PRIMAL-DUAL CUTTING-PLANE METHOD FOR DISTRIBUTED DESIGN

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Alessandro Magnani
December 2006

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Stephen P. Boyd)
Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Mark A. Horowitz)

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

(Sanjay Lall)

Approved for the University Committee on Graduate Studies.

Abstract

This thesis studies aspects of convex optimization for design. In particular we consider the problem of designing a complex system formed from a number of subsystems. Each subsystem represents a *soft design*, *i.e.*, a system that is not completely specified, or can be configured in different ways. The design problem is to coordinate the subsystems in order to achieve a given global specification. There are two major parts of this thesis.

The first part introduces a new framework and algorithm that, under certain conditions, optimally solves the design problem. We call this new algorithm the *primal-dual cutting-plane method*. We describe the properties of this algorithm and we demonstrate with examples the generality of this framework.

The second part introduces new algorithms that can be used to generate high-level models of subsystems for use in this design framework. In particular we address the problem of fitting data (perhaps obtained from simulations or complex models) with piecewise-linear or polynomial convex functions. Moreover we show other techniques that allow us to generate high-level models for use in this framework. We will also describe other possible applications of these algorithms.

Acknowledgments

First and foremost I would like to thank my advisor Professor Stephen Boyd, for his excellent guidance and teaching. His wealth of ideas, clarity of thought, enthusiasm and energy have made working with him an exceptional experience for me.

I would like to thank my academic advisor Professor Mark Horowitz and Professor Sanjay Lall for giving me valuable feedback and being on my reading committee. I thank Professor Joseph Kahn for acting as the chairman of my defense committee.

My special thanks go to a dynamic research group whose members are always fun to be with: Alexandre d'Aspremont, Jon Dattorro, Maryam Fazel, Arpita Ghosh, Michael Grant, Haitham Hindi, Siddharth Joshi, Seung Jean Kim, Kwangmoo Koh, Robert Lorenz, Almir Mutapcic, Sikandar Samar, Jöelle Skaf, Jun Sun, Argyris Zymnis and many others; I am very grateful to Denise Murphy for her superb administrative assistance.

I wish to acknowledge the Stanford Graduate Fellowship program for providing funding for my Ph.D. study and research.

I dedicate this thesis to my parents and my wife.

Contents

List of Figures

Chapter 1

Introduction and overview

In this thesis we consider the problem of designing a complex system formed by a number of subsystems. Each subsystem can design itself given specifications and needs to satisfy a number of constraints imposed by the system. The goal is to find a design for each subsystem so that the constraints are satisfied and the system meets some global specifications.

In the first part of the thesis we introduce a new framework and a new algorithm to solve this design problem. In the second part of the thesis we address the problem of creating models for the subsystems starting from data or equations describing the subsystems.

1.1 The design problem

We consider a generic system formed by k subsystems. Each subsystem represents a *soft design*, *i.e.*, a system that is not completely specified, or can be configured in different ways. For example, a subsystem might represent a family of amplifiers, each with a different gain, power, area, noise power, input capacitance, and so on. The details of each design are private to the subsystem; only the interface variables are available outside the subsystem. For our amplifier example, the specific circuit design is private to the subsystem; its specifications, that matter to the surrounding system, are exported. For subsystem i we call $x_i \in \mathbf{R}^{n_i}$ the vector of interface variables and $\mathcal{X}_i$ the set of feasible interface variables. In other words $x_i \in \mathcal{X}_i$ if and only if subsystem i includes a design with interface variables x_i.

The subsystems need to satisfy a set of constraints. For example, some of the constraints come from the fact that the subsystems share some common resources, or have electrical connections or need to meet some global specifications. We can assume without loss of generality that all the constraints are linear and can therefore be written as

$$\sum_{i=1}^{k} F_i x_i = g.$$

We will show later why this assumption is not restrictive.

The design problem is to find interface variables that are feasible for all the subsystems and satisfy the constraints $\sum_{i=1}^{k} F_i x_i = g$. We can therefore write the design problem as

$$\begin{array}{ll} \text{find} & x \\ \text{subject to} & x_i \in \mathcal{X}_i, \quad i = 1, \ldots, k \\ & \sum_{i=1}^{k} F_i x_i = g. \end{array} \tag{1.1}$$

This problem can describe a large variety of design problems and it is in general hard to solve. We will therefore only consider the situation where the sets $\mathcal{X}_i$ are convex. Practically it has been shown that many subsystems have set $\mathcal{X}_i$ convex at least after a change of coordinates [dMH02, HBL98, BK05]. Under this assumption the problem is a *convex problem* [BV04] and can be solved efficiently. For example, if the sets $\mathcal{X}_i$ are polyhedral the problem becomes a *linear program* [BV04].

In this thesis we are interested in the situation where we are not given a complete description of the sets $\mathcal{X}_i$ but we can only get information on them using an *oracle.* An oracle is a simple way of getting information about $\mathcal{X}_i$. For example, an oracle can be queried with a point $\hat{x}$ and it checks if the point belongs to the set $\mathcal{X}_i$ or not. Therefore the oracle gives localization information regarding the set $\mathcal{X}$. In the framework we present in this thesis, each subsystem has two oracles that we call primal and dual oracle. The primal oracles checks if a point $\hat{x}$ belongs to the set $\mathcal{X}$. If the point doesn't belong to the set, it returns a halfspace that contains $\mathcal{X}$ and not the point $\hat{x}$. The dual oracle checks if a given halfspace contains the set $\mathcal{X}$. If the halfspace doesn't contain $\mathcal{X}$, the dual oracle returns a point in the set that is not in the halfspace.

The goal is to coordinate between subsystems to find a solution to problem (1.1), by only using the information provided by the oracles of the subsystems. We call this design problem *distributed design.*

In practice this problem is usually solved by a manual iterative process where at each iteration a new set of interface variables for each subsystem is created and then tested to see if it achieves the given global specification. If some of the constraints cannot be met, a new set of specifications is derived and this process is repeated until a feasible design is found. Usually the new set of specifications is obtained by the designer using his experience and the information gathered while designing each subsystem.

In chapter 2 we introduce a framework and an algorithm that allows us to solve the distributed design problem efficiently. In particular, we will precisely define the oracles and we will introduce a new algorithm called *primal-dual cutting-plane method* that either finds a globally optimal design or certifies the infeasibility of the design problem.

One reason for doing distributed design is the ability to solve large design problems [BT89]. In fact, by splitting the system in smaller subsystems we can distribute the computation cost and solve larger problems. Another reason is the ability to encapsulate subsystems. In fact, in this framework, as long as a subsystem has the specified interface, it can be swapped or re-used in many different designs. We can also easily design a system where the internal models of each subsystems are different. The goal of a designer in this framework is therefore to create soft designs that provides the correct interface for each of the subsystems.

1.2 Model creation and convex data fitting

In the second part of this thesis, we study the problem of creating suitablemodels for the subsystems to be used in this framework. We will first address the problem of creating the oracles of the subsystems if we are given a mathematical description of the subsystem. We will also consider the problem of estimating the set $\mathcal{X}$ if we are only given a set of points in the set $\mathcal{X}$, possibly on its boundary. This situation can occur if, for example, we are gathering information on a subsystem by performing simulations. Each simulation result corresponds to a design and therefore a point in $\mathcal{X}$. Given these points, we would like to fit them using a family of functions, in order to have an approximation of the set $\mathcal{X}$. We can then use this fit as a description of the subsystem's feasible set. We will describe exactly what we mean by fitting data points.

Chapter 3 shows how given a mathematical representation of the subsystem we can create the interface for the subsystems (oracles). In particular we consider the case

where the design of a subsystem is carried out using a convex optimization problem. In this case we will show the relationship that exists between the dual variables of the optimization problem [BV04] and the subsystem's interface. Moreover we will show how to create the interface for a subsystem, in the case where the feasible set is described either as the sublevel set or the epigraph of some given function.

In chapter 4 and chapter 5, we introduce two new algorithms that can be used to create models for subsystems from given data perhaps obtained from simulations or complex models. In particular the two algorithms allow us to fit data with a convex function. The obtained fit together with the techniques of chapter 3, can then be used to create a subsystem model.

In chapter 4 we present a new algorithm that fit data using convex piecewise-linear functions. In particular given data

$$(u_1, y_1), \ldots, (u_m, y_m) \in \mathbf{R}^n \times \mathbf{R}$$

we would like to fit them with a convex piecewise-linear function $f : \mathbf{R}^n \to \mathbf{R}$ from some set $\mathcal{F}$ of candidate functions. The problem is

$$\begin{array}{ll} \text{minimize} & J(f) = \sum_{i=1}^m (f(u_i) - y_i)^2 \\ \text{subject to} & f \in \mathcal{F}, \end{array}$$

with variable f. We will consider the case where f is given by

$$f(x) = \max\{a_1^T x + b_1, \ldots, a_k^T x + b_k\},$$

and we will also consider a more general form of a convex piecewise-linear function. We will introduce new heuristic algorithms for this problem.

In chapter 5 we introduce an algorithm to fit data using convex polynomials. In particular given data

$$(u_1, y_1), \quad \ldots, \quad (u_m, y_m)$$

we want to solve the problem

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^m (f(u_i) - y_i)^2 \\ \text{subject to} & f \quad \text{is convex}, \end{array}$$

where f is a polynomial with the form

$$f = c_1 p_1 + \cdots + c_w p_w,$$

and the p_i are given polynomials. We will also introduce a heuristic to approximate a set of data using the sublevel set of polynomials. In particular the problem is

$$\begin{array}{ll} \text{minimize} & \text{volume}(P) \\ \text{subject to} & P = \{x \mid f(x) \leq 1\} \\ & u_i \in P \quad i = 1, \ldots, m, \\ & P \text{ is convex.} \end{array}$$

In chapters 4, and 5, we will also show other possible applications of these algorithms and we will demonstrate them with examples. In particular these algorithms can be also used to do LP modeling, to simplify convex functions and to create equations compatible with geometric programming.

1.3 Contribution of this thesis

In Chapter 2, the framework the the algorithm are new. The convergence result and the analytical cutting-plane method are based on the standard results [Ye97, BV04].

In Chapter 4, the algorithms presented are new. Chapter 4 is based on the paper

> A. Magnani, and S. Boyd. Convex Piecewise-Linear Fitting. Under review in *Optimization and Engineering*.

In Chapter 5, the algorithms presented are new. They were originally presented in

> A. Magnani, S. Lall and S. Boyd. Tractable fitting with convex polynomials via sum-of-squares . In proceedings of *Conference on Decision and Controls*, 2005.

Chapter 2

Primal-dual cutting-plane method

2.1 Introduction

In this chapter we first describe a new framework (§2.2) for convex distributed design. We define the oracles which are the only interface between subsystems and we define the form of the constraints between subsystems. We also state all the assumptions that need to be satisfied by the subsystems and the system. We also define the design problem that we would like to solve in §2.3. We then describe a class of algorithms (cutting-plane methods) that allow us to solve convex feasibility problems. We then introduce (§2.4) the primal-dual cutting-plane method that can be viewed as a improvements of the above mentioned algorithms, that can efficiently solve the distributed design problem. Finally we show two examples of how this framework can be used to solve design problems in practice. We will consider two examples from digital and analog circuit design but the framework can also be used in other fields as well.

2.1.1 Previous work

For a recent survey of relevant work on automated design, see [GME05]. In the *flat methodology*, the design of the entire system is carried out at the same time, designing each component and subsystem in one large optimization problem. This can be done by repeatedly simulating the system [KPRC99] or by deriving convex approximate constraints, which are then solved with a convex solver [dMH02]. Another approach is called the *top-down constrained driven methodology* [VCBS04, CCC+97, DGS+96, DDNAV04]. These methods traverse the design from the highest level design specifications down to

the design of the lowest level subsystems in the system hierarchy. At each step a set of specifications is chosen for the lower level subsystems and the optimization is then performed at the lower level using this set of specifications. If this fails, the algorithm must relax the specifications from the previous step and try again. Designs using this approach have been reporting using simulated annealing [AWV05] to perform the optimization.

The next approach is the so called bottom-up with top-down constrained driven methodology [HS96, SGA03, BJV03, BNV05]. This methodology is similar to the previous one but in this one the top down methodology follows a first step in which the feasibility problem of deciding the feasibility of the design is addressed in a bottom up fashion. In this methodology each subsystem in the first phase of the design can be queried with a set of specifications and returns true if the design is feasible and false vice versa. Using a bottom up algorithm it is possible to find a feasible design for each subsystem. This technique is similar to our proposed approach, except that we will require a cutting-plane to be returned when a set of specifications in infeasible. (This allows us to drive the overall design first to feasibility, and the optimality.) Yet another methodology we mention is the so-called multi-objective bottom-up technique [EMG05], which proceeds in a bottom up fashion using a genetic algorithm. At each step the design of a subsystem is chosen from a library of Pareto optimal designs.

Multidisciplinary design optimization (MDO) is an enabling methodology for the design of complex system, the physics of which involve coupling between various interacting subsystems. The underlying focus of MDO methodology is to develop formal procedure for exploiting the coupling in the problem at every stage of the design process. A review of the state of the art in MDO can be found in [SH97]. The earlier research in MDO has focused on system optimization approaches, also known as multidisciplinary feasible methods [KSD$^+$97]. These approaches carry out the optimization in a centralized way using a single optimizer. More recently distributed optimization architecture has been introduced. In particular, concurrent subspace optimization [RG94, WRBB96] and collaborative optimization [AL00, BK97] are the most promising ones. The algorithms are all heuristic and they don't guarantee convergence to a global optimum.

Many modeling and optimization techniques have been developed for subsystem. One way to internally optimize the subsystem is using geometric programming [DPZ67]. Models used in this approach can be automatically generated [DGS05] or manually obtained [dMH02]. Support Vector Machine and kernel techniques can be used to

obtained models of the subsystem that return the feasibility or infeasibility of the design [BJV03, PAOS03]. Other possible modeling techniques are extensive simulation [KPRC99], (boosted) neural networks [WV03, MG05], genetic programming [MG05], model-order reduction [Roy03], data mining [LSRC02], and Kriging [MG05].

2.2 General framework

A system in our framework is a set of subsystems together with a set of constraints that the subsystems have to satisfy. The constraints might represent physical connections between subsystems, some global performance measure that the system needs to achieve, a limit on a common resource shared by subsystems, or any other specification that allows us to completely describe the interaction between subsystems.

Each subsystem represents possibly many different designs and therefore we say that it is a soft design. A given design for subsystem i can be completely described with a vector y_i that we call the vectors of internal variables. In other words the internal variables are all that is needed to completely specify a given design. If, for example, the subsystem represents an amplifier the vector of internal variables might contain the size of all the components in the circuit.

For every design of subsystem i there is an associated vector x_i that we call the vector of interface variables. The vector x_i contains all the variables that are visible from the other subsystems of the system. Therefore all the constraints of the system are expressed only in term of the interface variables. For the amplifier example the vector x_i might contain the gain, power consumption, area of the amplifier obtained by the design y_i.

Figure 2.1 shows a system formed by four subsystems. The lines between subsystems represents the constraints that are expressed only in term of the interface variables x_i. For simplicity we define $n = \sum_{i=1}^{k}$ and $x = (x_1, \ldots, x_k)$ to be respectively the number and vector of all interface variables in the system.

2.2.1 Subsystem characterization

If we consider subsystem i, for given interface variables x_i there might not be a vector of internal variables y_i associated with x_i. In this case we say that x_i is infeasible for the subsystem i. In the amplifier example, a set of infeasible interface variables correspond to some area, noise and power of the amplifier that cannot be achieved by any design. If

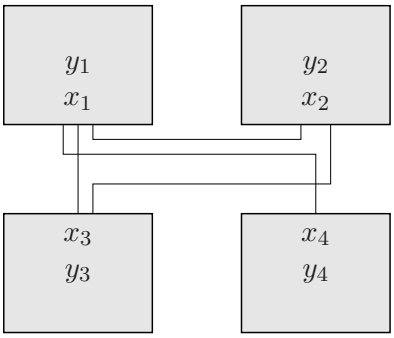

Figure 2.1: System with four subsystems. The variables x_i are the interface variables, the y_i are the internal variables. Constraints are represented with lines between subsystems.

instead there is a vector y_i associated with x_i, we say that x_i is feasible for the subsystem.

We call the set of all interface variables that are feasible for the subsystem i, $\mathcal{X}_i$. In other words if $x_i \in \mathcal{X}_i$, there is a design of the subsystem y_i that corresponds to x_i and if $x_i \notin \mathcal{X}_i$ there is no y_i that corresponds to x_i.

Without loss of generality we will assume for the rest of the paper that for any subsystem the design $x = 0 \in \mathcal{X}$. If this is not the case we can always change coordinates. We make this assumption only to simplify the mathematical notation.

We will refer to a subsystem by using the set of feasible design. We will therefore say the subsystem $\mathcal{X}_i$ to refer to system i with a feasible set $\mathcal{X}_i$.

2.2.2 System's constraints

As described before, a system is formed by a set of subsystems and a set of constraints on their interface variables. In this framework we make the assumption that all the constraints can be written as

$$Fx = g,$$

where $F \in \mathbf{R}^{p \times n}$ and $g \in \mathbf{R}^p$. In other words the constraints are affine in the interface variables.

This might seem like a restrictive assumption but as we will show with examples, this is not the case. In fact, we can handle nonlinear constraints by having a new subsystem whose only purpose is to enforce nonlinear constraints.

If, for example, we have the following nonlinear constraints between interface variables x_1 and x_2

$$\|x_1 - x_2\| \leq 1,$$

we can add to the system a new subsystem with feasible set $\mathcal{X}_3 = \{x \mid \|x\| \leq 1\}$ and impose the linear constraint $x_1 - x_2 = x_3$. We will show more examples later in this thesis.

2.2.3 Convexity and closure assumptions

The framework we just described is extremely general, and in general, the associated design problem, is hard to solve. We therefore make the assumption that the sets $\mathcal{X}_i$ are convex. Practically it has been shown that many subsystems have set $\mathcal{X}_i$ convex at least after a change of coordinates [dMH02, HBL98, BK05]. Moreover if the convexity assumption of the set $\mathcal{X}$ is not satisfied we can restrict $\mathcal{X}$ to a convex subset and still use this framework. This might be the case if for example we already know that the final design for a given subsystem will lie in a smaller set of all the feasible designs.

We also assume that the set $\mathcal{X}$ is closed. This is from a practical point of view not important and can be neglected.

2.2.4 The subsystem's interface

As we said before, each subsystem is described using interface variables $x \in \mathbf{R}^n$ and the convex set of feasible design $\mathcal{X}$. To coordinate between subsystems we need to be able to get information about the sets $\mathcal{X}$. In this framework, the only way to get information about the feasibility set of a subsystem is through oracles. An oracle is simply an interface of the subsystem that once queried returns information on the feasibility set. Each subsystem needs to have what we call a *primal oracle* and a *dual oracle*.

