DISCIPLINED CONVEX PROGRAMMING

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

Michael Charles Grant
December 2004

Preface

Convex programming unifies and generalizes least squares (LS), linear programming (LP), and quadratic programming (QP). It has received considerable attention recently for a number of reasons: its attractive theoretical properties; the development of efficient, reliable numerical algorithms; and the discovery of a wide variety of applications in both scientific and non-scientific fields. Courses devoted entirely to convex programming are available at Stanford and elsewhere.

For these reasons, convex programming has the potential to become a ubiquitous numerical technology alongside LS, LP, and QP. Nevertheless, there remains a significant impediment to its more widespread adoption: the high level of expertise in both convex analysis and numerical algorithms required to use it. For potential users whose focus is the application, this prerequisite poses a formidable barrier, especially if it is not yet certain that the outcome will be better than with other methods. In this dissertation, we introduce a modeling methodology called *disciplined convex programming* whose purpose is to lower this barrier.

As its name suggests, disciplined convex programming imposes a set of conventions to follow when constructing problems. Compliant problems are called, appropriately, *disciplined convex programs*, or DCPs. The conventions are simple and teachable, taken from basic principles of convex analysis, and inspired by the practices of experts who regularly study and apply convex optimization today. The conventions do *not* limit generality; but they *do* allow the steps required to solve DCPs to be automated and enhanced. For instance, determining if an arbitrary mathematical program is convex is an intractable task, but determining if that same problem is a DCP is straightforward. A number of common numerical methods for optimization can be adapted to solve DCPs. The conversion of DCPs to solvable form can be fully automated, and the natural problem structure in DCPs can be exploited to improve performance.

Disciplined convex programming also provides a framework for collaboration between users with different levels of expertise. In short, disciplined convex programming allows applications-oriented users to focus on modeling, and—as they would with LS, LP, and QP—to leave the underlying mathematical details to experts.

Acknowledgments

To the members my reading committee, Stephen Boyd, Yinyu Ye, and Michael Saunders: thank you for your guidance, criticism, and endorsement. I am truly fortunate to have studied under men of such tremendous (and well-deserved) reputation. If my bibliography were trimmed to include only the works you have authored, it would remain an undeniably impressive list.

To my adviser Stephen Boyd: it has been a true privilege to be under your tutelage. Your knowledge is boundless, your compassion for your students is heartfelt, and your perspective on engineering, mathematics, and academia is unique and attractive. I would not have returned to Stanford to finish this work for any other professor. I look forward to our continued collaboration.

To the Durand 110 crowd and all of my other ISL colleagues, including David Banjerdpongchai, Ragu Balakrishnan, Young-Man Cho, Laurent El-Ghaoui, Maryam Fazel, Eric Feron, Marc Goldburg, Arash Hassibi, Haitham Hindi, Herve Lebret, Miguel Lobo, Costis Maglaras, Denise Murphy, Greg Raleigh, Lieven Vandenberghe, and Clive Wu: thank you for your collaboration, encouragement, and friendship.

To Alexandre d'Aspremont, Jon Dattoro, Arpita Ghosh, Kan-Lin Hsiung, Siddharth Joshi, Seung Jean Kim, Kwangmoo Koh, Alessandro Magnani, Almir Mutapcic, Dan O'Neill, Sikandar Samar, Jun Sun, Lin Xiao, and Sunghee Yun: you have made this "old veteran" feel welcome in Professor Boyd's group again, and I wish you true success in your research.

To Doug Chrissan, Paul Dankoski, and V.K. Jones: thank you for your faithful friendship. Just so you know, it is a holiday at my house every day. Forgive me, though, because I still prefer Jing Jing over Su Hong to Go.

To my parents, my brother, and my entire family: this would simply never have happened without your support, encouragement, and love. Thank you in particular, Mom and Dad, for your tireless efforts to provide us with a first-class education—and not just in school.

Last but not least, to my darling wife Callie: how can I adequately express my love and gratitude to you? Thank you for your unwavering support during this endeavor, particularly in these last stages, and especially through the perils of home remodeling and the joys of parenthood. May the time spent completing this work represent but a paragraph in the book of our long lives together.

