)/(norm(theta)+1);
        theta=theta1;
    end
end
```

```
end
```

4.4.1 Testing the SVM Algorithm with the Iris Data Set

Fisher's Iris data set is a publicly available collection of measurements of Iris flowers [8] made in 1936 by statistician and biologist Sir Ronald Fisher. The data consists of 50 samples for each of 3 Iris species: *setosa*, *virginica*, and *versicolor*, with each sample containing four features: sepal length, sepal width, petal length, and petal width. This data is a classical case study in machine learning, allowing algorithms to be tested for their ability to correctly classify unknown observations based on some known training data subset of the 150 observations.

Here, we choose k observations from each species, and use a support vector machine algorithm to construct 3 hyperplanes—one to separate each pair of species. Then each of the remaining $(50-k)$ unknown samples from each species are tested to determine which side of each hyperplane they lie. For instance, a hyperplane θ distinguishes *setosa* from *versicolor*, α distinguishes *setosa* from *virginica*, and β separates *versicolor* from *virginica*. The unknown samples are classified as whichever species is predicted twice. For example, a sample may result in a prediction of *setosa–virginica–virginica*, from θ, α, and β, respectively, in which case it is classified as virginica.

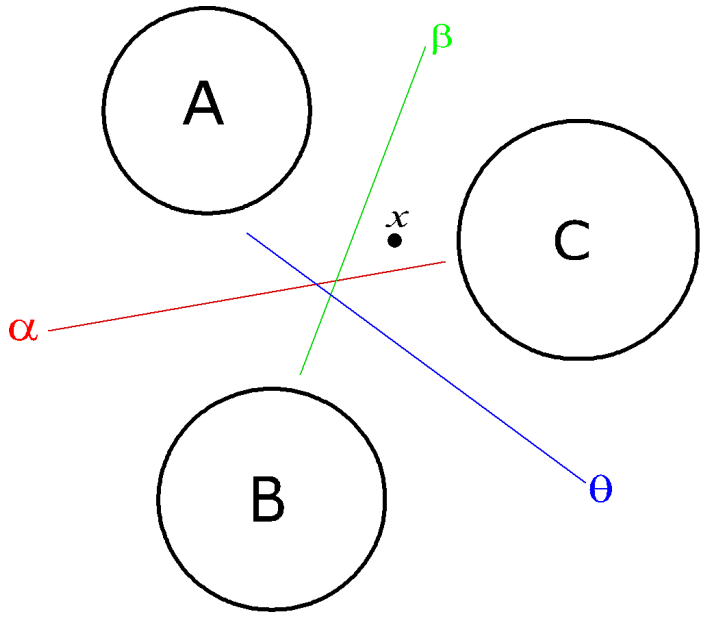


FIGURE 4.4.1: Multiple hyperplanes used to classify an element into 1 of 3 data types. The data classes A and B are separated by α, classes B and C are separated by θ, and A and C, by β. In this example, the hyperplanes respectively predict that x belongs to A, C, and C; with C receiving 2 out of 3 hits, the element is classified as C-type. If no class appears in the majority, then the test is inconclusive.