In the next sections we will give formal definitions of the oracles. We first introduce some notation.

2.2.5 Cutting-plane

Given a set $\mathcal{X}$, with $0 \in \mathcal{X}$, and a vector x° we say that x° is a *cutting-plane* of $\mathcal{X}$ if $x^{\circ T} x < 1$ for all $x \in \mathcal{X}$. In other words a cutting-plane defines a halfspace $\{x \mid x^{\circ T} x < 1\}$ that contains $\mathcal{X}$ and a halfspace $\{x \mid x^{\circ T} x \geq 1\}$ that doesn't contain it.

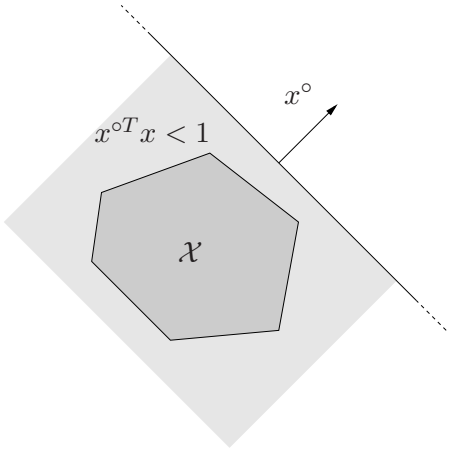

Figure 2.2: Cutting-plane example: x° is a cutting-plane of $\mathcal{X}$.

If the set $\mathcal{X}$ is closed and convex for any point $y \notin \mathcal{X}$ there is cutting-plane x° of $\mathcal{X}$ such that $y^T x^\circ = 1$.

Figure 2.2 shows a cutting-plane of the set $\mathcal{X}$ and the shaded area represents the set $\{x \mid x^{\circ T}x < 1\}$.

2.2.6 Polar set

Given a set $\mathcal{X}$, with $0 \in \mathcal{X}$, we define the *polar* set $\mathcal{X}^\circ$ as

$$\mathcal{X}^\circ = \{x^\circ \mid x^{\circ T}x < 1 \text{ for all } x \in \mathcal{X}\}.$$

In other words, the polar set is the set of all cutting-plane of the set $\mathcal{X}$. It's easy to show that the polar set of a set is always convex.

2.2.7 Primal oracle

The primal oracle allows us to check if a given point belongs to the feasibility set. If the point doesn't belong to the feasibility set the primal oracle returns a cutting-plane that separates the point from the set $\mathcal{X}$. We now define in more detail the primal oracle.

Given a point $\hat{x}$ we can query the primal oracle of a subsystem. If $\hat{x} \in \mathcal{X}$ the primal oracle returns true and if $\hat{x} \notin \mathcal{X}$, it returns a vector $x^circ \in \mathcal{X}^\circ$ such that $x^{\circ T}\hat{x} > 1$. In other words if the specification $\hat{x}$ is infeasible for the subsystem the oracle

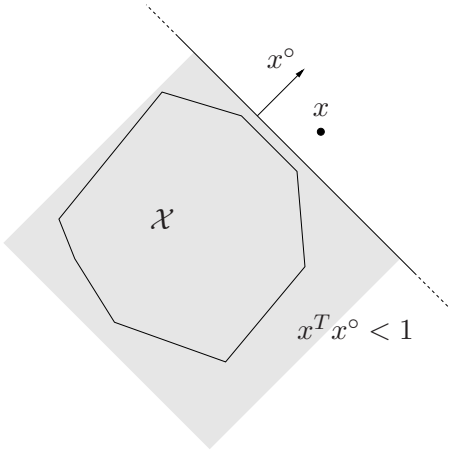

Figure 2.3: Primal oracle: primal oracle of the subsystem $\mathcal{X}$ returns cutting-plane x° when queried with the point x.

returns a cutting-plane which separates $\mathcal{X}$ from the point $\hat{x}$. Therefore we can interpret x° as a certificate of the infeasibility of $\hat{x}$. The shaded area represents the halfspace $\{x \mid x^{\circ T} x < 1\}$, that contains the set $\mathcal{X}$.

Figure 2.3 shows the cutting-plane x° returned by the primal oracle of the subsystem $\mathcal{X}$ when queried with the point x.

2.2.8 Dual oracle

The dual oracle checks if a given point belongs to the polar of the feasible set. If the point doesn't belong to the polar set then it returns an hyperplane that separates the point from the polar set. We now define in more detail the dual oracle.

Given a vector x° we can query the dual oracle of a subsystem. The subsystem returns true if $x^\circ \in \mathcal{X}^\circ$ otherwise it returns $\hat{x} \in \mathcal{X}$ such that $x^{\circ T}\hat{x} \geq 1$. The point $\hat{x}$ is a certificate that x° is not a cutting-plane for $\mathcal{X}$. Figure 2.4 shows the point x returned by the dual oracle of system $\mathcal{X}$ when queried with a vector x°. Clearly x° is not a cutting-plane for $\mathcal{X}$ since the halfspace $x \mid x^{\circ T} x < 1$ does not contain $\mathcal{X}$.

We can also interpret the dual oracle as simply a primal oracle for the polar of $\mathcal{X}$.

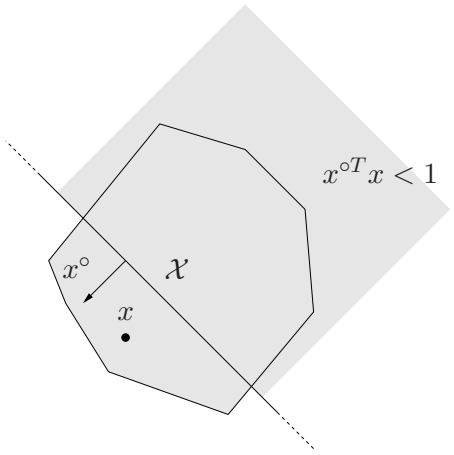

Figure 2.4: Dual oracle: the dual oracle of system $\mathcal{X}$ returned point x when queried with cutting-plane x°.

2.3 The design problem

The problem we want to solve for a given system is to find a feasible specific design for each subsystems so that all the constraints between them are satisfied. In other words we would like to find a specific configuration of the subsystem that yield a feasible system. The problem is then, given a system with constraints described by F and g, to find x so that the constraints are satisfied, and each x_i is feasible. If we define $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_k$ to be the set of all possible x such that x_i for $i = 1, \ldots, k$ is feasible, we can rewrite the problem as

$$\begin{array}{ll} \text{find} & x \\ \text{subject to} & x \in \mathcal{X}, \\ & Fx = g. \end{array} \tag{2.1}$$

We call problem (2.1) the primal feasible problem. We say that we can solve the design problem if we either find a feasible solution of problem (2.1) or if we can prove that the problem is infeasible. In the next section we will show how we can certify infeasibility of (2.1) by introducing the so called dual problem.

2.3.1 Certificate of infeasibility

It is easy to show that if we can find a solution to problem

$$\begin{array}{ll} \text{find} & \nu \\ \text{subject to} & F^T\nu \in \mathcal{X}^\circ, \\ & g^T\nu \geq 1. \end{array} \tag{2.2}$$

the design problem is infeasible and therefore a solution to problem (2.2) is a certificate of infeasibility for problem (2.1). We call this problem the dual feasibility problem.

In fact by contradiction suppose that ν is a solution of problem (2.2) and $x \in \mathcal{X}$ is such that $Fx = g$ we have that $\nu^T Fx < 1$ and therefore $\nu^T g < 1$ but this is a contradiction since $\nu^T g \geq 1$. In general if problem (2.1) is infeasible is not always possible to find a feasible solution to problem (2.2).

If $\mathbf{int}\,\mathcal{X} \neq \emptyset$ problem (2.1) and problem (2.2) are *strong alternatives*, *i.e.*, problem (2.1) is feasible if and only if problem (2.2) is infeasible and vice versa [BV04].

Under this assumption the design problem is either to find a solution to problem (2.1) or a solution to problem (2.2).

2.3.2 Reformulation of the design problem

It is possible to rewrite problem (2.1) and problem (2.2) by using the sets $\mathcal{X}_i$ and $\mathcal{X}_i^\circ$ explicitly. This play an important role since the oracles return information on the individual sets $\mathcal{X}_i$ and not $\mathcal{X}$.

We define $F_1, \ldots, F_k$ so that $Fx = F_1x_1 + \cdots + F_kx_k$. We can write problem (2.1) as

$$\begin{array}{ll} \text{find} & x \\ \text{subject to} & x_i \in \mathcal{X}_i, \quad i = 1, \ldots, k \\ & \sum_{i=1}^k F_i x_i = g. \end{array} \tag{2.3}$$

and problem (2.2) as

$$\begin{array}{ll} \text{find} & \nu,\ \lambda \\ \text{subject to} & \lambda_i^{-1} F_i^T \nu \in \mathcal{X}_i^\circ, \quad i = 1, \ldots, k \\ & \mathbf{1}^T \lambda = 1 \\ & \lambda \succ 0 \\ & g^T \nu \geq 1. \end{array} \tag{2.4}$$

It's easy to see that if problem (2.4) is feasible then problem (2.3) is infeasible. In fact suppose $\lambda,\ \nu$ is a solutions of problem (2.4) and x is solution of problem (2.3) we have

$$\begin{aligned} \lambda_i^{-1} \nu^T F_i x_i < 1 \quad &\Rightarrow \quad \nu^T F_i x_i < \lambda_i \Rightarrow \\ \nu^T \sum_{i=1}^{k} F_i x_i < \mathbf{1}^T \lambda \quad &\Rightarrow \quad \nu^T g < 1. \end{aligned}$$

This is a contradiction since we also should have $\nu^T g \geq 1$.

It's also possible to show (see [BV04]) that if $\mathbf{int}\,\mathcal{X} \neq \emptyset$, these two problems are strong alternatives. In the next section we will show how to simultaneously look for a solution of either problem (2.3) or problem (2.4).

2.3.3 Extensions

The design problem we introduce is a *feasibility problem i.e.*, it only involves finding a feasible design. In practice we are usually interested in a design that minimizes some given quantity, for example noise or power. We might also be interested in a tradeoff analysis to see how two or more parameters specification of the design tradeoff. It's easy, once the feasibility design problem can be solved, to solve both an optimization problem or a tradeoff analysis by solving a sequence of feasibility problems [BV04]. For simplicity in this thesis we will only consider the feasibility problem.

2.4 Solving the design problem

There are multiple ways of solving problem (2.1) when the set $\mathcal{X}$ is convex. We first describe a simple *cutting-plane algorithm* [Kel60] that allows us to solve problem (2.1) if

there exists a feasible design. Since this simple algorithm doesn't guarantee convergence in the case of an infeasible system we introduce a new algorithm called *primal-dual cutting-plane method* (PDCPM) that guarantees to either find a feasible design or to certify infeasibility by finding a solutions of problem (2.2).

2.4.1 Cutting-plane methods

Cutting-plane methods form a category of effective algorithms for solving nondifferentiable convex optimization problems (see, *e.g.*, [Ber99, BV04, MGV98, GV99]). They can be regarded as a *localization methods* that localize a feasible point to a polyhedron (the localization set). At each iteration a new point in the the localization set is computed (centering step) and based on the information returned by the primal oracle queried at that point, the localization set is narrowed down. The classical centering methods that have been suggested include the center of gravity method of Levin [Lev65], the largest sphere method of Elzinga and Moore [EM73], the ellipsoid method of Yudin and Nemirovsky [NY83], the maximum volume ellipsoid method of Tarasov, Khachiyan, and Erlich [KT93, TKE88], and the method of volumetric centers of Vaidya [Vai89], among others.

In this paper we focus on analytic center cutting-plane method (ACCPM) where the centering method uses the analytic center of the localization set. This method was first proposed by Goffin and Vial [GV90] and then continued in [GHV92, GHVZ93].

Analytic center

Let $\mathcal{P} = \{x \mid Ax \leq b,\ Fx = g\} = \{x \mid a_i^T x \leq b_i,\ i = 1, \ldots, m,\ Fx = g\}$ be a bounded polyhedron, with nonempty interior, defined through inequalities and equality constraints. Then its analytic center is defined as the unique optimal point of

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^m \log\left(b_i - a_i^T x\right) \\ \text{subject to} & Fx = g \end{array}$$

which can be computed very efficiently using Newton's method; see, *e.g.*, [Ye97, BV04].

2.4.2 The ACCPM

The analytic center cutting-plane method starts with an initial polyhedron which includes any possible solution of (2.3). At each iteration we narrow the polyhedron so that the polyhedron still contains all solutions of the problem. We then stop as soon as we find a feasible solution of our problem.

We start with the polyhedron $\mathcal{P}^{(0)} = \{x \mid A^{(0)}x \leq b^{(0)}, Fx = g\}$ that contains any possible feasible point $x^\star$. For example, we can choose

$$A^{(0)} = \begin{bmatrix} I \\ -I \end{bmatrix} \quad b^{(0)} = \begin{bmatrix} \overline{x} \\ -\underline{x} \end{bmatrix}$$

where $\underline{x}$ and $\overline{x}$ are known lower and upper bounds of the variable x. This is usually the case in any design problem because there is always a specified range in which the design variables should lie. Notice that all the points in the localization set are such that $Fx = g$ and therefore they satisfy the system's constraints. We therefore just need to find x in $\mathcal{X}$.

The ACCPM can then be outlined as follows:

given $\mathcal{P}^{(0)}$
$j := 0$.
repeat

- Compute the analytic center $x^{(j)}$ of $\mathcal{P}^{(j)}$.
- Query each subsystem at the point $x_i^{(j)}$ using primal oracle.
- If $x_{[i]}^\circ$ for $i = 1, \ldots, t$ are the cutting-planes returned, x must lie in the polyhedron

$$\mathcal{H}^{(j)} = \{x \mid x^{\circ T} x_{[i]} < 1, \text{ for } i = 1, \ldots, t\}.$$

- Form the polyhedron $\mathcal{P}^{(j+1)} = \mathcal{P}^{(j)} \cap \mathcal{H}^{(j)}$
- If no cutting-plane is returned, quit; else, $j := j + 1$.

It can be shown that if the problem is feasible and $\textbf{relint}\{x \mid Fx = g,\ x \in \mathcal{X}\}$ is not empty, then the algorithm finds a solution in a finite number of steps. For computational details and convergence analysis of ACCPM, see [Ye97, GV99] and references therein.

The problem with this algorithm is that if the problem is infeasible there is no guarantee of convergence. For this reason we introduce a primal-dual ACCPM that either finds a feasible point or produces a certificate of infeasibility.

2.4.3 Primal-dual ACCPM

We describe a generalized version of ACCPM that we call primal-dual ACCPM. This algorithm tries to either localize a feasible point of problem (2.3) or a feasible point of (2.4). We use two polyhedra, $\mathcal{P}$ which includes any possible solution of (2.3) and $\mathcal{P}^\circ$ which includes any possible solution of (2.4). At each iteration we shrink one or the other of these two polyhedra until we find a feasible point for one of them. To shrink the localization polyhedra we use the information returned by the primal and dual oracles of each subsystems.

Before describing the ACCPM in detail we give some intuition on what the algorithm does and how it shrinks the two polyhedra. Suppose that we query a subsystem at the point x_i using the primal oracle. If the point is not feasible for the subsystem $\mathcal{X}_i$ then the cutting-plane x° can be used to shrink the polyhedron $\mathcal{P}$ since we now know that a solution should satisfy $x_i^T x^\circ < 1$. Vice versa if the point x_i is feasible for system $\mathcal{X}_i$, we can shrink the polyhedron $\mathcal{P}^\circ$ because we know that any cutting-plane x° of $\mathcal{X}_i$ should satisfy $x_i^T x^\circ < 1$ and therefore we need to have $x_i^T F_i^T \nu < \lambda$. On the other hand if we query subsystem i with x° using the dual oracle and the dual oracle states that x° is a cutting-plane for subsystem i, we can shrink set $\mathcal{P}$ because we need to have $x_i^T x^\circ < 1$. If instead the dual oracle returns a point $x_i \in \mathcal{X}_i$ then we can shrink $\mathcal{P}^\circ$ because we need to have $x_i^T F_i^T \nu < \lambda$.

This shows that by querying each subsystem with either the primal oracle or dual oracle we can either shrink the polyhedron $\mathcal{P}$ or $\mathcal{P}^\circ$. As before after shrinking each polyhedron we use the analytical center to find two new points in $\mathcal{P}$ and $\mathcal{P}^\circ$ that we use to query the primal and dual oracles of each subsystem at the next iteration.

The first polyhedron is defined as before so that it includes any possible solution of the problem (2.3) and it is consistent with the system constraints. The second bounded polyhedron contains any possible solution of (2.4) can be defined as

$$\mathcal{P}^{\circ(0)} = \{(\nu,\ \lambda) \mid C^{(0)}\nu \leq d^{(0)},\ g^T\nu = -1,\ \lambda \geq 0,\ \mathbf{1}^T\lambda = 1\}.$$

For example, we can choose

$$C^{(0)} = \begin{bmatrix} I \\ -I \end{bmatrix} \quad d^{(0)} = \begin{bmatrix} \overline{y} \\ -\underline{y} \end{bmatrix},$$

where $\underline{y}$ and $\overline{y}$ are known bounds. We should notice that the polyhedron $\mathcal{P}^{\circ(0)}$ contains all possible solutions of (2.4) and every point in it satisfy

$$\mathbf{1}^T\lambda = 1, \quad \lambda \succeq 0, \quad g^T\nu \geq 1,$$

and it is therefore feasible for the problem (2.4).

The primal-dual ACCPM can then be outlined as follows:

given $\mathcal{P}^{(0)}$ and $\mathcal{P}^{\circ(0)}$.
$j := 0$.
repeat

- Compute the analytic center $x^{(j)}$ of $\mathcal{P}^{(j)}$.
- Compute the analytic center $(\nu^{(j)},\ \lambda^{(j)})$ of $\mathcal{P}^{\circ(j)}$.
- Define $\mathcal{H}^{(j)} = \mathbf{R}^n$ and $\mathcal{H}^{\circ(j)} = \mathbf{R}^{k+n}$
- **for** i=1, ..., k
  - Query subsystem i at the point $x_i^{(j)}$ using the primal oracle.
  - If primal oracle returns true then

$$\mathcal{H}^{\circ(j)} = \mathcal{H}^{\circ(j)} \cap \{(\nu, \lambda) \mid x_i^{(j)T} F_i^T \nu < \lambda\}.$$

  - If primal oracle returns x° then

$$\mathcal{H}^{(j)} = \mathcal{H}^{(j)} \cap \{x \mid x_i^T x^\circ < 1\}.$$

  - Query subsystem i at the point $F_i\nu(j)\lambda_i^{-1}$ using the dual oracle.
  - If dual oracle return true then

$$\mathcal{H}^{(j)} = \mathcal{H}^{(j)} \cap \{x \mid x_i^T F_i \nu(j) \lambda_i^{-1} < 1\}.$$

– If dual oracle returns $x^\star$ then

$$\mathcal{H}^{\circ(j)} = \mathcal{H}^{\circ(j)} \cap \{(\nu, \lambda) \mid x^{\star T} F_i^T \nu < \lambda \}.$$

- If $\mathcal{H} = \mathbf{R}^n$ quit.
- If $\mathcal{H}^\circ = \mathbf{R}^{k+n}$ quit.
- Form the polyhedron $\mathcal{P}^{(j+1)} = \mathcal{P}^{(j)} \cap \mathcal{H}^{(j)}$.
- Form the polyhedron $\mathcal{P}^{\circ(j+1)} = \mathcal{P}^{\circ(j)} \cap \mathcal{H}^{\circ(j)}$.
- $j := j + 1$.