Contents

Preface **v**

Acknowledgments **vi**

1 Introduction **1**

1.1 Motivation 2

1.1.1 The Euclidean norm 3

1.1.2 The Manhattan norm 3

1.1.3 The Chebyshev norm 3

1.1.4 The Hölder norm 4

1.1.5 An uncommon choice 6

1.1.6 The expertise barrier 6

1.1.7 Lowering the barrier 7

1.2 Convex programming 8

1.2.1 Theoretical properties 9

1.2.2 Numerical algorithms 10

1.2.3 Applications 13

1.2.4 Convexity and differentiability 14

1.3 Convexity verification 15

1.3.1 Smooth NLP analysis 15

1.3.2 Parsing methods 17

1.3.3 Empirical analysis 18

1.4 Modeling frameworks 18

1.5 Synopsis 19

1.6 Conventions 20

2 Motivation **21**

2.1 Introduction 21

2.2 Analysis 22

2.2.1 The dual problem 22

2.2.2 Definitions . . . 23
2.2.3 Feasibility . . . 24
2.2.4 Optimality . . . 25
2.2.5 Sensitivity . . . 25
2.2.6 Summary . . . 26
2.3 The solver . . . 27
2.4 The smooth case . . . 29
2.5 The Euclidean norm . . . 31
2.6 The Manhattan norm . . . 32
2.7 The Chebyshev norm . . . 34
2.8 The Hölder l_p norm . . . 36
2.9 The largest-L norm . . . 39
2.10 Conclusion . . . 43

3 Disciplined convex programming **44**
3.1 The ruleset . . . 45
3.1.1 Top-level rules . . . 45
3.1.2 The product-free rules . . . 47
3.1.3 The sign rules . . . 50
3.1.4 The composition rules . . . 52
3.2 The atom library . . . 56
3.2.1 The prototype . . . 57
3.2.2 Attributes . . . 58
3.3 Verification . . . 61
3.4 Navigating the ruleset . . . 64
3.5 Implementing atoms . . . 67
3.5.1 Simple implementations . . . 68
3.5.2 Graph implementations . . . 69
3.6 Conclusion . . . 72

4 Solving disciplined convex programs **73**
4.1 The canonical form . . . 74
4.1.1 Our choice . . . 74
4.1.2 The dual problem . . . 76
4.2 Canonicalization and dual recovery . . . 77
4.2.1 Linearization . . . 78
4.2.2 Set decomposition . . . 82
4.2.3 Graph expansion . . . 84
4.2.4 Dual recovery . . . 86
4.3 Numerical solution . . . 88

4.3.1 Barrier methods . . . 89
4.3.2 Primal-dual methods . . . 93
4.3.3 Cutting-plane methods . . . 95
4.4 Conclusions . . . 99

5 Conclusion **100**
5.1 Motivation, revisited . . . 101
5.2 Future work . . . 103
5.2.1 Improving the `cvx` framework . . . 103
5.2.2 Extending disciplined convex programming . . . 105
5.2.3 Advanced applications . . . 110
5.3 A brief defense—or, there's nothing like leather . . . 111

A `cvx`: An example **112**
A.1 Initial parsing . . . 113
A.2 Ruleset verification . . . 115
A.3 Canonicalization . . . 123
A.4 Numerical instance generation . . . 127
A.5 Numerical solution . . . 128

Bibliography **132**

List of Figures

1.1 `cvx` specifications for the Chebyshev, Hölder, and largest-L cases. 7

1.2 An example CP in the `cvx` modeling language. 19

2.1 C++ code to compute the value, gradient, and Hessian of $f(x) = \|Ax - b\|_2$. 30

3.1 Valid constraints. 46

3.2 Three different "versions" of $f(y) = y^2$. 53

3.3 The `cvx` specification for (3.60). 61

3.4 An expression tree for (3.60), annotated with applicable convexity rules. 63

3.5 An entropy maximization problem. 64

3.6 A geometric program in convex form. 64

3.7 An entropy maximization problem, expressed as a DCP. 65

3.8 A geometric program in convex form, expressed as a DCP. 66

3.9 Simple implementations. 69

3.10 Graph implementations. 70

4.1 The DCP workflow. 73

5.1 A norm minimization test problem for the `cvx` modeling framework. 102

5.2 The entry in the file `norms.cvx` for the largest-L norm. 102

5.3 An example implementation of the logarithm function with support for quasiconvexity and quasilinearity. 107

Chapter 1

Introduction

Convex programming is a subclass of nonlinear programming (NLP) that unifies and generalizes least squares (LS), linear programming (LP), and convex quadratic programming (QP). This generalization is achieved while maintaining many of the important, attractive theoretical properties of these predecessors. Numerical algorithms for solving convex programs are maturing rapidly, providing reliability, accuracy, and efficiency. A large number of applications have been discovered for convex programming in a wide variety of scientific and non-scientific fields, and it seems clear that even more remain to be discovered. For these reasons, convex programming arguably has the potential to become a ubiquitous technology alongside LS, LP, and QP. Indeed, efforts are underway to develop and teach it as a distinct discipline [29, 18, 122].