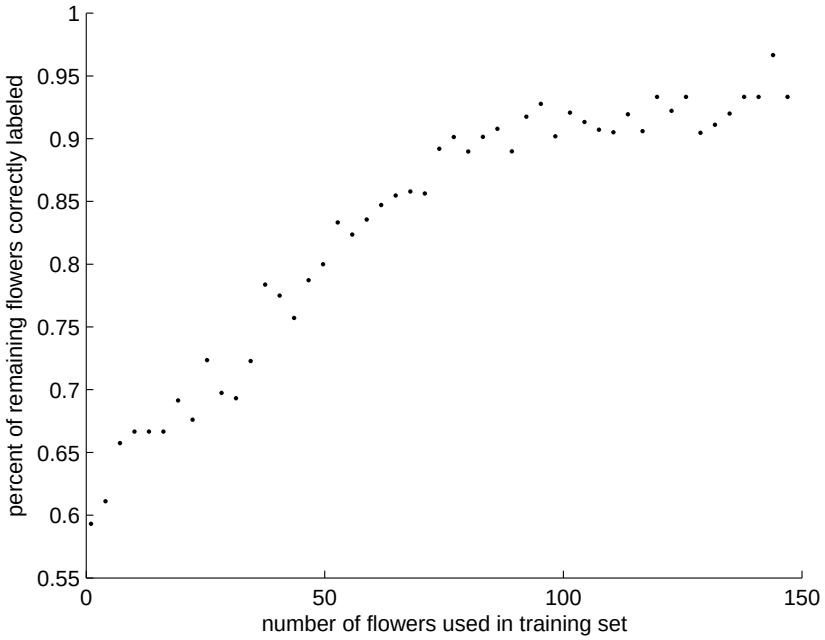


FIGURE 4.4.2: Average (n=10) percent of correctly predicted *Iris* species versus number of samples used in the training set. At greater than 75 of the 150 observations, the algorithm correctly predicted flower types with over 90% success. Note that *versicolor* and *virginica* classes are not linearly separable, so it is not expected that a support vector machine attain perfect predictive power.

4.5 The Smallest Intersecting Ball Problem

This section will provide the code for solving the smallest intersecting ball problem for m balls in n dimensions with log-exponential smoothing, set expansion, and weighted projections. Empirical testing of the various methods follows.

4.5.1 Log-Exponential Smoothing

When using the MM principle in the log-exponential smoothing method, each iterate x_{k+1} of the algorithm is found as the minimum of the surrogate $\mathcal{G}_p(x, x_k)$ (equation 3.4.2), which is itself minimized by Nesterov's accelerated gradient method.

Let us first write a function which minimizes a single iteration of this surrogate. We will pass to it some x_0 value, the set of target points `pts` (remember the distance majorization

surrogate uses projections to approximate distance to sets,) a stopping criterion `stop`, and the smoothing parameter `p`.

The loop in `sibNesterov` performs two tasks: (a) find the gradient of $\mathcal{G}_p$, and (b) compute the λ, γ and y terms needed to update x_{k+1}. We discuss here computing the gradient; updating the γ and λ terms is described in section 4.2.

Recall the gradient is given by

$$\nabla \mathcal{G}_p(x,k) = \sum_{i=1}^{m} \frac{\mathrm{e}^{\frac{g_i(x,k,p)}{p}}}{g_i(x,k,p)} \frac{x - P_{\Omega_i}(k)}{\sum_{j=1}^{m} \mathrm{e}^{\frac{g_j(x,k,p)}{p}}},$$

with

$$g_i(x,k,p) = \sqrt{\|x - P_{\Omega_i}(k)\|^2 + p^2}.$$

We compute a vector `g` whose i^{th} row holds $g_i(x,k,p)$ by first finding the distance from x to target points using a `repmat` matrix `X` and the column summation of pointwise squared elements. We include the p parameter before taking the root. Then `g=sqrt(sum((X-pts).^2,2)+p^2)` is an $m \times 1$ vector with $g_i(x,k,p)$ in its i^{th} row. Note that $\mathrm{e}^{g_i(x,k,p)}$ appears in both the numerator and denominator of the gradient. If we use `max(g)` to find the $\max\{g_i | i = 1,2,...,m\}$ and multiply the gradient by $\frac{\mathrm{e}^{-\texttt{max(g)}/p}}{\mathrm{e}^{-\texttt{max(g)}/p}} = 1$, each term $\mathrm{e}^{\frac{g_i(x,k,p)}{p}}$ in the summations can be equivalently expressed as

$$\mathrm{e}^{\frac{g_i(x,k,p) - \texttt{max(g)}}{p}},$$

and this bounds each exponential term by 1. Without this, as p becomes small the exponential terms exceed computational precision.