If the algorithm terminates because $\mathcal{H} = \mathbf{R}^n$, the problem is feasible and x is a solution. Vice versa the problem is infeasible and $(\nu^{(j)},\ \lambda^{(j)})$ is a certificate of infeasibility.

2.4.4 Convergence analysis

The PDCPM, as we noted before, tries to localize a solutions of two different problems at the same time. It is in other words like two ACCPMs running in parallel. One looks for a solution of problem (2.3) and the other one looks for a solution of problem (2.4). We can therefore apply the convergence properties of ACCPM to the PDCPM. Therefore we conclude that if **relint**$\{x \mid Fx = g,\ x \in \mathcal{X}\} \neq \emptyset$ or **relint**$\{\nu \mid g^T\nu \geq 1,\ F^T\nu \in \mathcal{X}^\circ\} \neq \emptyset$ the algorithm converges in a finite number of steps [GLY96].

The two conditions above are for every practical purpose always satisfied. It's also possible to add to the algorithm a maximum number of iteration after which it is automatically terminated.

2.5 Numerical examples

We now present two numerical examples. The main purpose of these examples is to show the generality of the presented framework and how it can be used to describe systems and handle nonlinear constraints. We will then show some qualitative results of the primal-dual cutting-plane method that allows us to better understand the behavior of the algorithm.

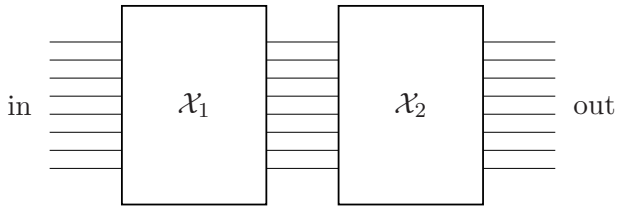

Figure 2.5: Cascaded combinatorial logic block example.

2.5.1 Cascaded combinatorial logic blocks

In the first example we consider two subsystems each containing combinatorial logic. Internally each block contains 30 gates that are connected randomly and has 8 inputs and 8 outputs. The output of the first subsystems are connected to the input of the second one. Figure 2.5 shows the 2 blocks connected together.

To be able to represent this system in our framework we will use four subsystems. Two subsystems will represent the combinatorial logic block and the remaining 2 will be used to express some constraints of the system. Moreover we will work with the logarithm of the variables. In fact it has been shown that under reasonable assumptions the feasibility set for a combinatorial logic circuit is convex after a logarithmic change of coordinates [BK05].

We start by describing the two subsystems $\mathcal{X}_1$ and $\mathcal{X}_2$ that encapsulates the combinatorial logic. The interface variables are the same for these two subsystems and are power consumption, maximum input capacitance among all inputs, maximum load capacitance over all outputs, and worst case delay from input to output. We can therefore, using the logarithm, define the interface variables as

$$\begin{aligned} x_1 &= \log C_{1\text{in}},\ x_2 = \log C_{1\text{load}},\ x_3 = \log D_1,\ x_4 = \log P_1 \\ x_5 &= \log C_{2\text{in}},\ x_6 = \log C_{2\text{load}},\ x_7 = \log D_2,\ x_8 = \log P_2 \end{aligned}$$

where C_{in} is the maximum input capacitance among all inputs, C_{load} is the maximum output load, D is the worst case delay and P is the power consumption. We should notice that we are using the maximum input capacitance among all inputs in order to simplify this example but we could as well use different interface variables for each of the inputs and outputs. Internally each block is described with equations that are compatible with

geometric programming [BKVH06].

We now describe the two subsystems $\mathcal{X}_3$ and $\mathcal{X}_4$ used to impose some constraints on the circuit. We will first give the mathematical description of each of them and we will then explain why we need them and how we are going to use them. We have

$$\begin{aligned}\mathcal{X}_3 &= \{(x_9, x_{10}, x_{11}) \mid \exp(x_9) + \exp(x_{10}) \leq \exp(x_{11})\} \\ \mathcal{X}_4 &= \{(x_{12}, x_{13}, x_{14}) \mid \exp(x_{12}) + \exp(x_{13}) \leq \exp(x_{14})\}\end{aligned}$$

It's easy to see that both $\mathcal{X}_3$ and $\mathcal{X}_4$ are convex.

We will use them to describe the constraint on power and delay of the system. Since we have a maximum power consumption specification for the system we need to impose the constraint $P_1 + P_2 \leq P_{\text{tot}}$, but since we work in a logarithmic space we need to express the condition $\log P_1 + \log P_2 \leq \log P_{\text{tot}}$. We have a similar equation for the delay; since the system has a specification on the maximum delay D_{tot} we need to impose $\log D_1 + \log D_2 \leq \log D_{\text{tot}}$. It's clear now how we can use the two subsystems to impose the above constraints. We will have

$$\begin{aligned}x_9 = \log D_1,\ x_{10} = \log D_2,\ x_{11} = \log D_{\text{tot}} \\ x_{12} = \log P_1,\ x_{13} = \log P_2,\ x_{14} = \log P_{\text{tot}}.\end{aligned}$$

Figure 2.6 shows the block diagram of the original system represented in our framework.

We will now explicitly give the matrix F and the vector g that represent all the constraints for this system formed by $\mathcal{X}_1$, $\mathcal{X}_2$, $\mathcal{X}_3$, and $\mathcal{X}_4$. We have

$$F = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \end{bmatrix} \quad g = \begin{bmatrix} \log C_{\text{in}} \\ \log C_{\text{load}} \\ \log D_{\text{tot}} \\ \log P_{\text{tot}} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

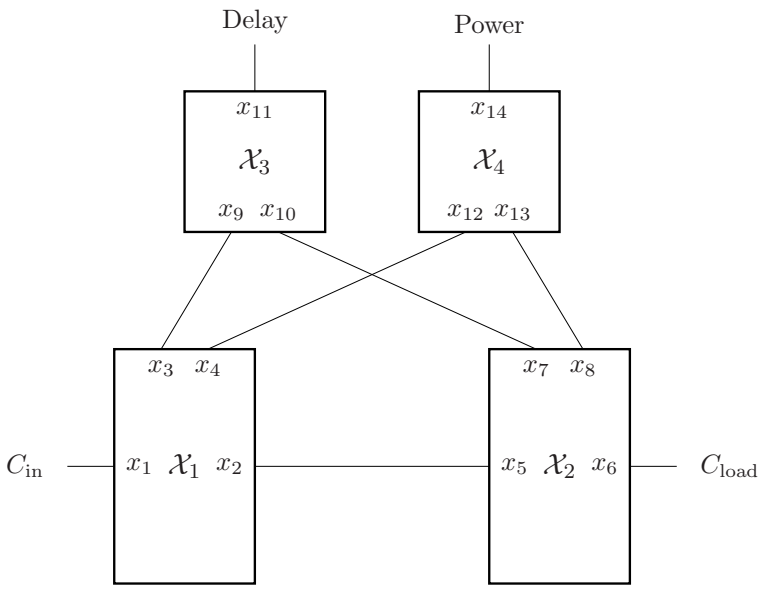

Figure 2.6: Block diagram of the combinatorial logic block systems represented in the new framework.

For example, the first row of F enforces that the input maximum input capacitance of the first block is equal to the specification for the input capacitance C_{in}. The last row instead specifies that $x_8 = x_{13}$.

Even though this example is very simple, it shows that our model can handle non-linear constraints by using additional subsystems and it's easy to see how can it be used for more complicated systems.

We ran the algorithm for some given value of D_{tot}, P_{tot}, C_{in}, and C_{load}. At each iteration of the algorithm we represent the ith subsystem with a white dot if the primal oracle returned feasibility and black otherwise. Figure 2.7 shows that in 25 steps the algorithm find a feasible design for this specific set of specifications. We should notice that a white dot corresponds a new cutting-plane in the dual problem and a black dot corresponds to a new cutting-plane for the primal problem.

Figure 2.8 shows the power allocation between the two subsystems as the algorithm runs. We notice that the final power allocation gives more power to the second block as we would expect. Moreover for the first iterations the power allocated to the blocks exceeds the total power allocation. This corresponds to a choice of x infeasible for $\mathcal{X}_4$. Eventually the specification on power is met and the sum of the powers of the two blocks

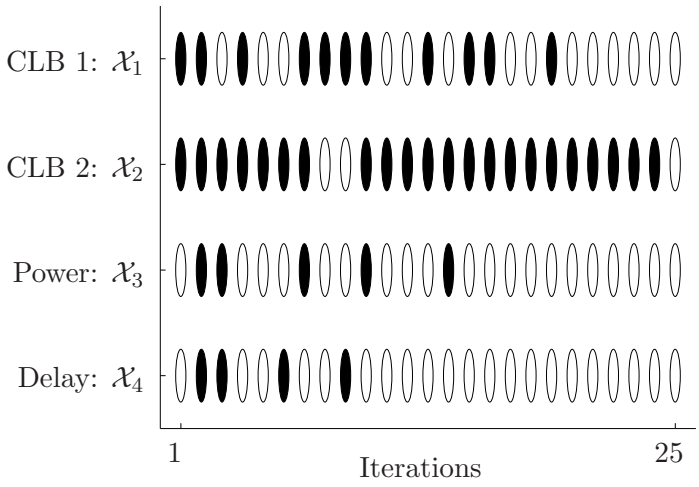

Figure 2.7: Primal-dual algorithm iterations for the cascaded combinatorial logic blocks example: white dot represents a feasible subsystem, a black dot represents an infeasible subsystem.

is less than the total allowed power.

2.5.2 Cascaded amplifiers

The second example consists of two cascaded amplifiers (figure 2.9). The global specifications are the total power, area, gain, input referred noise, bandwidth, phase margin, input capacitance, output load and input/output swing of the two cascaded amplifiers.

The interface variables are area, power, gain, input referred noise, bandwidth, phase margin, maximum input capacitance, output load capacitance and maximum input and output swing. As for the previous example we work with the logarithm of these variables because this allows us to have the set of feasible design convex [HBL98].

The internal topology of the amplifier is shown in figure 2.10. The common mode circuitry is not shown for simplicity. All the equations for the amplifier are compatible with geometric programming [HBL98]; therefore the oracles for the subsystems are obtained using a geometric programming solver as we will show in chapter 3.

It's also necessary to add another subsystem to enforce the nonlinear constraints that guarantee that the system meets the global specifications. All these constraints can be written in a form compatible with geometric programming and therefore we can again use a geometric programming solver to create the subsystem [HBL98].

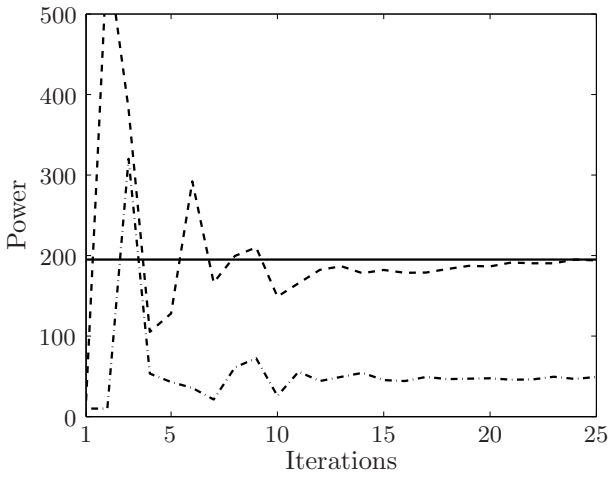

Figure 2.8: Power allocation versus algorithm iterations for the cascaded combinatorial logic blocks example. The black line is the total power constraint. The dash-dotted line represents the power allocated to the first block and the dashed line represents the sum of the power allocated to both blocks.

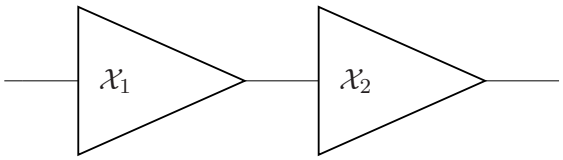

Figure 2.9: Cascaded amplifiers example.

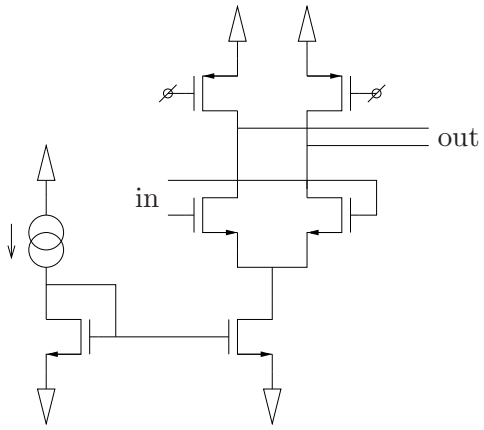

Figure 2.10: Amplifier topology. The common mode circuitry is not shown for simplicity.

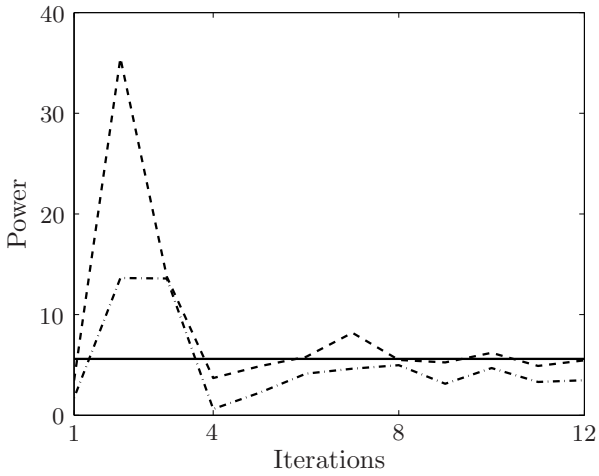

Figure 2.11: Power allocation versus iterations for the cascaded amplifier example. The dashdotted line represents the power allocated to the first amplifier and the dashed line represents the power allocated to both of them. The black line is maximum total power.

We show the behavior of the algorithm for a given run of the algorithm. Figure 2.11 shows the power allocation at each iterations of the algorithm. The dash-dotted line represents the power allocated to the first amplifier and the dashed line represents the power allocated to both of them. The black line is maximum power specification. The algorithm produces a feasible design and at the end as we expected the power allocated to the first amplifier is more than the one allocated to the second one. This is due to the fact that the amplifier is in this case limited by the noise.

Figure 2.12 shows the input referred noise of each amplifier together with the noise specification (in black). The dashdotted line is the input referred noise of the first amplifier and the dashed line is the input referred noise of the both of them. Again as we expect the biggest contribution to the overall noise is due to the first amplifier.

2.6 Remarks

We should notice that if it's possible to write the design problem as a larger convex problem, it is also possible to rewrite it so that it would fit in this framework. For example, if it has been shown that a certain design problem can be cast as a convex problem, we could divide this problem in smaller ones that would represent the subsystems and rewrite everything in this framework. This applies, for example, to all the literature on

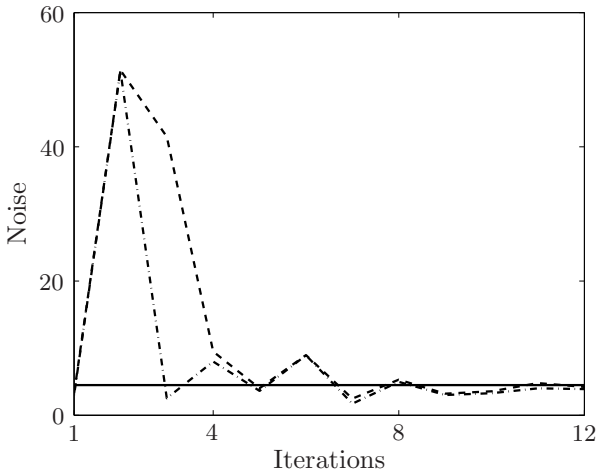

Figure 2.12: Input referred noise versus iterations for the cascaded amplifier example. The dashdotted line represents the input referred noise of the first amplifier and the dashed line represents the input referred noise of both of them. The black line is maximum total input referred noise.

design problems that can be carried out using geometric programming. The examples of this chapter fall in this category.

Chapter 3

Subsystem modeling

3.1 Introduction

In this chapter we consider the problem of creating the interface (primal and dual oracle) that can be used in the framework we introduced in chapter 2. A subsystem should have a convex feasible set $\mathcal{X}$ and should also have a primal and dual oracle available. We will address two different situations. In the first one, the subsystem contains internally an optimization problem that is used to determine if, for given interface variables, the subsystem is feasible. In the second case the feasible set of the subsystem is described as the sublevel set of some given function.

3.2 Subsystem model through an optimization problem

We consider a subsystem $\mathcal{X}$ with internal variables y, and interface variables x. We assume that internally the subsystem is modelled with an convex feasibility problem. In particular we have that a given x is feasible if and only if there is a feasible solution of the problem

$$\begin{array}{ll} \text{find} & y \\ \text{subject to} & f_i(x,y) \leq 0, \quad i = 1, \ldots, m \end{array} \tag{3.1}$$

where the functions $f_i(x,y)$ are jointly convex in x and y.

In the next two sections we will show how to create a primal and a dual oracle from the dual variables of an optimization problem closely related to problem (3.1).

3.2.1 Primal oracle

We would like to create the primal oracle in the case where it is queried with the vector $\bar{x}$.

It's easy to see that if we solve the problem

$$\begin{array}{ll} \text{find} & y,\ x \\ \text{subject to} & f_i(x,y) \leq 0, \quad i = 1,\ldots,m \\ & x = \bar{x} \end{array} \tag{3.2}$$

with variables x and y, and we find a feasible solution then by definition $\bar{x}$ is feasible and the primal oracle should return feasibility.

Vice versa if the problem is infeasible and μ are the dual variables associated with the constraints $x = \bar{x}$ we know [BV04] that for any x such that

$$x^T\mu/\mu^T\bar{x} < 1$$

there won't be any feasible solution of problem (3.2) and therefore no feasible design with interface variables x. This shows that $x^\circ = -\mu/\mu^T\bar{x}$ is a cutting-plane and can be used as the output of the primal oracle.