Nevertheless, there is a significant impediment to the more widespread adoption of convex programming: the high level of expertise required to use it. With mature technologies such as LS, LP, and QP, problems can be specified and solved with relatively little effort, with at most a very basic understanding of the computations involved. This is not the case with general convex programming. That a user must understand the basics of convex analysis is both reasonable and unavoidable; but in fact, a much deeper understanding is required. Furthermore, a user must find a way to transform his problem into one of the many limited standard forms; or, failing that, develop a custom solver. For *applications-oriented* users—users whose focus is on a particular application, and not on the underlying mathematics—these requirements constitute a formidable barrier to the use of convex programming, especially if it is not yet certain that the outcome will be any better than with other methods. The purpose of the work presented here is to lower this barrier.

In this dissertation, we introduce a new modeling methodology called *disciplined convex programming*. As the term "disciplined" suggests, the methodology imposes a set of conventions that one must follow when constructing convex programs. The conventions are simple and teachable, taken from basic principles of convex analysis, and inspired by the practices of those who regularly study and apply convex optimization today. Conforming problems are called, appropriately, *disciplined convex programs*, or *DCPs*. These conventions, we hasten to say, do not limit generality; but they *do* allow much of the manipulation and transformation required to analyze and solve convex programs to be automated. For example, the task of determining if an arbitrary NLP is convex is both theoretically and practically intractable; the task of determining if it is a *disciplined* convex program is straightforward. In addition, the transformations necessary to convert disciplined convex programs into solvable form, as well as the reverse transformations required to recover valuable dual information, can be fully automated.

A novel aspect of this work is a new way to define a function or set in a modeling framework: as the solution of a convex program. We call such a definition a *graph implementation* because it exploits the properties of epigraphs and hypographs of convex and concave functions, respectively. The benefits of graph implementations are numerous; most significant is their ability to support nondifferentiable functions without the loss of reliability or performance traditionally associated with them. Because a wide variety of useful convex programs involve nondifferentiable functions, this support is of considerable practical benefit.

We have created a modeling framework called `cvx` that implements in software the key principles of the disciplined convex programming methodology, in order to prove their feasibility and usefulness. The system is built around a specification language that allows DCPs to be specified in a natural mathematical form. The components of the system address key tasks such as verification, conversion to standard form, and construction of an appropriate numerical solver. It is our genuine hope that `cvx` will prove useful not only for applications-oriented users wishing to create and solve models, but also as a platform for advanced research in convex programming algorithms and applications. In the final chapter of this dissertation we suggest areas of advanced study where the `cvx` framework could be used effectively.

The remainder of this chapter begins with some motivation for this work, by examining how current numerical methods can be used to solve a simple, common convex program. Next, we present brief introductions to convex programming and modeling frameworks. The chapter concludes with a synopsis of the dissertation and a brief notational comment.

1.1 Motivation

To illustrate the complexities of practical convex programming, let us examine how to solve a basic and yet common problem: the unconstrained norm minimization

$$\text{minimize} \quad f(x) \triangleq \|Ax-b\| \qquad A \in \mathbb{R}^{m\times n} \quad b \in \mathbb{R}^m. \tag{1.1}$$

We will examine several choices of the norm $\|\cdot\|$. For simplicity, we assume that $m \geq n$, that A has full rank, and that A and b are partitioned by rows as follows:

$$A^T \triangleq [a_1 \quad a_2 \quad \ldots \quad a_m] \qquad b^T \triangleq [b_1 \quad b_2 \quad \ldots \quad b_m]. \tag{1.2}$$

1.1.1 The Euclidean norm

The most common choice of the norm is certainly the ℓ_2 or Euclidean norm,

$$f(x) \triangleq \|Ax-b\|_2 = \sqrt{\textstyle\sum_{i=1}^m (a_i^T x - b_i)^2}. \tag{1.3}$$

In this case, (1.1) is easily recognized as a least squares problem, which as its name implies is often presented in an equivalent quadratic form,

$$\text{minimize} \quad f(x)^2 = \|Ax-b\|_2^2 = \textstyle\sum_{i=1}^m (a_i^T x - b_i)^2. \tag{1.4}$$

The solution x satisfies the linear equation $x = (A^TA)^{-1}A^Tb$, which can be solved using a Cholesky factorization of A^TA, or more accurately using a QR or SVD factorization of A [74]. A number of software packages to solve least squares problems are readily available; *e.g.*, [6, 111].