As the term $\mathrm{e}^{\frac{g_i - \max(\mathrm{g})}{p}}$ appears more than once, we create the variable `expGoverP`, a column vector whose rows hold the i^{th} such term. The variable `term1=expGoverP./sum(expGoverP` then holds

$$\frac{\mathrm{e}^{\frac{g_i(x,k,p) - \max(\mathrm{g})}{p}}}{\sum_{j=1}^{m} \mathrm{e}^{\frac{g_j(k,p)}{p}}}$$

in its i^{th} row. Now in each term of the larger summation which remains in our computation of the gradient, we are taking the product of two terms, $(x - P_{\Omega_i}(k))$ and $\texttt{term1}_i/g_i$, which lends itself to dot product multiplication, giving us the gradient as `gradient=(term1./g)'*(X-pts)`.

```
function [x] = sibNesterov(x0,pts,stop,p)
numPts=size(pts,1);
dim=size(pts,2);
x = x0;
```

```
y = x0;
gradient=ones(1,dim);
lambda=0;
while norm(gradient)>stop;
     X=repmat(x,numPts,1);
     g=sqrt(sum((X-pts).^2,2)+p^2);
     expGoverP = exp((g-max(g))/p);
     term1 = expGoverP./sum(expGoverP);
     gradient = (term1./ g)'*(X-pts);

     yNext = x - p/2*gradient;
     lambdaNext=(1+sqrt(1+4*lambda^2))/(2);
     gamma=(1-lambda)/lambdaNext;
     x = (1-gamma)*yNext + gamma*y;
     y=yNext;
     lambda=lambdaNext;
end
end
```

With the `sibNesterov` function ready to minimize a single surrogate, we are now ready for our main function `SIBLogSmooth`. Let us first discuss the parameters involved in this algorithm.

The log exponential function which we are minimizing best approximates the max distance function with small p, so it is tempting to use some infinitesimal p-value. However, the gradient of each surrogate is $2/p$-Lipschitz continuous, and as this increases, the convergence of the accelerated gradient method requires more iterations (see equation 2.3.1). Experiments suggest that starting with a relatively large p and decreasing it after each surrogate is minimized allows the first surrogates to converge quickly, which decreases the $\|x_1 - x^*\|^2$ term for the following surrogates, upon which convergence also depends.

Each time we minimize a surrogate, we pass a stopping criterion to the `sibNesterov` function. We cannot run an infinite number of iterations, so any solution is necessarily an approximation. The question here is: how accurately do we want to minimize the surrogate before updating x_{k+1}? Again, experiments suggest that less precise optimal solutions to the initial surrogates is made up for by requiring less iterations; after all, the surrogates are themselves approximations to the log-exponential smoothing function. For this reason, we start with a relatively high stopping criterion and, as with p, decrease it with each iteration.

For m balls in $\mathbb{R}^n$, this function takes as arguments an $m \times n$ matrix of centers, and an $m \times 1$ vector of corresponding radii. It assigns $x_0 = 0$ and uses an initial $p = 5$ and stopping criterion of `grad_stop`$= 0.5$. Constants $\sigma = 0.2$ and $\gamma = 0.3$ will act to decrease p and the stopping criterion. The main loop finds the projections of the current x_k, calls `SIBnesterov` to find x_{k+1}, and then scales the parameters back. This continues until $p < 10^{-6}$, which

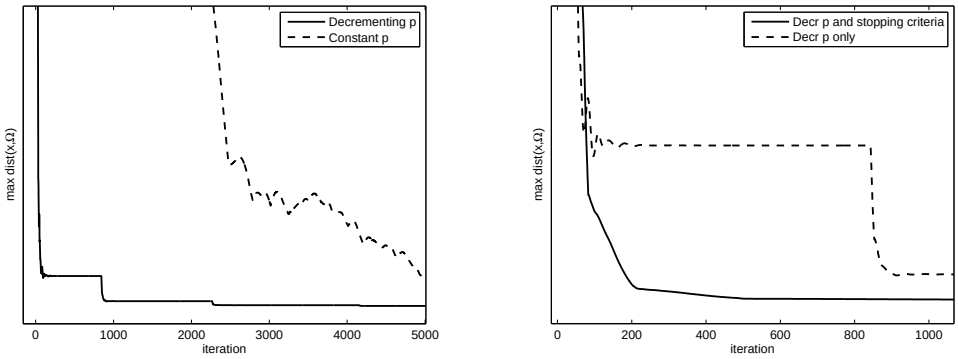


FIGURE 4.5.1: The log-exponential smoothing technique for 50 balls in $\mathbb{R}^{50}$. Performance is affected by choice of p parameter and stopping criteria for minimizing each surrogate. *Left:* Gradually decreasing p-values improves convergence (solid). The dashed curve attempts to solve using one constant p approximation. *Right:* Decreasing the stopping criteria at each iteration further improves performance (solid). The flat region of the (dashed) plot corresponds to a surrogate for some p-approximation being minimized unnecessarily accurately.