Therefore to create the primal oracle we solve problem (3.2) and if the problem is feasible the primal oracle returns feasibility otherwise it returns $x^\circ = -\mu/\mu^T\bar{x}$ where μ is the dual variable associated with the equality constraint $x = \bar{x}$ of problem (3.2).

3.2.2 Dual oracle

We would like to find the output of the dual oracle when queried with vector x°. Therefore we would like to check if x° is a cutting-plane for the set $\mathcal{X}$. If x° was a cutting-plane it would mean that any x such that $x^Tx^\circ \geq 1$ is infeasible for problem (3.1). We can check for this condition by solving the optimization problem

$$\begin{array}{ll} \text{find} & y,\ x \\ \text{subject to} & f_i(x,y) \leq 0, \quad i = 1,\ldots,m \\ & x^Tx^\circ \geq 1 \end{array} \tag{3.3}$$

with variables x and y. Clearly if this problem is infeasible then x° is a cutting-plane for $\mathcal{X}$. If there is a solution $x^\star$, $y^\star$ to problem (3.3) then $x^\star$ corresponds to a feasible design with $x^{\star T}x^\circ \geq 1$ and can be therefore used as the output of the dual oracle.

To create a dual oracle we solve problem (3.3) and if the problem is infeasible we return feasibility for the dual oracle and if the problem is feasible, the dual oracle would return the feasible point $x^\star$.

We showed that by solving two simple modified versions of problem (3.1) we can obtain the primal and dual oracles for the subsystem. In practice the two modified problems can be solved with only a small overhead compared to solving problem (3.1).

3.3 Subsystem model through a sublevel set and epigraph of a function

In this section, we consider the case where the feasible set $\mathcal{X}$ of a subsystem is represented as the sublevel set of a convex function f and the case where the set $\mathcal{X}$ is represented as the epigraph of a convex function f.

We therefore have for the first case

$$\mathcal{X} = \{x \mid f(x) \leq \alpha\},$$

where α is a given constant that we can assume without loss of generality to be 1, and f is a convex function.

In the second case we have

$$\mathcal{X} = \{(z, t) \mid f(z) \leq t\},$$

where f is a convex function.

These situations may arise if for example the model is obtained by fitting data using some given family of functions or if we have an analytical expression for $\mathcal{X}$ obtained using some physical or mathematical assumptions. In the next chapters we will show algorithms that allow us to obtain convex fits of given data that can then be used to create subsystems using the techniques of this chapter.

We start by noticing that these two situations are special cases of a subsystem described through problem (3.1). In fact for the first case, we have that the interface

variables x are feasible if and only if problem

$$\begin{array}{ll} \text{find} & y \\ \text{subject to} & f(x) - 1 \leq 0, \end{array}$$

is feasible. For the second case instead we have that the interface variables $x = (z, t)$ are feasible if and only if problem

$$\begin{array}{ll} \text{find} & y \\ \text{subject to} & f(z) - t \leq 0 \end{array}$$

is feasible. We can therefore apply the previous results in these cases.

We should notice that if the function f has some special form the problem of creating primal and dual oracles can be simplified. As an example let's assume that

$$f(x) = \max(a_i^T x + b_i), \quad i = 1, \ldots, k$$

and $\mathcal{X}$ is the epigraph of f:

$$\mathcal{X} = \{(z,\ t) \mid f(z) \leq t\}.$$

In this case the primal oracle can be obtained very efficiently. Let's suppose that the primal oracle is queried with the vector $\hat{x} = (\hat{z},\ \hat{t})$. We can easily check if this point belongs to $\mathcal{X}$ by simply checking if the inequalities

$$a_i^T \hat{z} + b_i \geq t, \quad i = 1, \ldots, k$$

are all satisfied. If for example the mth inequality is not satisfied the primal oracle would return the vector $(b_i^{-1} a_i,\ -b^{-1})$ which is a cutting-plane for the set $\mathcal{X}$.

Chapter 4

Convex piecewise-linear fitting

4.1 Introduction

We consider the problem of fitting some given data

$$(u_1, y_1), \ldots, (u_m, y_m) \in \mathbf{R}^n \times \mathbf{R}$$

with a convex piecewise-linear function $f : \mathbf{R}^n \to \mathbf{R}$ from some set $\mathcal{F}$ of candidate functions. With a least-squares fitting criterion, we obtain the problem

$$\begin{array}{ll} \text{minimize} & J(f) = \sum_{i=1}^m (f(u_i) - y_i)^2 \\ \text{subject to} & f \in \mathcal{F}, \end{array} \tag{4.1}$$

with variable f. We refer to $(J(f)/m)^{1/2}$ as the RMS (root-mean-square) fit of the function f to the data. The convex piecewise-linear fitting problem (4.1) is to find the function f, from the given family $\mathcal{F}$ of convex piecewise-linear functions, that gives the best (smallest) RMS fit to the given data.

Our main interest is in the case when n (the dimension of the data) is relatively small, say not more than 5 or so, while m (the number of data points) can be relatively large, *e.g.*, 10^4 or more. The methods we describe, however, work for any values of n and m.

Several special cases of the convex piecewise-linear fitting problem (4.1) can be solved exactly. When $\mathcal{F}$ consists of the affine functions, *i.e.*, f has the form $f(x) = a^T x + b$, the problem (4.1) reduces to an ordinary linear least-squares problem in the function

parameters $a \in \mathbf{R}^n$ and $b \in \mathbf{R}$ and so is readily solved. As a less trivial example, consider the case when $\mathcal{F}$ consists of *all* piecewise-linear functions from $\mathbf{R}^n$ into $\mathbf{R}$, with no other constraint on the form of f. This is the *nonparametric* convex piecewise-linear fitting problem. Then the problem (4.1) can be solved, exactly, via a quadratic program (QP); see [BV04, §6.5.5]. This nonparametric approach, however, has two potential practical disadvantages. First, the QP that must be solved is very large (containing more than mn variables), limiting the method to modest values of m (say, a thousand). The second potential disadvantage is that the piecewise-linear function fit obtained can be very complex, with many terms (up to m).

Of course, not all data can be fit well (*i.e.*, with small RMS fit) with a convex piecewise-linear function. For example, if the data are samples from a function that has strong negative (concave) curvature, then no convex function can fit it well. Moreover, the best fit (which will be poor) will be obtained with an affine function. We can also have the opposite situation: it can occur that the data can be *perfectly* fit by an affine function, *i.e.*, we can have $J = 0$. In this case we say that the data is *interpolated* by the convex piecewise-linear function f.

4.1.1 Max-affine functions

We consider the parametric fitting problem, in which the candidate functions are parametrized by a finite-dimensional vector of coefficients $\alpha \in \mathbf{R}^p$. One very simple form is given by $\mathcal{F}^k_{\mathrm{ma}}$, the set of functions on $\mathbf{R}^n$ with the form

$$f(x) = \max\{a_1^T x + b_1, \ldots, a_k^T x + b_k\}, \tag{4.2}$$

i.e., a maximum of k affine functions. We refer to a function of this form as 'max-affine', with k terms. The set $\mathcal{F}^k_{\mathrm{ma}}$ is parametrized by the coefficient vector

$$\alpha = (a_1, \ldots, a_k, b_1, \ldots, b_k) \in \mathbf{R}^{k(n+1)}.$$

In fact, any convex piecewise-linear function on $\mathbf{R}^n$ can be expressed as a max-affine function, for some k, so this form is in a sense universal. Our interest, however, is in the case when the number of terms k is relatively small, say no more than 10, or a few 10s. In this case the max-affine representation (4.2) is compact, in the sense that the number of parameters needed to describe f (*i.e.*, p) is much smaller than the number of

parameters in the original data set (*i.e.*, $m(n+1)$). The methods we describe, however, do not require k to be small.

When $\mathcal{F} = \mathcal{F}^k_{\rm ma}$, the fitting problem (4.1) reduces to the nonlinear least-squares problem

$$\text{minimize} \quad J(\alpha) = \sum_{i=1}^{m} \left(\max_{j=1,\ldots,k} (a_j^T u_i + b_j) - y_i \right)^2, \tag{4.3}$$

with variables $a_1, \ldots, a_k \in \mathbf{R}^n$, $b_1, \ldots, b_k \in \mathbf{R}$. The function J is a piecewise-quadratic function of α. Indeed, for each i, $f(u_i) - y_i$ is piecewise-linear, and J is the sum of squares of these functions, so J is convex quadratic on the (polyhedral) regions on which $f(u_i)$ is affine. But J is not globally convex, so the fitting problem (4.3) is not convex.

4.1.2 A more general parametrization

We will also consider a more general parametrized form for convex piecewise-linear functions,

$$f(x) = \psi(\phi(x, \alpha)), \tag{4.4}$$

where $\psi : \mathbf{R}^q \to \mathbf{R}$ is a (fixed) convex piecewise-linear function, and $\phi : \mathbf{R}^n \times \mathbf{R}^p \to \mathbf{R}^q$ is a (fixed) bi-affine function. (This means that for each x, $\phi(x, \alpha)$ is an affine function of α, and for each α, $\phi(x, \alpha)$ is an affine function of x.) The simple max-affine parametrization (4.2) has this form, with $q = k$, $\psi(z_1, \ldots, z_k) = \max\{z_1, \ldots, z_k\}$, and $\phi_i(x, \alpha) = a_i^T x + b_i$.

As an example, consider the set of functions $\mathcal{F}$ that are sums of k terms, each of which is the maximum of two affine functions,

$$f(x) = \sum_{i=1}^{k} \max\{a_i^T x + b_i, c_i^T x + d_i\}, \tag{4.5}$$

parametrized by $a_1, \ldots, a_k,\ c_1, \ldots, c_k \in \mathbf{R}^n$ and $b_1, \ldots, b_k,\ d_1, \ldots, d_k \in \mathbf{R}$. This family corresponds to the general form (4.4) with

$$\psi(z_1, \ldots, z_k, w_1, \ldots, w_k) = \sum_{i=1}^{k} \max\{z_i, w_i\},$$

and

$$\phi(x, \alpha) = (a_1^T x + b_1, \ldots, a_k^T x + b_k, c_1^T x + d_1, \ldots, c_k^T x + d_k).$$

Of course we can expand any function with the more general form (4.4) into its max-affine representation. But the resulting max-affine representation can be very much larger than the original general form representation. For example, the function form (4.5) requires $p = 2k(n + 1)$ parameters. If the same function is written out as a max-affine function, it requires 2^k terms, and therefore $2^k(n+1)$ parameters. The hope is that a well chosen general form can give us a more compact fit to the given data than a max-affine form with the same number of parameters.

As another interesting example of the general form (4.4), consider the case in which f is given as the optimal value of a linear program (LP) with the righthand side of the constraints depending bi-affinely on x and the parameters:

$$f(x) = \min\{c^T v \mid Av \leq b + Bx\}.$$

Here c and A are fixed; b and B are considered the parameters that define f. This function can be put in the general form (4.4) using

$$\psi(z) = \min\{c^T v \mid Av \leq z\}, \qquad \phi(x, b, B) = b + Bx.$$

The function ψ is convex and piecewise-linear (see, *e.g.*, [BV04]); the function ϕ is evidently bi-affine in x and (b, B).

4.1.3 Dependent variable transformation and normalization

We can apply a nonsingular affine transformation to the dependent variable u, by forming

$$\tilde{u}_i = Tu_i + s, \quad i = 1, \ldots, m,$$

where $T \in \mathbf{R}^{n\times n}$ is nonsingular and $s \in \mathbf{R}^n$. Defining $\tilde{f}(\tilde{x}) = f(T^{-1}(x - s))$, we have $\tilde{f}(\tilde{u}_i) = f(u_i)$. If f is piecewise-linear and convex, then so is $\tilde{f}$ (and of course, vice versa). Provided $\mathcal{F}$ is invariant under composition with affine functions, the problem of fitting the data (u_i, y_i) with a function $f \in \mathcal{F}$ is the same the problem of fitting the data $(\tilde{u}_i, y_i)$ with a function $\tilde{f} \in \mathcal{F}$.

This allows us to normalize the dependent variable data in various ways. For example,

we can assume that it has zero (sample) mean and unit (sample) covariance,

$$\bar{u} = (1/m)\sum_{i=1}^{m} u_i = 0, \qquad \Sigma_u = (1/m)\sum_{i=1}^{m} u_i u_i^T = I, \tag{4.6}$$

provided the data u_i are affinely independent. (If they are not, we can reduce the problem to an equivalent one with smaller dimension.)

4.1.4 Previous work

Piecewise-linear functions arise in many areas and contexts. Some general forms for representing piecewise-linear functions can be found in, *e.g.*, [KC78, KC90]. Several methods have been proposed for fitting general piecewise-linear functions to (multidimensional) data. A neural network algorithm is used in [GDD02]; a Gauss-Newton method is used in [JJD98, HB97] to find piecewise-linear approximations of smooth functions. An iterative procedure, similar in spirit to our method, is described in [FTM02]. Software for fitting general piecewise-linear functions to data include, *e.g.*, [TB04, SF02].

The special case $n = 1$, *i.e.*, fitting a function on $\mathbf{R}$, by a piecewise-linear function has been extensively studied. For example, a method for finding the minimum number of segments to achieve a given maximum error is described in [Dun86]; the same problem can be approached using dynamic programming [Goo94, BR69, HS91, WHCL93], or a a genetic algorithm [PM00]. The problem of simplifying a given piecewise-linear function on $\mathbf{R}$, to one with fewer segments, is considered in [II86].

Another related problem that has received much attention is the problem of fitting a piecewise-linear curve, or polygon, in $\mathbf{R}^2$ to given data; see, *e.g.*, [ABO$^+$85, MS92]. An iterative procedure, closely related to the k-means algorithm and therefore similar in spirit to our method, is described in [PR88, Yin98].

Piecewise-linear functions and approximations have been used in many applications, such as detection of patterns in images [RDPR85], contour tracing [DLTW90], extraction of straight lines in aerial images [VC92], global optimization [MRT05], compression of chemical process data [BS96], and circuit modeling [JJD98, CD86, VdMV89].

We are aware of only two papers which consider the problem of fitting a piecewise-linear convex function to given data. In [MRT05] Mangasarian et al. describe a heuristic method for fitting a piecewise-linear convex function of the form $a + b^T x + \|Ax + c\|_1$ to given data (along with the constraint that the function underestimate the data). The

closest related work that we know of is [KLVY04]. In this paper, Kim et al. describe a method for fitting a (convex) max-affine function to given data, increasing the number of terms to get a better fit. (In fact they describe a method for fitting a max-monomial function to circuit models; see §4.2.3.)

4.2 Applications

As we described in chapter 3 the resulting fit can be used as the starting block to construct a subsystem to be used in a distributed deisgn problem. In this section we briefly describe some other applications of convex piecewise-linear fitting.

4.2.1 LP modeling

One application is in LP modeling, *i.e.*, approximately formulating a practical problem as an LP. Suppose a problem is reasonably well modeled using linear equality and inequality constraints, with a few nonlinear inequality constraints. By approximating these nonlinear functions by convex piecewise-linear functions, the overall problem can be formulated as an LP, and therefore efficiently solved.

As an example, consider a minimum fuel optimal control problem, with linear dynamics and a nonlinear fuel-use function,

$$\begin{array}{ll} \text{minimize} & \sum_{t=0}^{T-1} f(u(t)) \\ \text{subject to} & x(t+1) = A(t)x(t) + B(t)u(t), \quad t = 0, \ldots, T-1, \\ & x(0) = x_{\text{init}}, \quad x(T) = x_{\text{des}}, \end{array}$$

with variables $x(0), \ldots, x(T) \in \mathbf{R}^n$ (the state trajectory), and $u(0), \ldots, u(T-1) \in \mathbf{R}^m$ (the control input). The problem data are $A(0), \ldots, A(T-1)$ (the dynamics matrices), $B(0), \ldots, B(T-1)$ (the control matrices), x_{init} (the initial state), and x_{des} (the desired final state). The function $f : \mathbf{R}^m \to \mathbf{R}$ is the fuel-use function, which gives the fuel consumed in one period, as a function of the control input value. Now suppose we have empirical data or measurements of some values of the control input $u \in \mathbf{R}^m$, along with the associated fuel use $f(u)$. If we can fit these data with a convex piecewise-linear function, say,

$$f(u) \approx \hat{f}(u) = \max_{j=1,\ldots,k} (a_j^T u + b_j),$$

then we can formulate the (approximate) minimum fuel optimal control problem as the LP

$$\begin{array}{ll} \text{minimize} & \sum_{t=0}^{T-1} \tilde{f}(t) \\ \text{subject to} & x(t+1) = A(t)x(t) + B(t)u(t), \quad t = 0, \ldots, T-1, \\ & x(0) = x_{\text{init}}, \quad x(T) = x_{\text{des}}, \\ & \tilde{f}(t) \geq a_j^T u(t) + b_j, \quad t = 1, \ldots T-1, \quad j = 1, \ldots, k, \end{array} \tag{4.7}$$

with variables $x(0), \ldots, x(T) \in \mathbf{R}^n$, $u(0), \ldots, u(T-1) \in \mathbf{R}^m$, and $\tilde{f}(0), \ldots, \tilde{f}(T-1) \in \mathbf{R}$.

4.2.2 Simplifying convex functions

Another application of convex piecewise-linear fitting is to simplify a convex function that is complex, or expensive to evaluate. To illustrate this idea, we continue our minimum fuel optimal control problem described above, with a piecewise-linear fuel use function. Consider the function $V : \mathbf{R}^n \to \mathbf{R}$, which maps the initial state x_{init} to its associated minimum fuel use, *i.e.*, the optimal value of the LP (4.7). (This is the Bellman value function for the optimal control problem.) The value function is piecewise-linear and convex, but very likely requires an extremely large number of terms to be expressed in max-affine form. We can (possibly) form a simple approximation of V by a max-affine function with many fewer terms, as follows. First, we evaluating V via the LP (4.7), for a large number of initial conditions. Then, we fit a max-affine function with a modest number of terms to the resulting data. This convex piecewise-linear approximate value function can be used to construct a simple feedback controller that approximately minimizes fuel use; see, *e.g.*, [BBM02].

4.2.3 Max-monomial fitting for geometric programming

Max-affine fitting can be used to find a max-monomial approximation of a positive function, for use in geometric programming modeling; see [BKVH06]. Given data $(z_i, w_i) \in \mathbf{R}^n_{++} \times \mathbf{R}_{++}$, we form

$$u_i = \log z_i, \quad y_i = \log w_i, \quad i = 1, \ldots, m.$$

(The log of a vector is interpreted as componentwise.) We now fit this data with a max-affine model,

$$y_i \approx \max\{a_1^T u_i + b_1, \ldots, a_k^T u_i + b_k\}.$$

This gives us the max-monomial model

$$w_i \approx \max\{g_1(z_i), \ldots, g_K(z_i)\},$$

where g_i are the monomial functions

$$g_j(z) = e^{b_i} z_1^{a_{j1}} \cdots z_n^{a_{jn}}, \quad j = 1, \ldots, K.$$

(These are not monomials in the standard sense, but in the sense used in geometric programming.)