1.1.2 The Manhattan norm

For the ℓ_1 or Manhattan norm

$$f(x) \triangleq \|Ax-b\|_1 = \sum_{i=1}^m |a_i^T x - b_i|, \tag{1.5}$$

there is no analytical solution to (1.1); but a solution can be obtained from the LP

$$\begin{array}{ll} \text{minimize} & \vec{1}^T v \\ \text{subject to} & -v \le Ax - b \le v \end{array} \qquad (v \in \mathbb{R}^m). \tag{1.6}$$

It is not difficult to solve (1.6) in practice because efficient and reliable LP solvers are widely available. In fact, one is included with virtually every piece of spreadsheet software sold [63].

1.1.3 The Chebyshev norm

Similarly, for the ℓ_∞ or Chebyshev norm,

$$f(x) \triangleq \|Ax-b\|_\infty = \max_{i=1,2,\dots,m} |a_i^T x - b_i|, \tag{1.7}$$

a solution can also be determined from an appropriate LP:

$$\begin{array}{ll} \text{minimize} & q \\ \text{subject to} & -q\vec{1} \le Ax - b \le q\vec{1} \end{array} \qquad (q \in \mathbb{R}). \tag{1.8}$$

Again, this problem can be easily solved using readily available LP software.

1.1.4 The Hölder norm

Now consider the Hölder or ℓ_p norm

$$f(x) \triangleq \|Ax - b\|_p = \left(\sum_{i=1}^{m} |a_i^T x - b_i|^p \right)^{1/p}. \tag{1.9}$$

Newton's method

When $p \geq 2$, $f(x)$ is twice differentiable everywhere except when $Ax = b$. This condition is not likely to be encountered in practical norm minimization problems, so we may consider applying Newton's method directly. In fact, let us minimize instead the related function

$$g(x) \triangleq f(x)^p = \sum_{i=1}^{m} |a_i^T x - b_i|^p, \tag{1.10}$$

which produces the same solution x but simplifies the calculations somewhat, and eliminates the nondifferentiability at $Ax = b$. The iterates produced by Newton's method are

$$\begin{aligned} x^{(k+1)} &= x^{(k)} - \alpha_k \left(\nabla^2 g(x^{(k)}) \right)^{-1} \nabla g(x^{(k)}) \\ &= x^{(k)} - \frac{\alpha_k}{p-1} (A^T W^{(k)} A)^{-1} A^T W^{(k)} (Ax^{(k)} - b) \\ &= \left(1 - \frac{\alpha_k}{p-1} \right) x^{(k)} + \frac{\alpha_k}{p-1} \arg\min_{\bar{w}} \left\| (W^{(k)})^{1/2} (A\bar{w} - b) \right\|_2 \end{aligned} \tag{1.11}$$

where $x_0 = 0$, $k = 1, 2, 3, \ldots$, and we have defined

$$W^{(k)} \triangleq \mathbf{diag}(|a_1^T x^{(k)} - b_1|^{p-2}, |a_2^T x^{(k)} - b_2|^{p-2}, \ldots, |a_m^T x^{(k)} - b_m|^{p-2}). \tag{1.12}$$

The quantities $\alpha_k \in (0, 1]$ are either fixed or determined at each iteration using a line search technique. Notice how the Newton computation involves a (weighted) least squares problem: in fact, if $p = 2$, then $W_k \equiv I$, and a single iteration with $\alpha_1 = 1$ produces the correct solution. So the more "complex" ℓ_p case simply involves solving a series of similar least squares problems—a resemblance that turns up quite often in numerical methods for convex programming.

An important technical detail must be mentioned here. If one or more of the residuals $a_i^T x^{(i)} - b_i$ are zero, the matrix $W^{(k)}$ (1.12) is singular, and $\nabla^2 g(x) = A^T W^{(k)} A$ can therefore be singular as well. If $m \gg n$ and A has full column rank, this is not likely—but care must be taken nonetheless to guard for the possibility. A variety of methods can be considered to address the issue—for example, adding a slight damping factor ϵI to either $W^{(k)}$ or $\nabla^2 g(x)$.

Newton's method is itself a relatively straightforward algorithm, and a number of implementations have been developed; *e.g.*, [117]. These methods require that code be created to perform the computation of the gradient and Hessian of the function being minimized. This task is eased somewhat by automatic differentiation packages such ADIC [86] or ADIFOR [20], which can generate derivative code from code that simply computes a function's value.