is 10 iterations of the MM algorithm. The resulting x is considered an optimal center. Its optimal radius is the max distance, found as in other functions with a `repmat` of projections.

```
function [x,optimalRadius] = SIBLogSmooth(SETS,RADII)
x  = zeros(1,size(SETS,2));
p=5;
grad_stop=0.5;
sigma = .2;
gamma = .3;
while p >1e-6
    pts=projectOntoBalls(x,SETS,RADII);
    [x]=sibNest1983(x,pts,grad_stop,p);
    p=sigma*p;
    grad_stop=gamma*grad_stop;
end
projections=projectOntoBalls(x,SETS,RADII);
X=repmat(x,size(SETS,1),1);
optimalRadius=max(sqrt(sum((X-projections).^2,2)));
end
```

4.5.2 Expanding Sets

The expanding sets algorithm increases the radii of all sets incrementally, at each step searching for the intersection. If the intersection is non-empty then we shrink the radii until it is empty, and decrease the step size. Otherwise, we increase the radii again. To implement this in *MATLAB*, our function `SIBexpand` will accept the m centers `SETS` and

radii `RADII` of target balls in the same matrix structure used in our other functions. The $m \times 1$ vector `altRADII` will store the inflated radii, and `step` is the amount by which we will expand the sets (We choose 20 in this function; an optimal initial step size would depend on the distance between sets.) We will determine inclusion in the intersection of expanded sets with the vector `maxDistToExpanded`.

We perform 50 iterations. In each iteration, we start an expansion `while` loop. In this loop we call a feasible point function and assign the convergent solution to our `x`, determine if `x` lies in the intersection of expanded sets, and then increase the `altRADII` of our expanding sets by `step`. We exit the loop only when `x` lies in the expanded intersection. We then begin a shrinkage `while` loop, which subtracts `step` from the radii of expanded sets and searches for a point in their intersection. When a feasible point function converges with positive distance to an expanded set, we exit the shrinkage loop. At this point we scale the expansion term `step` by 0.5 and begin our next iteration.

```
function [x,optimalRadius] = SIBexpand(SETS,RADII)
numSets = size(SETS,1);
dim = size(SETS,2);
x  = zeros(1,dim);
altRADII=RADII;
step=20;
maxDistToExpanded=999;
for ct=1:50
    while maxDistToExpanded > 0
        [x]=feasiblePoint(x,SETS,altRADII);
        X=repmat(x,numSets,1);
        altPROJ=projectOntoBalls(x,SETS,altRADII);
        maxDistToExpanded=max(sqrt(sum((X-altPROJ).^2,2)));
        altRADII=altRADII+step;
    end
    while maxDistToExpanded==0
        altRADII=altRADII-step;
        [x]=feasiblePoint(x,SETS,altRADII);
        X=repmat(x,numSets,1);
        altPROJ=projectOntoBalls(x,SETS,altRADII);
        maxDistToExpanded=max(sqrt(sum((X-altPROJ).^2,2)));
    end
    step=.5*step;
end
X=repmat(x,numSets,1);
PROJ=projectOntoBalls(x,SETS,RADII);
optimalRadius=max(sqrt(sum((X-PROJ).^2,2)));
end
```

The performance of this algorithm depends greatly on the stopping criterion of its feasible point function. Note that in implementing this algorithm, we assume that the intersection of our target sets is empty.