4.3 Least-squares partition algorithm

4.3.1 The algorithm

In this section we present a heuristic algorithm to (approximately) solve the k-term max-affine fitting problem (4.3), *i.e.*,

$$\text{minimize} \quad J = \sum_{i=1}^{m} \left(\max_{j=1,\ldots,k} (a_j^T u_i + b_j) - y_i \right)^2,$$

with variables $a_1, \ldots, a_k \in \mathbf{R}^n$ and $b_1, \ldots, b_k \in \mathbf{R}$. The algorithm alternates between partitioning the data and carrying out least-squares fits to update the coefficients.

We let $P_j^{(l)}$ for $j = 1, \ldots, k$, be a partition of the data indices at the lth iteration, *i.e.*, $P_j^{(l)} \subseteq \{1, \ldots, m\}$, with

$$\bigcup_j P_j^{(l)} = \{1, \ldots, m\}, \qquad P_i^{(l)} \cap P_j^{(l)} = \emptyset \text{ for } i \neq j.$$

(We will describe methods for choosing the initial partition $P_j^{(0)}$ later.)

Let $a_j^{(l)}$ and $b_j^{(l)}$ denote the values of the parameters at the lth iteration of the algorithm. We generate the next values, $a_j^{(l+1)}$ and $b_j^{(l+1)}$, from the current partition $P_j^{(l)}$, as follows. For each $j = 1, \ldots, k$, we carry out a least-squares fit of $a_j^T u_i + b_j$ to y_i, using only the data points with $i \in P_j^{(l)}$. In other words, we take $a_j^{(l+1)}$ and $b_j^{(l+1)}$ as

values of a and b that minimize

$$\sum_{i \in P_j^{(l)}} (a^T u_i + b - y_i)^2. \tag{4.8}$$

In the simplest (and most common) case, there is a unique pair (a, b) that minimizes (4.8), *i.e.*,

$$\begin{bmatrix} a_j^{(l+1)} \\ b_j^{(l+1)} \end{bmatrix} = \begin{bmatrix} \sum u_i u_i^T & \sum u_i \\ \sum u_i^T & |P_j^{(l)}| \end{bmatrix}^{-1} \begin{bmatrix} \sum y_i u_i \\ \sum y_i \end{bmatrix}, \tag{4.9}$$

where the sums are over $i \in P_j^{(l)}$.

When there are multiple minimizers of the quadratic function (4.8), *i.e.*, the matrix to be inverted in (4.9) is singular, we have several options. One option is to add some regularization to the simple least-squares objective in (4.8), *i.e.*, an additional term of the form $\lambda \|a\|_2^2 + \mu b^2$, where λ and μ are positive constants. Another possibility is to take the updated parameters as the unique minimizer of (4.8) that is closest to the previous value, $(a_j^{(l)}, b_j^{(l)})$, in Euclidean norm.

Using the new values of the coefficients, we update the partition to obtain $P_j^{(l+1)}$, by assigning i to $P_s^{(l+1)}$ if

$$f^{(l)}(u_i) = \max_{s=1,\ldots,k} (a_s^{(l)T} u_i + b_s^{(l)}) = a_j^{(l)T} u_i + b_j^{(l)}. \tag{4.10}$$

(This means that the term $a_j^{(l)T} u_i + b_j^{(l)}$ is 'active' at the data point u_i.) Roughly speaking, this means that $P_j^{(l+1)}$ is the set of indices for which the affine function $a_j^T z + b_j$ is the maximum; we can break ties (if there are any) arbitrarily.

This iteration is run until convergence, which occurs if the partition at an iteration is the same as the partition at the previous iteration, or some maximum number of iterations is reached.

We can write the algorithm as

Least-squares Partition Algorithm.

given partition $P_1^{(0)}, \ldots, P_K^{(0)}$ of $\{1, \ldots, m\}$, iteration limit $l_{\max}$

for $l = 0, \ldots, l_{\max}$

1. Compute $a_j^{(l+1)}$ and $b_j^{(l+1)}$ as in (4.9).

2. Form the partition $P_1^{(l+1)}, \ldots, P_k^{(l+1)}$ as in (4.10).
3. Quit if $P_j^{(l)} = P_j^{(l+1)}$ for $j = 1, \ldots, k$.

During the execution of the least-squares partition algorithm, one or more of the sets $P_j^{(l)}$ can become empty. The simplest approach is to drop empty sets from the partition, and continue with a smaller value of k.

4.3.2 Interpretation as Gauss-Newton method

We can interpret the algorithm as a Gauss-Newton method for the problem (4.3). Suppose that at a point $u \in \mathbf{R}^n$, there is a unique j for which $f(u) = a_j^T u + b_j$ (*i.e.*, there are no ties in the maximum that defines $f(u)$). In this case the function f is differentiable with respect to a and b; indeed, it is locally affine in these parameter values. Its first order approximation at a, b is

$$f(u) \approx \hat{f}(u) = \tilde{a}_j^T u + \tilde{b}_j.$$

This approximation is exact, provided the perturbed parameter values $\tilde{a}_1, \ldots, \tilde{a}_k$, $\tilde{b}_1, \ldots, \tilde{a}_b$ are close enough to the parameter values $a_1, \ldots, a_k$, $b_1, \ldots, a_b$.

Now assume that for each data point u_i, there is a unique j for which $f(u_i) = a_j^{(l)T} u_i + b_j^{(l)}$ (*i.e.*, there are no ties in the maxima that define $f(u_i)$). Then the first order approximation of $(f(u_1), \ldots, f(u_m))$ is given by

$$f(u_i) \approx \hat{f}(u_i) = \tilde{a}_{j(i)}^T u_i + \tilde{b}_{j(i)},$$

where $j(i)$ is the unique active j at u_i, *i.e.*, $i \in P_j^{(l)}$.

In the Gauss-Newton method for a nonlinear least-squares problem, we form the first order approximation of the argument of the norm, and solve the resulting least-squares problem to get the next iterate. In this case, then, we form the linear least-squares problem of minimizing

$$\hat{J} = \sum_{i=1}^m \left(\hat{f}(u_i) - y_i\right)^2 = \sum_{i=1}^m \left(\tilde{a}_{j(i)}^T u_i + \tilde{b}_{j(i)} - y_i\right)^2,$$

over the variables $\tilde{a}_1, \ldots, \tilde{a}_k$, $\tilde{b}_1, \ldots, \tilde{b}_k$. We can re-arrange the sum defining J into terms

involving each of the pairs of variables $a_1, b_1, \ldots, a_k, b_k$ separately:

$$\hat{J} = \hat{J}_1 + \cdots + \hat{J}_k,$$

where

$$\hat{J}_j = \sum_{i \in P_j^{(l)}} (\tilde{a}^T u_i + \tilde{b} - y_i)^2, \quad j = 1, \ldots, k.$$

Evidently, we can minimize $\hat{J}$ by separately minimizing each $\hat{J}_i$. Moreover, the parameter values that minimize $\hat{J}$ are precisely $a_1^{(l+1)}, \ldots, a_k^{(l+1)}$, $b_1^{(l+1)}, \ldots, b_k^{(l+1)}$. This is exactly the least-squares partition algorithm described above.

The algorithm is closely related to the k-means algorithm used in least-squares clustering [GG91]. The k-means algorithm approximately solves the problem of finding a set of k points in $\mathbf{R}^n$, $\{z_1, \ldots, z_k\}$, that minimizes the mean square Euclidean distance to a given data set $u_1, \ldots, u_m \in \mathbf{R}^n$. (The distance between a point u and the set of points $\{z_1, \ldots, z_k\}$ is defined as the minimum distance, *i.e.*, $\min_{j=1,\ldots,k} \|u - z_j\|_2$.) In the k-means algorithm, we iterate between two steps: first, we partition the data points according to the closest current point in the set $\{z_1, \ldots, z_k\}$; then we update each z_j as the mean of the points in its associated partition. (The mean minimizes the sum of the squares of the Euclidean distances to the point.) Our algorithm is conceptually identical to the k-means algorithm: we partition the data points according to which of the affine functions is active (*i.e.*, largest), and then update the affine functions, separately, using only the data points in its associated partition.

4.3.3 Nonconvergence of least-squares partition algorithm

The basic least-squares partition algorithm need not converge; it can enter a (nonconstant) limit cycle. Consider, for example, the data

$$\begin{array}{lllll} u_1 = -2, & u_2 = -1, & u_3 = 0, & u_4 = 1, & u_5 = 2 \\ y_1 = 0, & y_2 = 1, & y_3 = 3, & y_4 = 1, & y_5 = 0, \end{array}$$

and $k = 2$. The data evidently cannot be fit well by any convex function; the (globally) best fit is obtained by the constant function $f(u) = 1$. For many initial parameter values, however, the algorithm converges to a limit cycle with period 2, alternating between the

two functions

$$f_1(u) = \max\{u + 2, -(3/2)u + 17/6\}, \qquad f_2(u) = \max\{(3/2)u + 17/6, -u + 2\}.$$

The algorithm therefore fails to converge; moreover, each of the functions f_1 and f_2 gives a very suboptimal fit to the data.

On the other hand, with real data (not specifically designed to illustrate nonconvergence) we have observed that the least-squares partition algorithm appears to converge in most cases. In any case, convergence failure has no practical consequences since the algorithm is terminated after some fixed maximum number of steps, and moreover, we recommend that it be run from a number of starting points, with the best fit obtained used as the final fit.

4.3.4 Piecewise-linear fitting algorithm

The least-squares partition algorithm, used by itself, has several serious shortcomings. It need not converge, and when it does converge, it can (and often does) converge to a piecewise-linear approximation with a poor fit to the data. Both of these problems can be mitigated by running the least-squares partition algorithm multiple times, with different initial partitions. The final fit is taken to be the best fit obtained among all iterations of all runs of the algorithm.

We first describe a simple method for generating a random initial partition. We randomly choose points $p_1, \ldots, p_K$, and define the initial partition to be the Voronoi sets associated with these points. We have

$$P_j^{(0)} = \{i \mid \|u_i - p_j\| < \|u_i - p_s\| \text{ for } s \neq j\}, \quad j = 1, \ldots, K. \tag{4.11}$$

(Thus, $P_j^{(0)}$ is the set of indices of data points that are closest to p_j.) The seed points p_i should be generated according to some distribution that matches the shape of the data points u_i, for example, they can be chosen from a normal distribution with mean $\bar{u}$ and covariance Σ_u (see (4.6)).

The overall algorithm can be described as

Piecewise-Linear Fitting Algorithm.

given number of trials N_{trials}, iteration limit $l_{\max}$

for $i = 1, \ldots, N_{\text{trials}}$

1. Generate random initial partition via (4.11).
2. Run least-squares partition algorithm with iteration limit $l_{\max}$.
3. Keep track of best RMS fit obtained.

4.3.5 General form fitting

In this section we show the least-squares partition algorithm can be modified to fit piecewise-linear functions with the more general form (4.4),

$$f(x, \alpha) = \psi(\phi(x, \alpha)),$$

where ψ is a fixed convex piecewise-linear function, and ϕ is a fixed bi-affine function.

We described the least-squares partition algorithm in terms of a partition of the indices, according to which of the k affine functions is active at the point u_i. The same approach of an explicit partition will not work in the more general case, since the size of the partition can be extremely large. Instead, we start from the idea that the partition gives an approximation of $f(u_i)$ that is affine in α, and valid near $\alpha^{(l)}$. If there is no 'tie' at u_i (*i.e.*, there is a unique affine function that achieves the maximum), then the affine approximation is exact in a neighborhood of the current parameter value $\alpha^{(l)}$.

We can do the same thing with the more general form. For each i, we find $a_i(\alpha)$ and $b_i(\alpha)$, both affine functions of α, so that

$$f(u_i, \alpha) \approx a_i(\alpha)^T u_i + b_i(\alpha)$$

for α near $\alpha^{(l)}$, the current value of the parameters. This approximation is exact in a neighborhood of $\alpha^{(l)}$ if $\psi(u_i, \alpha)$ is a point of differentiability of ψ. (For max-affine functions, this is the case when there is no 'tie' at u_i.) If it is not such a point, we can choose any subgradient model of $f(u_i, \alpha)$, *i.e.*, any $a_i(\alpha)$ and $b_i(\alpha)$ for which

$$f(u_i, \alpha^{(l)}) = a_i(\alpha^{(l)})^T u_i + b_i(\alpha^{(l)}),$$

(the approximation is exact for $\alpha = \alpha^{(l)}$), and

$$f(u_i, \alpha) \geq a_i(\alpha)^T u_i + b_i(\alpha)$$

for all α. (In the case of max-affine functions, breaking any ties arbitrarily satisfies this condition.)

We then compute a new parameter value using a Gauss-Newton like method. We replace $f(u_i)$ in the expression for J with

$$\hat{f}^{(l)}(u_i, \alpha) = a_i(\alpha^{(l)})^T u_i + b_i(\alpha^{(l)}),$$

which is affine in α. We then choose $\alpha^{(l+1)}$ as the minimizer of

$$\hat{J} = \sum_{i=1}^{m} (\hat{f}^{(l)}(u_i) - y_i)^2,$$

which can be found using standard linear least-squares.

To damp this update rule, we can add a regularization term to $\hat{J}$, by choosing $\alpha^{(l+1)}$ as the minimizer of

$$\sum_{i=1}^{m} (\hat{f}^{(l)}(u_i) - y_i)^2 + \rho \|\alpha - \alpha^{(l)}\|^2,$$

where $\rho > 0$ is a parameter.

4.4 Improved least-squares partition algorithm

The main problem with the previous algorithm is that each minimization is performed locally on a set of points and no information regarding the other points is retained. This is in general not a problem if the given data can be interpolated with a convex function but, in other cases, the algorithm might not produce good results and not even converge.

We now present an improved partition algorithm that guarantees the convergence of J. In §4.4.1, we will interpret this algorithm and give an intuitive derivation of it. In §4.4.2, we will then give a proof of convergence and in §4.4.3, we will show the connection with the simple partition algorithm.

First, to simplify the notation we define

$$\tilde{f}_j^{(l)}(x) = \max\{a_1^{(l)T} x + b_1^{(l)}, \ldots, a_{j-1}^{(l)T} x + b_{j-1}^{(l)}, a_{j+1}^{(l)T} x + b_{j+1}^{(l)}, \ldots, a_k^{(l)T} u_i + b_k^{(l)}\}.$$

We can interpret $\tilde{f}_j$ as the fitting function we would obtain after the lth iteration of the simple partition algorithm by eliminating the jth term in the max-affine form.

As for the simple partition algorithm, we let $a_j^{(l)}$ and $b_j^{(l)}$ denote the values at the lth iteration of the algorithm and $P_j^{(l)}$ for $j = 1, \ldots, k$ a partition on the data indices at the lth iteration.

Given a partition of the data indices at the lth iteration, we pick a specific set $P_h^{(l)}$ and we take $a_h^{(l+1)}$ and $b_h^{(l+1)}$ as the optimal value a and b of the problem

$$\begin{array}{ll} \text{minimize} & \sum_{i \in P_h^{(l)}} \max\{(a^T u_i + b - y_i)^2, (\tilde{f}_h^{(l)}(u_i) - y_i)_+^2\} \\ & + \sum_{i \notin P_h^{(l)}} ((a^T u_i + b - y_i - |f^{(l)}(u_i) - y_i|)_+ + |f^{(l)}(u_i) - y_i|)^2, \end{array} \tag{4.12}$$

where $(x)_+ = 0$ for $x \leq 0$ and $(x)_+ = x$ for $x > 0$. We set $a_j^{(l+1)} = a_j^{(l)}$ and $b_j^{(l+1)} = b_j^{(l)}$ for $h \neq j$. We then find a new parttition $P_j^{(l+1)}$ that satisfy conditions (4.10) and we keep iterating until we reach convergence.

There could be different ways of picking the set $P_h^{(l)}$ at each iteration. A natural one is to iteratively pick its value from 1 to k. The algorithm then becomes

Improved Least-squares Partition Algorithm.

given partition $P_1^{(0)}, \ldots, P_K^{(0)}$ of $\{1, \ldots, m\}$, iteration limit $l_{\max}$

for $l = 0, \ldots, l_{\max}$

1. $h = \mathrm{mod}_k\ l + 1$.
2. Compute $a_h^{(l+1)}$ and $b_h^{(l+1)}$ as in (4.12).
3. Define $a_j^{(l+1)} = a_j^{(l)}$ and $b_j^{(l+1)} = b_j^{(l)}$ for $h \neq j$.
4. Form the partition $P_1^{(l+1)}, \ldots, P_k^{(l+1)}$ as in (4.10).
5. Quit if $P_j^{(l)} = P_j^{(l+1)}$ for $j = 1, \ldots, k$.

4.4.1 Interpretation

To gain some insight in this algorithm, we first make some observations on the value of J from the lth iteration to the next assuming we are only changing $a_1^{(l)}$ to $a_1^{(l+1)}$ and $b_1^{(l)}$ to $b_1^{(l+1)}$.

The contribution to the value of J associated with a point $i \notin P_1^{(l)}$ at iteration $(l+1)$

is

$$(f^{(l+1)} - y_i)^2 = (\max\{f^{(l)}(u_i), a_1^{(l+1)T} u_i + b_1^{(l+1)}\} - y_i)^2, \tag{4.13}$$

and for a point $i \in P_1^{(l)}$ is

$$(f^{(l+1)} - y_i)^2 = (\max\{a_1^{(l+1)T} u_i + b_1^{(l+1)}, \tilde{f}_1^{(l)}) - y_i\}^2. \tag{4.14}$$

The value of J at iteration $(l+1)$ is then given by

$$J = \sum_{i \in P_1^{(l)}} (\max\{a_1^{(l+1)T} u_i + b_1^{(l+1)}, \tilde{f}_1^{(l)}) - y_i\}^2 + \sum_{i \notin P_1^{(l)}} (\max\{f^{(l)}(u_i), a_1^{(l+1)T} u_i + b_1^{(l+1)}\} - y_i)^2.$$

We would like to choose $a_1^{(l+1)}$ and $b_1^{(l+1)}$ that minimize J. The problem is that both expression (4.13) and (4.14) are in general non convex in $a_1^{(l+1)}$ and $b_1^{(l+1)}$ and therefore the problem of minimizing J as a function of $a_1^{(l+1)}$ and $b_1^{(l+1)}$ is, in general, hard to solve.

We therefore seek convex functions of $a_1^{(l+1)}$ and $b_1^{(l+1)}$ that gives an upper bound of (4.13) and (4.14). It's easy to see that

$$((a_1^{(l+1)T} u_i + b_1^{(l+1)} - y_i - |f^{(l)}(u_i) - y_i|)_+ + |f^{(l)}(u_i) - y_i|)^2 \tag{4.15}$$

is convex as a function of $a_1^{(l+1)}$ and $b_1^{(l+1)}$ and is always greater or equal than (4.13). Notice that (4.15) is equal to (4.13) if $a_1^{(l+1)T} u_i + b_1^{(l+1)} \leq y_i + |f^{(l)}(u_i) - y_i|$ and in particular if $a_h^{(l+1)} = a_h^{(l)}$ and $b_h^{(l+1)} = b_h^{(l)}$. It's also easy to see that

$$\max\{(a_1^{(l+1)T} u_i + b_1^{(l+1)} - y_i)^2, (\tilde{f}_1^{(l)}(u_i) - y_i)_+^2\} \tag{4.16}$$

is convex as a function of $a_1^{(l+1)}$ and $b_1^{(l+1)}$ and is always greater or equal than (4.14). Notice that (4.16) is equal to (4.14) if $a_h^{(l+1)} = a_h^{(l)}$ and $b_h^{(l+1)} = b_h^{(l)}$.

It's then clear that at each step, the algorithm minimizes an upper bound of J over the parameters $a_h^{(l+1)}$ and $b_h^{(l+1)}$ by fixing all the other parameters. It's also important to notice that the upper bound of J is equal to J for $a_h^{(l+1)} = a_h^{(l)}$ and $b_h^{(l+1)} = b_h^{(l)}$. This is the key property to show the convergence of the algorithm.

From this derivation is clear that problem (4.12) is a convex problem and it can also

be rewritten as

$$\begin{array}{ll} \text{minimize} & \|t\|^2 \\ \text{subject to} & \tilde{f}_h^{(l)}(u_i) - y_i \leq t_i, \quad i \in P_h^{(l)}, \\ & -t_i \leq a^T u_i + b - y_i \leq t_i, \quad i \in P_h^{(l)}, \\ & a^T u_i + b - y_i \leq t_i, \quad i \notin P_h^{(l)}, \\ & -t_i \leq f^{(l)}(u_i) - y_i \leq t_i, \quad i \notin P_h^{(l)}, \end{array} \tag{4.17}$$

which is a quadratic program [BV04].

4.4.2 Convergence

It's easy to show that the optimal value of (4.3) will converge. In fact we have

$$\begin{aligned} J^{(l+1)} &= \sum_{i \in P_h^{(l)}} (f^{(l+1)}(u_i) - y_i)^2 + \sum_{i \notin P_h^{(l)}} (f^{(l+1)}(u_i))^2 \\ &\leq \sum_{i \in P_h^{(l)}} \max\{(a_h^{(l+1)T} u_i + b_h^{(l+1)} - y_i)^2, (\tilde{f}_h^{(l)}(u_i) - y_i)_+^2\} \\ &+ \sum_{i \notin P_h^{(l)}} ((a_h^{(l+1)T} u_i + b_h^{(l+1)} - y_i - |f^{(l)}(u_i) - y_i|)_+ + |f^{(l)}(u_i) - y_i|)^2 \\ &\leq \sum_{i \in P_h^{(l)}} \max\{(a_h^{(l)T} u_i + b_h^{(l)} - y_i)^2, (\tilde{f}_h^{(l)}(u_i) - y_i)_+^2\} \\ &+ \sum_{i \notin P_h^{(l)}} ((a_h^{(l)T} u_i + b_h^{(l)} - y_i - |f^{(l)}(u_i) - y_i|)_+ + |f^{(l)}(u_i) - y_i|)^2 \\ &= \sum_{u_i} (f^{(l)}(u_i) - y_i)^2 \\ &= J^{(l)}. \end{aligned}$$

Therefore at each iteration the value of J can only decrease and will therefore converge.

4.4.3 Connection with the simple partition algorithm

We should notice that, if the given data can be interpolated by a convex function the solution of (4.12) is be the same as the one of

$$\text{minimize} \quad \textstyle\sum_{i \in P_h^{(l)}} \max\{(a^T u_i + b - y_i)^2, (\tilde{f}_h^{(l)}(u_i) - y_i)_+^2\}, \tag{4.18}$$

provided this problem has a unique solution. This minimization problem is performed only locally on the points of the set $P_h^{(l)}$ as for the simple partition algorithm. Moreover, as long as $\tilde{f}_h^{(l)}(u_i) - y_i \leq 0$ problem (4.18) reduces to

$$\text{minimize} \quad \textstyle\sum_{i \in P_h^{(l)}} (a^T u_i + b - y_i)^2,$$

which is the same as the minimization performed in the simple partition algorithm. This condition is usually satisfied if the data can be interpolated with a convex function. This shows that under certain conditions the improved and simple partition algorithm are identical.

Finally we should notice that the starting point and the other techniques introduced for the simple algorithm can be used for the improved partition algorithm.

4.5 Numerical examples

In this section we show some numerical results, using the following data set. The dimension is $n = 3$, and we have $m = 11^3 = 1331$ points. The set of points u_i is given by $\mathcal{V} \times \mathcal{V} \times \mathcal{V}$, where $\mathcal{V} = \{-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5\}$. The values are obtained as $y_i = g(u_i)$, where g is the (convex) function

$$g(x) = \log(\exp x_1 + \exp x_2 + \exp x_3).$$

We use the piecewise-linear fitting algorithm described in §4.3.4, with iteration limit $l_{\max} = 50$, and number of terms varying from $k = 0$ to $k = 20$. For $k = 0$, the fitting function is taken to be zero, so we report the RMS value of $y_1, \ldots, y_m$ as the RMS fit. For $k = 1$, the fit is the best affine fit to the data, which can be found using least-squares. Figure 4.5 shows the RMS fits obtained after $N_{\text{trials}} = 10$ trials (top curve), and after $N_{\text{trials}} = 100$ trials (bottom curve). These show that good fits are obtained with only 10 trials, and that (slightly) better ones are obtained with 100 trials.

To give an idea of the variation in RMS fit obtained with different trials, as well as the number of steps required for convergence (if it occurs), we fix the number of terms at $k = 12$, and run the least-squares partition algorithm 200 times, with a limit of 50 iterations, recording both the final RMS fit obtained, and the number of steps before convergence. (The number of steps is reported as 50 if the least-squares partition

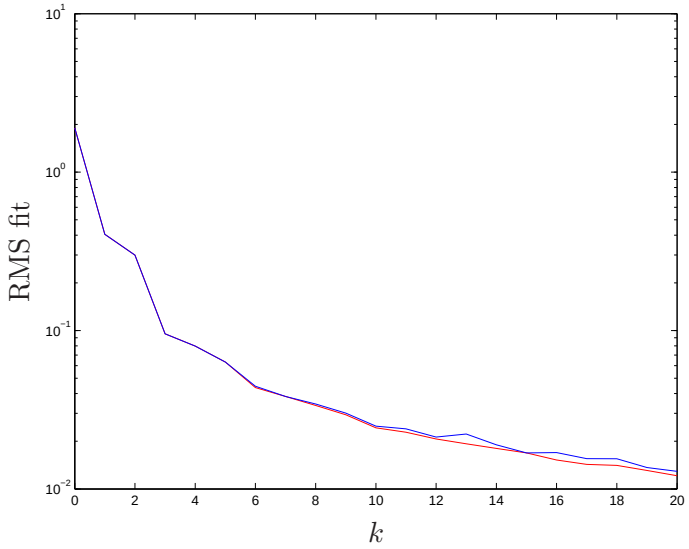

Figure 4.1: Best RMS fit obtained with 10 trials (top curve) and 100 trials (bottom curve), versus number of terms k in max-affine function.

algorithm has not converged in 50 steps.) Figure 4.5 shows the histogram of RMS fit obtained. We can see that the fit is often, but not always, quite good; in just a few cases the fit obtained is poor. Evidently the best of even a modest number of trials will be quite good.

Figure 4.5 shows the distribution of the number of iterations of the least-squares partition algorithm required to converge. Convergence failed in 13 of the 200 trials; but in fact, the RMS fit obtained in these trials was not particularly bad. Typically convergence occurs within around 25 iterations.

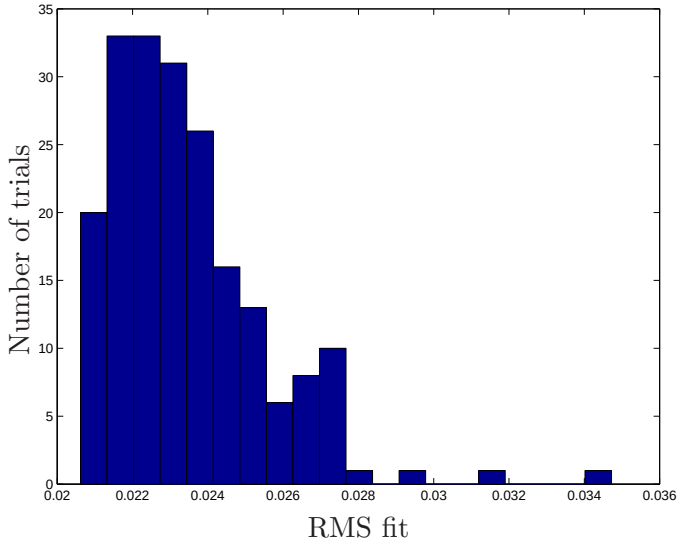

Figure 4.2: Distribution of RMS fit obtained in 200 trials of least-squares partition algorithm, for $k = 12$, $l_{\max} = 50$.

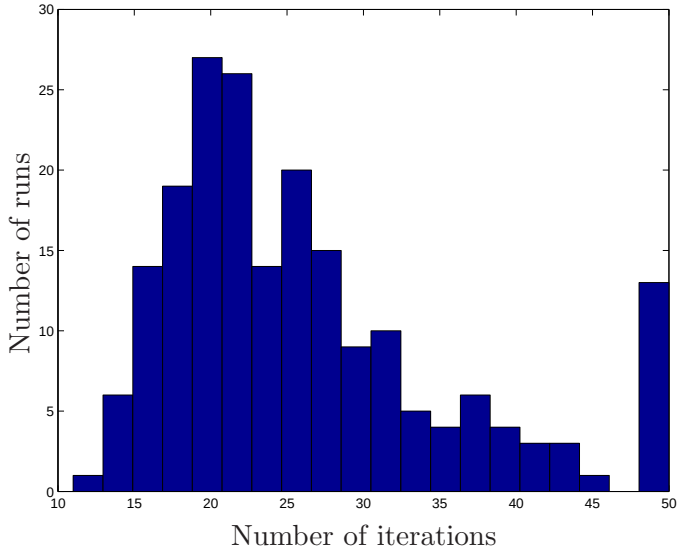

Figure 4.3: Distribution of the number of steps required by least-squares partition algorithm to converge, over 200 trials. The number of steps is reported as 50 if convergence has not been obtained in 50 steps.

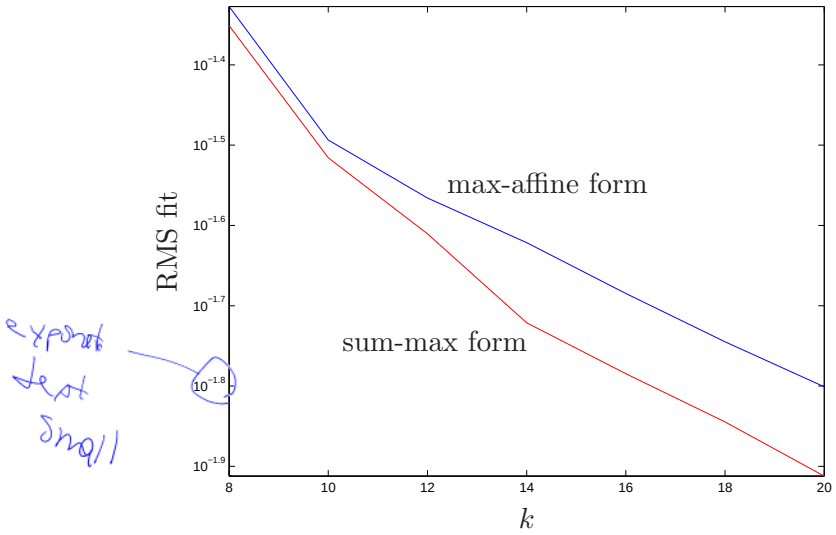

Figure 4.4: Best RMS fit obtained for max-affine function (top) and sum-max function (bottom).

In our last numerical example, we compare fitting the data with a max-affine function with k terms, and with the more general form

$$f(x) = \max_{i=1,\ldots,k/2} \left(a_i^T x + b_i\right) + \max_{i=k/2+1,\ldots,k} \left(a_i^T x + b_i\right),$$

parametrized by $a_1, \ldots, a_k \in \mathbf{R}^n$ and $b_1, \ldots, b_k \in \mathbf{R}$. (Note that the number of parameters in each function form is the same.) This function corresponds to the general form (4.4) with

$$\psi(z_1, \ldots, z_k) = \max_{i=1,\ldots,k/2} z_i + \max_{i=k/2+1,\ldots,k} z_i,$$

and

$$\phi(x, \alpha) = (a_1^T x + b_1, \ldots, a_k^T x + b_k).$$

We set the iteration limit for both forms as $l_{\max} = 100$, and take the best fit obtained in $N_{\text{trials}} = 10$ trials. We use the value $\rho = 10^{-5}$ for the regularization parameter in the general form algorithm. Figure 4.4 shows the RMS fit obtained for the two forms, versus k. Evidently the sum-max form gives (slightly) better RMS fit than the max-affine form.

Chapter 5

Convex polynomial fitting

5.1 Introduction

We consider the problem of fitting given data

$$(u_1, y_1), \quad \ldots, \quad (u_m, y_m)$$

with $u_i \in \mathbf{R}^n$ and $y_i \in \mathbf{R}$ with a convex polynomial f.

Given polynomials $p_1, \ldots, p_w$ in n variables, we restrict the polynomials we are considering so that f has the form

$$f = c_1 p_1 + \cdots + c_w p_w,$$

where c_i for $i = 1, \ldots, w$, are reals. We would like to choose variables $c = (c_1, \ldots, c_w)$. For example, this description of f allows us to describe the set of polynomials of degree less than a constant or the polynomials of a specific degree.

Using least-squares fitting we obtain the problem

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m} (f(u_i) - y_i)^2 \\ \text{subject to} & f \quad \text{is convex,} \end{array} \tag{5.1}$$

One may also consider other norms but we will use the above formulation in this chapter.

In some special cases the solution to this problem is known. If, for example, the polynomials p_i are such that f is affine in x and therefore convex, we have that f has

the form $f = c_1 + c_2 x_1 + \cdots + c_{n+1} x_n$ and the problem becomes

$$\text{minimize} \quad \sum_{i=1}^{m} (f(u_i) - y_i)^2.$$

This is a least-squares problem with variable c_i and it has an analytical solution.

If instead the polynomials p_i have degree less than or equal to 2 then f is a quadratic form and can be written as

$$f(x) = x^T A x + b^T x + r,$$

where A, b, and r linearly depend on c. Since imposing the convexity of f is equivalent to imposing A to be positive semidefinite, the problem becomes

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m} (f(u_i) - y_i)^2 \\ \text{subject to} & f(x) = x^T A x + b^T x + r, \\ & A \succeq 0. \end{array}$$

This problem is a *semidefinite program* (SDP) [BV04] with variables A, b, and r and can be solved efficiently.

In the general case, if we consider the set $\mathcal{C}$ of coefficients such that f is convex

$$\mathcal{C} = \{c \mid f = c_1 p_1 + \cdots + c_d p_w, f \text{ is convex}\},$$

problem (5.1) can be rewritten as

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m} (f(u_i) - y_i)^2 \\ \text{subject to} & c \in \mathcal{C}. \end{array} \tag{5.2}$$

Since the set $\mathcal{C}$ is convex, this is a convex optimization problem. Nevertheless, since there is no known tractable description of the set $\mathcal{C}$ in general and so the problem is hard to solve.

We will consider a subset of $\mathcal{C}$ so that the problem becomes tractable. We will also show conditions under which one can solve the original problem exactly.

5.2 Convex polynomials via SOS

We first consider the problem of imposing convexity on a generic polynomial f in n variables of the form $f = c_1 p_1 + \cdots + c_w p_w$ where p_i, $i = 1, \ldots, w$, are given polynomials in n variables, $c = (c_1, \ldots, c_w) \in \mathbf{R}^w$, and d is the degree of f.

We know that a necessary and sufficient condition for f to be convex is that

$$h = s^T \nabla^2 f(x) s \geq 0 \quad \text{for all } x, s. \tag{5.3}$$

Notice that h is a polynomial expression with variables s and x and moreover is of the same degree d as f.

A polynomial $g(t)$ such that $g(t) \geq 0$ for all $t \in \mathbf{R}^n$ is called *positive semidefinite* (PSD). Therefore f is convex if and only if h is PSD.

Except for special cases (*e.g.*, $n = 1$ or $d = 2$), it is NP-hard to determine whether or not a given polynomial is PSD, let alone solve an optimization problem, with the coefficients of c as variables, with the constraint that h is PSD.

A famous sufficient condition for a polynomial to be PSD is that it has the form

$$g(x) = \sum_{i=1}^{r} q_i(x)^2,$$

for some polynomials q_i, with degree no more than $d/2$. A polynomial g that has this sum-of-squares form is called SOS.

The condition that a polynomial g be SOS (viewed as a constraint on its coefficients) turns out to be equivalent to an *linear matrix inequality* (LMI) ([Nes00, Par00]). In particular a polynomial g of even degree w is SOS if and only if there exist monomials of degree less that $d/2$, $e_1, \ldots, e_s$ and a positive semidefinite matrix V such that

$$g = e^T V e. \tag{5.4}$$

Since the condition $g = e^T V e$ is a set of linear equality constraints relating the coefficients of g to the elements of V, we have that imposing the polynomial g to be SOS is equivalent to the positive semidefiniteness constraint that $V \succeq 0$ together with a set of linear equality constraints.

We will impose convexity on the polynomial f by requiring h to be SOS. We then

clearly have

$$\mathcal{S} = \{c \mid h \text{ is SOS}\} \subseteq \mathcal{C}.$$

Since the condition of a polynomial being PSD is not equivalent to being SOS, in general $\mathcal{C} \neq \mathcal{S}$ and therefore by imposing h to be SOS, we are not considering all the possible convex polynomials but only a subset of them. Only in special cases does $\mathcal{S} = \mathcal{C}$, for example if $n = 1$ or $d = 2$.

As mentioned above, the advantage of h being SOS is that imposing this constraint can be cast as LMI and handled efficiently [BGFB94].