4.5.3 Weighted Projections

This function, as usual, accepts an $m \times n$ matrix of m target ball centers in $\mathbb{R}^n$, with corresponding radii in an $m \times 1$ vector. We start with $x_0 = 0$, and build a $1 \times m$ vector `weightList` whose i^{th} entry holds γ_i corresponding to the set Ω_i. The `donationTerm` is initialized to $1/m$; this is the amount by which γ_i decreases if Ω_i is not the most distant set from x. The `donationTerm` is decreased if the best-yet-attained objective value is not decreasing over some k iterations (in this case we choose $k = m$). Without knowledge of convergence conditions, the algorithm is set to run for $k = 5000$ iterations, with each performing the following:

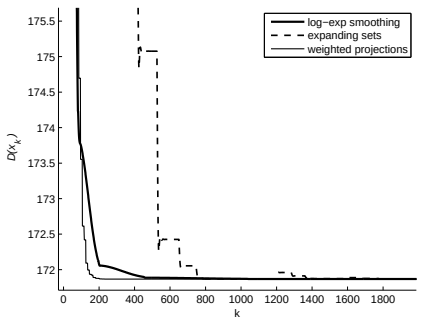
- Compute the vector `PROJECTIONS` to hold $P_{\Omega_i}(x_k)$ in its i^{th} entry.
- Set $x_{k+1} := \sum_i^m \gamma_i P_{\Omega_i}(x_k)$ with `x = weightList * PROJECTIONS;`
- Compute the max distance $\mathcal{D}(x_k)$ and find a most distant set `MDS`.
- Append $\max\{\mathcal{D}(x_k), \mathcal{D}(x_{k-1})\}$ into `fBest` to track progress. (This is why the loop index starts at q=2)
- Update γ weights.
- Begin next iteration.

```
function [x,optimalRadius] = SIBweightedProjections(SETS,RADII)
numSets=size(SETS,1);
dim=size(SETS,2);
x=zeros(1,dim);
weightList = 1/numSets * ones(1,numSets);
donationTerm = 1/numSets;
fBest=999*ones(1,5000);
for q=2:5000
    PROJECTIONS = projectOntoBalls(x,SETS,RADII);
    x = weightList*PROJECTIONS;
    PROJECTIONS = projectOntoBalls(x,SETS,RADII);
    X=repmat(x,numSets,1);
    dists=sqrt(sum((X-PROJECTIONS).^2,2));
    [maxDist MDS]=max(dists);
    fBest(q)=max(maxDist,fBest(q-1));
    if q > numSets
        if fBest(q-numSets)-fBest(q)<1e-3
            donationTerm=donationTerm*0.9;
```

```
        end
    end
    for r=1:numSets
        if dists(r) < maxDist
            if weightList(r) > donationTerm
                weightList(MDS) = weightList(MDS) + donationTerm;
                weightList(r) = weightList(r) - donationTerm;
            else
                weightList(MDS) = weightList(MDS) + weightList(r);
                weightList(r) = 0;
            end
        end
    end
end
PROJECTIONS = projectOntoBalls(x,SETS,RADII);
X=repmat(x,numSets,1);
optimalRadius=max(sqrt(sum((X-PROJECTIONS).^2,2)));
end
```

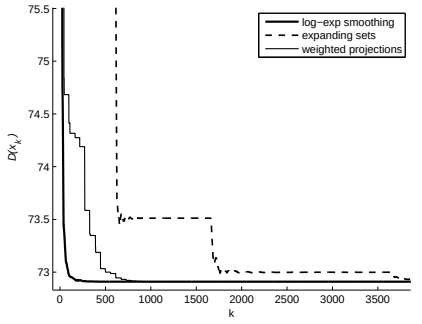
4.5.4 Performance Comparison

5 balls, 50 dimensions



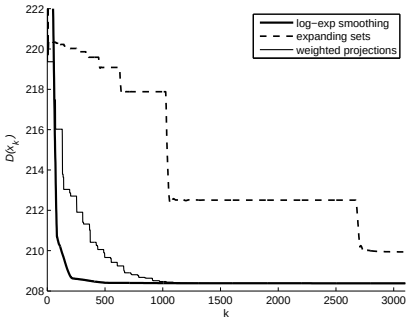
	Optimal Radius
log-exp smoothing	171.866577494
expanding sets	171.866577440
weights	171.866576752

50 balls, 5 dimensions



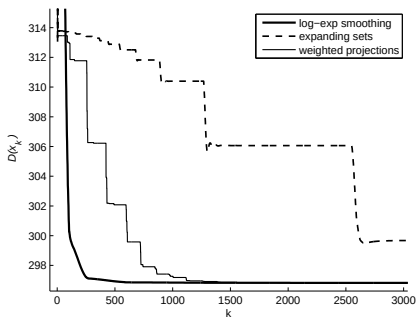
	Optimal Radius
log-exp smoothing	72.909753027
expanding sets	72.909756404
weights	72.909752439

50 balls, 50 dimensions



	Optimal Radius
log-exp smoothing	208.380382995
expanding sets	208.380432514