5.3 Function fitting via SOS

Using the approximation of the previous section to solve problem (5.1), we obtain

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m}(f(u_i) - y_i)^2 \\ \text{subject to} & c \in \mathcal{S}. \end{array} \tag{5.5}$$

Equivalently, using the necessary and sufficient condition for a polynomial to be SOS, we obtain the problem

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m}(f(u_i) - y_i)^2 \\ \text{subject to} & h = s^T \nabla^2 f(x) s = e^T V e \quad \text{for all } x, s \\ & V \succeq 0, \end{array} \tag{5.6}$$

where e is a vector of monomials in s and x and the variables are the matrix V and c. Since the equation $h = e^T V e$ is simply a set of linear equations in the coefficients of V and c, this problem can be cast as a semidefinite program for which there are efficient algorithms [BV04, VB96].

5.3.1 Numerical example

We present a very simple example for $n = 1$, where the data u_i for $i = 1, \ldots, 100$, is obtained by uniformly sampling the interval $[-5, 5]$ and $y_i = \exp(u_i)$. In this case, since $\mathcal{S} = \mathcal{C}$ we can tractably solve problem (5.1). Figure 5.1 shows an example, where stars correspond to given data points.

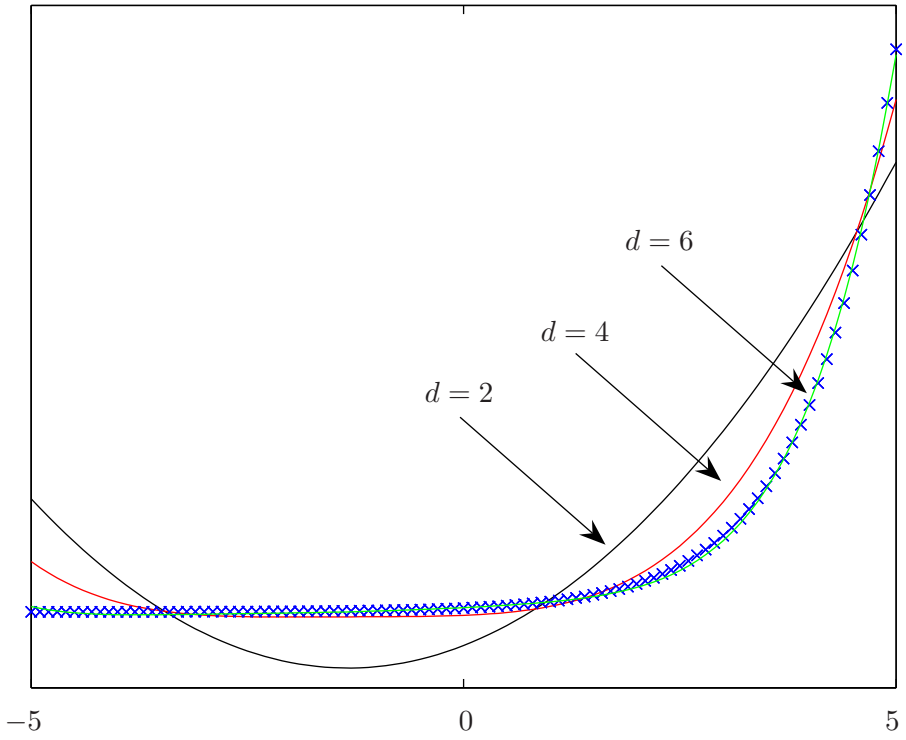

Figure 5.1: Convex polynomial fitting example.

5.4 Minimum volume set fitting

In this section we address the problem of finding a convex set P, described through a sub-level set of a convex polynomial g, that contains a set of points and is close in some sense to them. We would like, for example, to find the minimum volume set P that includes all points u_i.

As before, given polynomials $p_1, \ldots, p_w$ we restrict ourselves to consider a polynomial g of the form

$$g = b_1 p_1 + \cdots + b_w p_w,$$

where we would like to choose $b \in \mathbf{R}^w$. Therefore we want to solve the problem

$$\begin{array}{ll} \text{minimize} & \text{volume}(P) \\ \text{subject to} & P = \{x \mid g(x) \leq 1\} \\ & u_i \in P \quad \text{for all } i = 1, \ldots, m, \\ & P \text{ is convex.} \end{array} \tag{5.7}$$

If for example g is a polynomial of degree 2, P will be the minimum volume ellipsoid containing all the data points. This is a well-known problem [BV04] and if we write g as $g = x^T A x + b^T x + r$ we then g is convex if and only if A is positive semidefinite. The above problem then becomes

$$\begin{array}{ll} \text{minimize} & \text{volume}(P) \\ \text{subject to} & u_i^T A u_i + b^T u_i + r \leq 1, \quad i = 1, \ldots, m \\ & A \succeq 0. \end{array}$$

We can assume without loss of generality that $g(x) \geq 0$ for all x, in which case the volume of P is proportional to $\sqrt{\det A^{-1}}$ and we can write the problem as

$$\begin{array}{ll} \text{minimize} & \log\det A^{-1} \\ \text{subject to} & u_i^T A u_i + b^T u_i + r \leq 1, \quad i = 1, \ldots, m \\ & A \succeq 0 \\ & \left[\begin{array}{cc} A & b \\ b^T & r \end{array} \right] \succeq 0, \end{array}$$

where the last constraint is equivalent to $g(x) \geq 0$ for all x. This problem can be cast

as an SDP [NN95].

In the general case the problem can be written as

$$\begin{array}{ll} \text{minimize} & \text{volume}(P) \\ \text{subject to} & u_i \in P \quad \text{for } i = 1, \ldots, m, \\ & h = s^T \nabla^2 g(x) s \geq 0 \quad \text{for all } x, s. \end{array}$$

Now not only is the second constraint hard to handle exactly, but there is also no known way to efficiently compute the volume of P. We propose a heuristic algorithm that tries to shrink the set P around the data points and that for $d = 2$ is equivalent to the minimum volume ellipsoid. This problem has possible applications in data mining [KNZ01, LN95], robotics, and computer vision [KG99, TCS+94, RB97].

5.4.1 Pseudo minimum volume heuristic

The main idea of the algorithm is to increase the curvature of g along all directions so that the set P gets closer to the points u_i. Since the curvature of g along the direction s is proportional to

$$h = s^T \nabla^2 g(x) s,$$

we will write h in a specific form so that we can, at the same time, enforce h to be PSD and increase the curvature of g.

The first step, as before, is to impose

$$h = s^T \nabla^2 g(x) s \quad \text{is SOS},$$

or equivalently

$$\begin{array}{l} h = s^T \nabla^2 g(x) s = e^T V e \quad \text{for all } x, s \\ V \succeq 0, \end{array}$$

where e is a vector of monomials in x and s. In this way we have that g is convex. Similarly to the case of the minimum volume ellipsoid, we maximize the determinant of V which has the effect of increasing the curvature of g along all possible directions.

The heuristic becomes

$$\begin{array}{ll} \text{minimize} & \log\det V^{-1} \\ \text{subject to} & g(x) \geq 0 \quad \text{for all } x, \\ & s^T \nabla^2 g(x) s = e^T V e \quad \text{for all } x, s \\ & g(u_i) \leq 1 \quad i = 1, \ldots, m. \end{array}$$

Again replacing the positivity constraint $g(x) \geq 0$ by an SOS condition, we arrive at

$$\begin{array}{ll} \text{minimize} & \log\det V^{-1} \\ \text{subject to} & g \text{ is SOS} \\ & s^T \nabla^2 g(x) s = e^T V e \quad \text{for all } x, s \\ & g(u_i) \leq 1 \quad i = 1, \ldots, m, \end{array}$$

or equivalently

$$\begin{array}{ll} \text{minimize} & \log\det V^{-1} \\ \text{subject to} & g = h^T C h, \\ & C \succeq 0, \\ & s^T \nabla^2 g(x) s = e^T V e \quad \text{for all } x, s \\ & g(u_i) \leq 1 \quad i = 1, \ldots, m, \end{array} \tag{5.8}$$

where h is a vector of monomials in x and e is a vector of monomials in x and s. This problem can now be solved efficiently.

It is clear that for $d = 2$, the problem reduces to finding the minimum volume ellipsoid. Note that the matrix C is not unique and it depends on the choice of monomials e. It is also possible for the heuristic to fail; for example, if we choose a redundant set of monomials for e, then C may not be full rank and the determinant of C will be zero. One workaround for this is to use fewer monomials for e. Moreover we should notice that it is not strictly needed for e to be made out of monomials but any polynomial expression would work.

It can be shown (see Appendix) that, under some minor conditions, the solution to this problem has the nice property of being invariant to affine coordinate transformations. In other words, if P is the solution of the problem, by changing the coordinates of the points u_i through an affine transformation, we would have that the set P scaled by the same transformation, is an optimal point for the problem in the new set of coordinates.

Example

We show in a simple case how to derive the matrices C and V for problem (5.8). Suppose g has the form

$$g(x, y) = c_1 + c_2 x^4 y^4,$$

we can choose the vectors of monomials h as

$$h = (1, x^2 y^2).$$

With this choice of h the matrix C will be

$$C = \begin{bmatrix} c_1 & 0 \\ 0 & c_2 \end{bmatrix}.$$

We then have

$$h = s^T \nabla^2 g(x) s = 12 c_3 x^2 y^4 s_1^2 + 12 c_3 x^4 y^2 s_2^2 + 32 x^3 y^3 s_1 s_2,$$

and by picking the vector of monomials e to be

$$e = (x y^2 s_1, x^2 y s_2),$$

we obtain

$$h = e^T V e = e^T \begin{bmatrix} 12c_3 & 16c_3 \\ 16c_3 & 12c_3 \end{bmatrix} e.$$

Numerical example

We show the result of the algorithm for a set of points corresponding to the simulated first 150 steps of a dynamical system. We pick g to be a generic polynomial of degree less than d. Figure 5.2 shows the level set for different degrees d of the polynomial g.

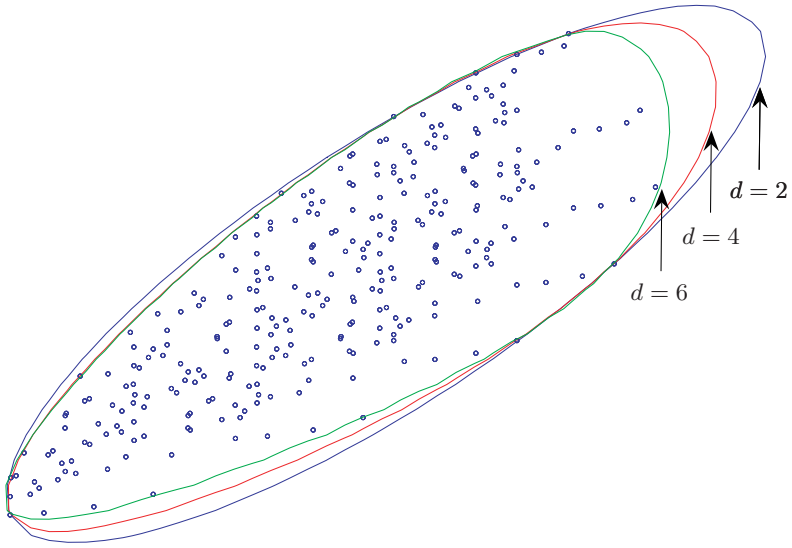

Figure 5.2: Pseudo minimum volume example.

5.5 Conditional convex polynomial fitting

In this section we want to solve problem (5.1) with the relaxed constraint that f is convex only over a set P and not on the entire space

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m}(f(u_i)-y_i)^2 \\ \text{subject to} & f \quad \text{is convex on } P. \end{array} \tag{5.9}$$

We require that the set P contains the points u_i and that is convex. Moreover the set P should be described as the sub-level set of a polynomial g

$$P = \{x \mid g(x) \leq 1\}.$$

For example, the previously presented algorithm gives us a set P with the required properties that can be used to solve problem (5.9).

We will write a sufficient condition for f to be convex over the set P. We will show that for P compact this condition has a nice property that allows to prove a stronger result. For $h = s^T \nabla^2 f s$ we define

$$l(x,s) = h(x,s) + w(x,s)(1-g(x)), \tag{5.10}$$

where w is a sum of squares polynomial. It is clear that if l is SOS then the function f will be convex over the set P. Vice versa it can be shown [Sch91] that if P is compact and h is strictly positive over P, there exist SOS polynomials l and w so that (5.10) holds.

Therefore, by using this condition to impose convexity of f over P, the problem becomes

$$\begin{array}{lll} \text{minimize} & \sum_{i=1}^{m}(f(u_i)-y_i)^2 & \\ \text{subject to} & s^T \nabla^2 f s - w(x,s)(1-g(x)) & \text{is SOS} \\ & w \quad \text{is SOS.} & \end{array}$$

Notice that we have a wide range of choice for the polynomial w since the only constraint is that it should be SOS. Therefore we cannot solve this problem because to describe the polynomial w we would need an infinite number of variables. Nevertheless we should notice that if we were able to solve this problem and P was compact, we would be able

to find a polynomial for which the cost function is no greater than the optimal value of

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m}(f(u_i)-y_i)^2 \\ \text{subject to} & f \quad \text{is strictly convex on } P. \end{array} \tag{5.11}$$

To make this problem tractable we can, for example, impose the maximum degree of w to be less than a given constant t. In this case, w will have the form

$$w = h^T W h,$$

where h is a vector of all monomials of degree less or equal than $t/2$ and W is a generic positive semidefinite matrix.

Once we fix the order of the polynomial w, the problem can be cast as a convex program (SDP) and solved efficiently. We obtain the problem

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{m}(f(u_i)-y_i)^2 \\ \text{subject to} & s^T\nabla^2 f s - w(x,s)(1-g(x)) \quad \text{is SOS} \\ & w = h^T W h, \\ & W \succeq 0, \end{array}$$

where the variables are c and W and h is a vector of monomials of degree less or equal than $t/2$. By increasing t we obtain larger problems that in the limit tend to a solution for which the cost function is not greater than the optimal value of problem (5.11).

5.5.1 Numerical example

We solve the same numerical example presented in section 5.3.1 but imposing convexity only on the interval $[-5, 5]$. In this way we can, for example, fit using odd degree polynomials. We describe the interval with $g(x) = x^2 - 24$ and we fix the degree of w to be less or equal than 4. In particular figure 5.3 shows the result for a third and fifth order polynomial. Clearly the function is not convex on $\mathbf{R}$ but it is still convex on the interval $[-5, 5]$.

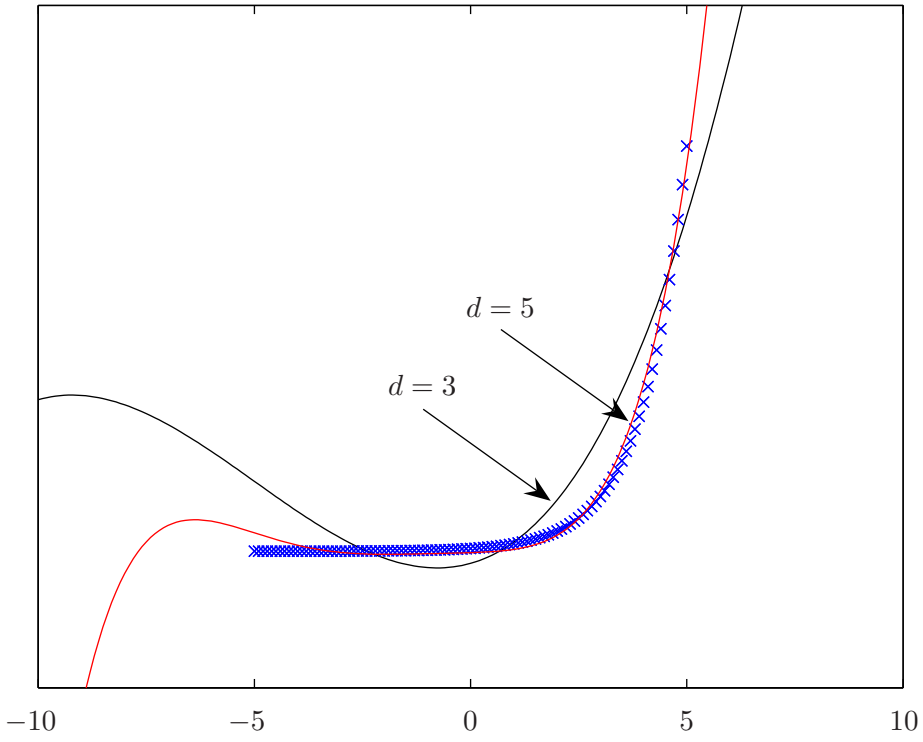

Figure 5.3: Conditional convex polynomial fitting.

5.6 Extensions

We present two simple extensions. The first one allows to fit a set of points described through the intersection of two sub-level sets of polynomials. The second one extends the results of this chapter to a different class of polynomials.

5.6.1 Fitting the intersection of sub-level sets of polynomials

One simple extension of the pseudo minimum volume heuristic is to find a convex set P that fits a set of the form

$$K = \{x \mid f_i(x) \leq 0 \quad i = 1, 2\},$$

where f_i are polynomials.

We can write a sufficient condition for P to contain K. In particular, if we have

$$(1 - g(u_i)) + w_1 f_1 + w_2 f_2 - w_3 f_1 f_2 \quad \text{is SOS}, \tag{5.12}$$

where w_i are SOS, clearly the set P will contain the set K. It can also be shown [Sch91] that if K is compact and P is such that $\hat{x} \in K$ implies $g(\hat{x}) < 1$ then there exist SOS polynomials w_i such that (5.12) is verified.

The heuristic can be modified to impose $K \subseteq P$ as

$$\begin{array}{ll} \text{minimize} & \log\det C^{-1} \\ \text{subject to} & g \text{ is SOS} \\ & s^T \nabla^2 g(x) s = e^T C e \quad \text{for all } x, s \in \mathbf{R}^n \\ & (1 - g(u_i)) + \sum_i w_i f_i - w_3 f_1 f_2 \quad \text{is SOS}, \\ & w_1, w_2, w_3 \quad \text{are SOS}. \end{array}$$

To be able to solve this problem, we need to impose some more constraints on the polynomials w_i since the only constraint is that they should be SOS. As we did before, we can impose them to have a maximum degree less than some constant, and the resulting optimization problem is an SDP. With similar techniques one may also also easily handle union and intersection of such sets.

5.6.2 Convexity along a line passing through a point

We can extends the techniques presented in this chapter to a different class of polynomials [KG99], that are convex only when restricted to a line passing for a given point x_0.

Given a polynomial f and a point x_0, the property is equivalent to

$$h(x) = (x - x_0)^T \nabla^2 f(x)(x - x_0) \geq 0 \quad \text{for all } x.$$

In other words we are replacing the generic direction s in (5.3) along which the curvature of the polynomial is evaluated, with the direction $x - x_0$ that goes through the point x_0.

We can therefore apply the function fitting algorithm (5.5) and the pseudo minimum volume algorithm (5.8) for polynomials with this property by simply substituting s with $x - x_0$. We should point out that in this case the algorithm loses the property of being invariant under an affine coordinate transformation.

Appendix A

A.1 Affine coordinate transformation invariance

Given problem (5.8), we would like to show the relationship between the solutions of it for two different systems of coordinates x, y such that $x = Ay + b$ where $\det A \neq 0$. In particular we have that, if u_i, v_i for $i = 1, \ldots, m$ are the points in the first and second system of coordinates respectively,

$$u_i = Av_i + b.$$

We will use subscript x to refer to the problem with the x coordinates and a y subscript for the problem in the y coordinates. We call, for example, e_x and h_x the vectors e and h in the first system of coordinates and e_y and h_y in the second. We also call $\hat{e}_y$ and $\hat{h}_y$ the vectors e_x and h_x where each component as been transformed in the other system of coordinates so that $x = Ay + b$ and $s_x = As_y$. Therefore each component of $\hat{h}_y$, for example, is a polynomial in y and $\hat{e}_y$ depends only on y and s_y. The same holds for $\hat{e}_x$ and $\hat{h}_x$ which are the vectors e_y and h_y in the other system of coordinates.

We make the assumption that the vector $\hat{e}_y$ can be represented as a linear combination of e_y and that $\hat{e}_x$ is a linear combination of e_x. Moreover we require the same property for the vectors h_x and h_y. This assumption is satisfied, for example, if h_x consists of all monomials up to a certain degree in x and the same choice is made for h_y. In other words, we require that in the two systems of coordinates, we can describe the same set of polynomials.

Given this property we have that

$$\hat{h}_y = U_1 h_y,$$
$$\hat{e}_y = U_2 e_y,$$

where U_1 and U_2 are matrices that depend nonlinearly on A and b. So suppose that $g(x)$ is a feasible solution for the problem in the x coordinates. We will show that the polynomial $f(y) = g(Ay + b)$ is feasible in the second system of coordinates.

We have that

$$f(v_i) = g(Av_i + b) = g(u_i) \leq 1,$$

and therefore the points are included in the sub-level set. We also have that

$$\begin{aligned} f(y) &= g(Ay + b) \\ &= \hat{h}_y^T C_x \hat{h}_y \\ &= h_y^T U_1^T C_x U_1 h_y, \end{aligned}$$

where clearly $C_y = U_1^T C_x U_1 \succeq 0$, and

$$\begin{aligned} s_y^T \nabla^2 f(y) s_y &= s_y^T \nabla^2 g(Ay + b) s_y \\ &= s_y^T A^T \nabla_x^2 g(Ay + b) A s_y \\ &= \hat{e}_y^T V_x \hat{e}_y \\ &= e_y^T U_2^T V_x U_2 e_y, \end{aligned}$$

where $V_y = U_2^T V_x U_2 \succeq 0$. It is also clearly true that

$$\log\det V_x^{-1} = 2\log\det U_2 + \log\det V_y,$$

in other words the same polynomial after a coordinate transformation is still feasible for the second problem and produce a value which is the same except for a constant independent of the polynomial. Since the same applies in the other direction, we can conclude that an optimal solution to the first problem will be optimal for the second one too.

Bibliography

[ABO+85] A. Aggarwal, H. Booth, J. O'Rourke, S. Suri, and C. Yap. Finding minimal convex nested polygons. In *Proceedings of the symposium on Computational geometry*, pages 296–304. ACM Press, 1985.

[AL00] N. Alexandrov and R. Lewis. Analytical and computational aspects of collaborative optimization. Technical Report 2000-210104, NASA, 2000.

[AWV05] A. Agarwal, G. Wolfe, and R. Vemuri. Accuracy driven performance macromodeling of feasible regions during synthesis of analog circuits. In *GLSVSLI '05: Proceedings of the 15th ACM Great Lakes symposium on VLSI*, pages 482–487, New York, NY, USA, 2005. ACM Press.

[BBM02] A. Bemporad, F. Borrelli, and M. Morari. Model Predictive Control Based on Linear Programming - The Explicit Solution. *IEEE Transactions on Automatic Control*, 47(12):1974–1985, December 2002.

[Ber99] D. Bertsekas. *Nonlinear Programming*. Athena Scientific, second edition, 1999.

[BGFB94] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*. SIAM, Philadelphia, 1994.

[BJV03] F. De Bernardinis, M. Jordan, and A. Vincentelli. Support vector machines for analog circuit performance representation. In *DAC '03: Proceedings of the 40th conference on Design automation*, pages 964–969, New York, NY, USA, 2003. ACM Press.

[BK97] R. Braun and I. Kroo. Development and application of the collaborative optimization architecture in a multidisciplinary design environment. *Multidisciplinary Design Optimization: State of the Art*, pages 98–116, 1997.

[BK05] S. Boyd and S. Kim. Geometric programming for circuit optimization. In *Proceedings of International Symposium on Physical Design.*, pages 44–46, 2005.

[BKVH06] S. Boyd, S. Kim, L. Vandenberghe, and A. Hassibi. A tutorial on geometric programming. *Optimization and Engineering*, 2006.

[BNV05] F. De Bernardinis, P. Nuzzo, and A. Vincentelli. Mixed signal design space exploration through analog platforms. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 875–880, New York, NY, USA, 2005. ACM Press.

[BR69] R. Bellman and R. Roth. Curve fitting by segmented straight lines. *American Statistical Association Journal*, 64:1079–1084, 1969.

[BS96] B. Bakshi and G. Stephanopoulos. Compression of chemical process data by functional approximation and feature extraction. *AIChE Journal*, 42(2):477–492, January 1996.

[BT89] D. Bertsekas and J. Tsitsiklis. *Parallel and distributed computation: numerical methods*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CCC+97] H. Chang, E. Charbon, U. Choudhury, A. Demir, E. Felt, E. Liu, A. Malvasi, A. Vincentelli, and I. Vassiliou. *A Top-Down, Constraint-Driven Design Methodology for Analog Integrated Circuits*. Kluwer Academic Publishers, 1997.

[CD86] L. Chua and A. Deng. Canonical piecewise-linear modeling. *IEEE Transactions on Circuits and Systems*, 33(5):511–525, May 1986.

[DDNAV04] A. Doboli, N. Dhanwada, A. Nunez-Aldana, and R. Vemuri. A two-layer library-based approach to synthesis of analog systems from vhdl-ams specifications. *ACM Trans. Des. Autom. Electron. Syst.*, 9(2):238–271, 2004.

[DGS+96] S. Donnay, G. Gielen, W. Sansen, W. Kruiskamp, D. Leenaerts, S. Buytaert, K. Marent, M. Buckens, and C. Das. Using top-down CAD tools for mixed analog/digital ASICs: a practical design case. *Analog Integr. Circuits Signal Process.*, 10(1-2):101–117, 1996.

[DGS05] W. Daems, G. Gielen, and W. Sansen. Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits. *Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(5):517–534, 2005.

[DLTW90] D. Dobkin, S. Levy, W. Thurston, and A. Wilks. Contour tracing by piecewise linear approximations. *ACM Transactions on Graphics*, 9(4):389–423, October 1990.

[dMH02] M. del Mar Hershenson. Design of pipeline analog-to-digital converters via geometric programming. In *ICCAD '02: Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 317–324, New York, NY, USA, 2002. ACM Press.

[DPZ67] R. Duffin, E. Peterson, and C. Zener. *Geometric Programming–Theory and Application*. Wiley, New York, 1967.

[Dun86] J. Dunham. Optimum uniform piecewise linear approximation of planar curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):67–75, 1986.

[EM73] J. Elzinga and T. Moore. Computational experience with the central cutting plane algorithm. In *ACM'73: Proceedings of the annual conference*, pages 448.5–456. ACM Press, 1973.

[EMG05] T. Eeckelaert, T. McConaghy, and G. Gielen. Efficient multiobjective synthesis of analog circuits using hierarchical pareto-optimal performance hypersurfaces. In *DATE '05: Proceedings of the conference on Design, Automation and Test in Europe*, pages 1070–1075, Washington, DC, USA, 2005. IEEE Computer Society.

[FTM02] G. Ferrari-Trecate and M. Muselli. A new learning method for piecewise linear regression. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 444–449. Springer-Verlag, 2002.

[GDD02] G. Gothoskar, A. Doboli, and S. Doboli. Piecewise-linear modeling of analog circuits based on model extraction from trained neural networks. In *Proceedings of IEEE International Workshop on Behavioral Modeling and Simulation*, pages 41–46, 2002.

[GG91] A. Gersho and R. Gray. *Vector quantization and signal compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.

[GHV92] J. Goffin, A. Haurie, and J. Vial. Decomposition and nondifferentiable optimization with the projective algorithm. *Management Science*, 38:284–302, 1992.

[GHVZ93] J. Goffin, A. Haurie, J. Vial, and L. Zhu. Using central prices in the decomposition of linear programs. *European Journal of Operation Research*, 64:393–409, 1993.

[GLY96] J. Goffin, Z. Luo, and Y. Ye. Complexity analysis of an interior cutting plane method for convex feasibility problems. *SIAM Journal on Optimization*, 6(3):638–652, 1996.

[GME05] G. Gielen, T. McConaghy, and T. Eeckelaert. Performance space modeling for hierarchical synthesis of analog integrated circuits. In *DAC '05: Proceedings of the 42nd annual conference on Design automation*, pages 881–886, New York, NY, USA, 2005. ACM Press.

[Goo94] M. Goodrich. Efficient piecewise-linear function approximation using the uniform metric: (preliminary version). In *Proceedings of the tenth annual*

symposium on Computational geometry, pages 322–331, New York, NY, USA, 1994. ACM Press.

[GV90] J. Goffin and J. Vial. Cutting planes and columm generation techniques with the projective alg orithm. *Journal of Optimization Theory and Applications*, pages 409–429, 1990.

[GV99] J. Goffin and J. Vial. Convex nondifferentiable optimization: A survey focussed on the analytic center cutting plane mathod. Technical Report 99.02, McGill University, Canada, February 1999.

[HB97] J. Horst and I. Beichel. A simple algorithm for efficient piecewise-linear approximation of space curves. In *Proceedings of International Conference on Image Processing.* IEEE Computer Society, 1997.

[HBL98] M. Hershenson, S. Boyd, and T. Lee. GPCAD: a tool for CMOS op-amp synthesis. In *ICCAD '98: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 296–303, New York, NY, USA, 1998. ACM Press.

[HS91] S. Hakimi and E. Schmeichel. Fitting polygonal functions to a set of points in the plane. *CVGIP: Graph. Models Image Process*, 53(2):132–136, 1991.

[HS96] R. Harjani and J. Shao. Feasibility and performance region modeling of analog and digital circuits. *Analog Integr. Circuits Signal Process.*, 10(1-2):23–43, 1996.

[II86] H. Imai and M. Iri. An optimal algorithm for approximating a piecewise linear function. *Journal of Information Processing*, 9(3):159–162, 1986.

[JJD98] P. Julian, M. Jordan, and A. Desages. Canonical piecewise-linear approximation of smooth functions. *IEEE Transactions on Circuits and Systems*, 45(5):567–571, May 1998.

[KC78] S. Kang and L. Chua. A global representation of multidimensional piecewise-linear functions with linear partitions. *IEEE Transactions on Circuits and Systems*, 25:938–940, 1978.

[KC90] C. Kahlert and L. Chua. A generalized canonical piecewise-linear representation. *IEEE Transactions on Circuits and Systems*, 37(3):373–383, March 1990.

[Kel60] J. Kelley. The cutting plane method for solving convex programs. *Journal of the SIAM*, 8:703–712, 1960.

[KG99] D. Keren and C. Gotsman. Fitting curves and surfaces with constrained implicit polynomials. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(1):31–41, 1999.

[KLVY04] J. Kim, J. Lee, L. Vandenberghe, and C. Yang. Techniques for improving the accuracy of geometric-programming based analog circuit design optimization. In *Proceedings of the IEEE International Conference on Computer Aided Design.*, pages 863–70, 2004.

[KNZ01] E. Knorr, R. Ng, and R. Zamar. Robust space transformations for distance-based operations. In *KDD '01: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 126–135. ACM Press, 2001.

[KPRC99] M. Krasnicki, R. Phelps, R. Rutenbar, and L. Carley. MAELSTROM: efficient simulation-based synthesis for custom analog cells. In *DAC '99: Proceedings of the 36th ACM/IEEE conference on Design automation*, pages 945–950, New York, NY, USA, 1999. ACM Press.

[KSD+97] J. Korte, A. Salas, H. Dunn, N. Alexandrov, W. Follett, G. Orient, and A. Hadid. Multidisciplinary approach to aerospike nozzle design. Technical Report 110326, NASA, 1997.

[KT93] L. Khachiyan and M. Todd. On the complexity of approximating the maximal inscribed ellipsoid for a polytope. *Math. Program.*, 61(2):137–159, 1993.

[Lev65] A. Levin. An algorithm of minimization of convex functions. *Soviet Math. Dokl.*, pages 1244–1247, 1965.

[LN95] D. Levin and E. Nadler. Convexity preserving interpolation by algebraic curves and surfaces. *Numerical Algorithms*, 9:113–139, 1995.

[LSRC02] H. Liu, A. Singhee, R. Rutenbar, and L. Carley. Remembrance of circuits past: macromodeling by data mining in large analog design spaces. In *DAC '02: Proceedings of the 39th conference on Design automation*, pages 437–442, New York, NY, USA, 2002. ACM Press.

[MG05] T. McConaghy and G. Gielen. Analysis of simulation-driven numerical performance modeling techniques for application to analog circuit optimization. In *ISCAS*, 2005.

[MGV98] O. Du Merle, J. Goffin, and J. Vial. On improvements to the analytic center cutting plane method. *Computational Optimization and Applications*, 11(1):37–52, 1998.

[MRT05] O. Mangasarian, J. Rosen, and M. Thompson. Global minimization via piecewise-linear underestimation. *Journal of Global Optimization*, 32, 2005.

[MS92] J. Mitchell and S. Suri. Separation and approximation of polyhedral objects. In *Proceedings of the ACM-SIAM symposium on Discrete algorithms*, pages 296–306. Society for Industrial and Applied Mathematics, 1992.

[Nes00] Y. Nesterov. Squared functional systems and optimization problems. In J. Frenk, C. Roos, T. Terlaky, and S. Zhang, editors, *High Performance Optimization Techniques*, pages 405–440. Kluwer, 2000.

[NN95] Y. Nesterov and A. Nemirovskii. *Interior-point polynomial algorithms in convex programming.* SIAM studies in applied mathematics, 1995.

[NY83] A. Nemirovsky and D. Yudin. *Problem Complexity and Method Efficiency in Optimization.* Wiley, 1983.

[PAOS03] J. Phillips, J. Afonso, A. Oliveira, and L. Miguel Silveira. Analog macromodeling using kernel methods. In *ICCAD '03: Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, page 446, Washington, DC, USA, 2003. IEEE Computer Society.

[Par00] P. Parrilo. *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*. PhD thesis, California Institute of Technology, 2000.

[PM00] J. Pittman and C. Murthy. Fitting optimal piecewise linear functions using genetic algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(7):701–718, 2000.

[PR88] T. Phillips and A. Rosenfeld. An isodata algorithm for straight line fitting. *Pattern Recognition*, 7(5):291–297, 1988.

[RB97] E. Rimon and S. Boyd. Obstacle collision detection using best ellipsoid fit. *Journal of Intelligent and Robotic Systems*, 18:105–126, 1997.

[RDPR85] G. Rives, M. Dhome, J. La Preste, and M. Richetin. Detection of patterns in images from piecewise linear contours. *Pattern Recognition Letters*, 3:99–104, 1985.

[RG94] J. Renaud and G. Gabriele. Approximation in nonhierarchic system optimization. *AIAA*, 32(1):198–205, 1994.

[Roy03] J. Roychowdhury. Automated macromodel generation for electronic systems. In *In Behavioral Modeling and Simulation Workshop*, 2003.

[Sch91] K. Schmudgen. The k-moment problem for compact semi-algebraic sets. *Math. Ann.*, 289:203–206, 1991.

[SF02] M. Storace and O. De Feo. Piecewise-linear approximation of nonlinear dynamical systems, September 2002.

[SGA03] G. Stehr, H. Graeb, and K. Antreich. Feasibility regions and their significance to the hierarchical optimization of analog and mixed-signal systems. *International Series of Numerical Mathematics*, 146:167–184, 2003.

[SH97] J. Sobiesky and R. Haftka. Multidisciplinary aerospace design optimization: Survey of recent developments. *Structural Optimization*, 14(1):1–23, 1997.

[TB04] F. Torrisi and A. Bemporad. Hysdel-a tool for generating computational hybrid models for analysis and synthesis problems. *IEEE Transactions on Control Systems Technology*, 12(2):235–249, March 2004.

[TCS+94] G. Taubin, F. Cukierman, S. Sullivan, J. Ponce, and D. Kriegman. Parameterized families of polynomials for bounded algebraic curve and surface fitting. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 16:287–303, 1994.

[TKE88] S. Tarasov, L. Khachiyan, and I. Erlich. The method of inscribed ellipsoids. *Soviet Math. Dokl.*, pages 226–230, 1988.

[Vai89] P. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *Proceedings of Symposium on Foundations of Computer Science*, pages 338–343, 1989.

[VB96] L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[VC92] V. Venkateswar and R. Chellappa. Extraction of straight lines in aerial images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(11):1111–1114, November 1992.

[VCBS04] A. Vincentelli, L. Carloni, F. De Bernardinis, and M. Sgroi. Benefits and challenges for platform-based design. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 409–414, New York, NY, USA, 2004. ACM Press.

[VdMV89] L. Vandenberghe, B. de Moor, and J. Vandewalle. The generalized linear complementarity problem applied to the complete analysis of resistive piecewise-linear circuits. *IEEE Transactions on Circuits and Systems*, 36(11):1382–1391, November 1989.

[WHCL93] D. Wang, N. Huang, H. Chao, and R. Lee. Plane sweep algorithms for the polygonal approximation problems with applications. In *Proceedings of the International Symposium on Algorithms and Computation*, pages 515–522, London, UK, 1993. Springer-Verlag.

[WRBB96] B. Wujek, J. Renaud, S. Batill, and J. Brockman. Concurrent subspace optimization using design variable sharing in a distributed computing environment. *Concurrent Engineering: Research and Applications*, 4(4):361–378, 1996.

[WV03] G. Wolfe and R. Vemuri. Extraction and use of neural network models in automated synthesis of operational amplifiers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(2), 2003.

[Ye97] Y. Ye. *Interior Point Algorithms: Theory and Analysis*. Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, New York, 1997.

[Yin98] P. Yin. Algorithms for straight line fitting using k-means. *Pattern Recognition Letters*, 19(1):31–41, January 1998.