weights	208.380381564

100 balls, 100 dimensions



	Optimal Radius
log-exp smoothing	296.823572528
expanding sets	296.823731530
weights	296.823588427

FIGURE 4.5.2: Comparison of smallest intersecting balls algorithms in various scenarios. The expanding sets method utilized the accelerated gradient method to find a feasible point. Log-exponential smoothing approaches the optimal solution in the fewest iterations in every case, except with 5 sets in $\mathbb{R}^{50}$. The weighted projection algorithm (whose plot represents the best-yet-attained $\mathcal{D}(x_k)$ to reduce noise) found the minimal radius in every case, except that of 100 sets in 100 dimensions.

Bibliography

[1] N.T. An, D. Giles, N.M. Nam, R.B. Rector: Log-exponential smoothing techniques and Nesterov's accelerated gradient method for generalized Sylvester problems. *arXiv:1303.7247v5* (2015).

[2] A. Beck, M. Teboulle: A fast iterative shrinkage-thresholding algorithm for linear inverse problems. SIAM J. Imaging Sciences, **2**(1):183-202 (2009).

[3] S. Bubeck: ORF523: Nesterov's Accelerated Gradient Method. [https://blogs.princeton.edu/imabandit/orf523-the-complexities-of-optimization] (2013).

[4] D.P Bertsekas, A. Nedic, A.E. Ozdaglar: Convex Analysis and Optimization. Athena Scientific, Belmont (2003).

[5] E.C. Chi, H. Zhou, K. Lange: Distance majorization and its applications. Math. Program.,Ser. A, **146**: 409-436 (2014).

[6] G. Cimmino: Calcolo approssimato per soluzioni dei sistemi di equazioni lineari. La Ricerca Scientifica XVI Ser. II Anno IX(1), 326-333 (1938).

[7] H.E. Krogstad: Penalty and barrier methods- a summary. TMA 4180 Optimeringsteori, Norwegian University of Science and Technology, (2010).

[8] M. Lichman: *UCI Machine Learning Repository* [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[9] J. Mairal: Optimization with first-order surrogate functions. International Conference on Machine Learning, Jun 2013, Atlanta, USA. **28**: 783-791; JMLR Proceedings.

[10] B.S. Mordukhovich, N.M. Nam: An Easy Path to Convex Analysis. Morgan & Claypool, (2014).

[11] Y. Nesterov: A method of solving a convex programming problem with convergence rate O(1/k2). Soviet Mathematics Doklady, **27**(2):372-376 (1983).

[12] Y. Nesterov: Introductory Lectures on Convex Programming. Vol 1. Springer US, 1998.

[13] Y. Nesterov: Smooth minimization of non-smooth functions. *Math.Program., Ser. A* **103** , 127-152 (2005).

[14] J.M. Ortega, W.C. Rheinboldt: Iterative Solutions of Nonlinear Equations in Several Variables. Academic, New York (1970).

[15] X. Zhai: Two problems in convex conic optimization. Master's thesis, National University of Singapore (2007).

[16] G. Zhou, K. Toh, J. Sun Efficient algorithms for the smallest enclosing ball problem. Journal of Computational Optimization and Applications, **30**(2):147-160 (2005).