A better approach

For $1 < p < 2$, Newton's method cannot reliably be employed, because neither $f(x)$ nor $g(x) \triangleq f(x)^p$ is twice differentiable whenever *any* of the residuals are zero. An alternative that works for all $p \in [1, +\infty)$ is to apply a barrier method to the problem. A full introduction to barrier methods is beyond the scope of this text, so we will highlight only key details. The reader is invited to consult [124] for a truly exhaustive development of barrier methods, or [29] for a gentler introduction.

To begin, we note that the solution to (1.1) can be obtained by solving

$$\begin{array}{ll} \text{minimize} & \vec{1}^T v \\ \text{subject to} & |a_i^T x - b_i|^p \leq v_i \quad i = 1, 2, \ldots, m. \end{array} \tag{1.13}$$

To solve (1.13), we construct a *barrier function* $\phi : \mathbb{R}^n \times \mathbb{R}^m \to (\mathbb{R} \cup +\infty)$ to represent the inequality constraints [124]:

$$\phi(x, v) \triangleq \begin{cases} \sum_{i=1}^m - \log(v_i^{2/p} - (a_i^T x - b_i)^2) - 2 \log v_i & (x, v) \in \operatorname{Int} S \\ +\infty & (x, v) \notin \operatorname{Int} S \end{cases} \tag{1.14}$$
$$S \triangleq \left\{ (x, v) \in \mathbb{R}^n \times \mathbb{R}^m \;\middle|\; |a_i^T x - b_i|^p \leq v_i, \; i = 1, 2, \ldots, m \right\}.$$

The barrier function is finite and twice differentiable whenever the inequality constraints in (1.13) are strictly satisfied, and $+\infty$ otherwise. This barrier is used to create a family of functions g_μ parameterized over a quantity $\mu > 0$:

$$g_\mu : \mathbb{R}^n \times \mathbb{R}^m \to (\mathbb{R} \cup +\infty), \quad g_\mu(x, v) \triangleq \vec{1}^T v + \mu \phi(x, v). \tag{1.15}$$

It can be shown that the minimizing values of $g_\mu(x, v)$ converge to the solution of the original problem as $\mu \to 0$. A practical barrier method takes Newton steps to minimize $g_\mu(x, v)$, decreasing the value of μ between iterations in a manner chosen to ensure convergence and acceptable performance.

This approach is obviously significantly more challenging than our previous efforts. As with the Newton method for the $p \geq 2$ case, code must be written (or automatic differentiation employed) to compute the gradient and Hessian of $g_\mu(x, v)$. Furthermore, the authors are unaware of any readily available software implementing a general-purpose barrier method such as this. Several custom solvers, at least one employing a barrier method, have been designed specifically for the p-norm minimization problem (*e.g.*, [171]), but applications-oriented users are not likely to know of them.

In §2.8, we provide yet another alternative, in which the problem is converted to a smooth nonlinear program. The conversion itself is not obvious, nor is it foolproof. Thus the point remains that the ℓ_p case is the most difficult option to solve thus far.

1.1.5 An uncommon choice

Given a vector $w \in \mathbb{R}^m$, let $w_{[|k|]}$ be the k-th element of the vector after it has been sorted from largest to smallest in absolute value:

$$|w_{[|1|]}| \geq |w_{[|2|]}| \geq \cdots \geq |w_{[|m|]}|. \tag{1.16}$$

Then let us define the *largest-L* norm as follows:

$$\|w\|_{[|L|]} \triangleq \sum_{k=1}^{L} |w_{[|k|]}| \quad (L \in \{1, 2, \ldots, m\}). \tag{1.17}$$

Solving (1.1) using this norm produces a vector x that minimizes the sum of the absolute values of the L largest residuals. It is equivalent to the ℓ_∞ case for $L = 1$ and the ℓ_1 case for $L = m$, but for $1 < L < m$ this norm produces novel results.

It may not be obvious that (1.17) is a norm or even a convex function, but it is indeed both. It is probably even less obvious how this problem can be solved. But in fact, a solution can be obtained from the following LP:

$$\begin{array}{ll} \text{minimize} & \vec{1}^T v + Lq \\ \text{subject to} & -v - q\vec{1} \leq Ax - b \leq v + q\vec{1} \qquad (v \in \mathbb{R}^m,\ q \in \mathbb{R}). \\ & v \geq 0 \end{array} \tag{1.18}$$

This LP is only slightly larger than the ones used for the ℓ_1 and ℓ_∞ cases. The result is known—see, for example [129]—but not widely so, even among those who actively study optimization. We provide a more detailed development of the largest-L norm in §2.9.

1.1.6 The expertise barrier

The conceptual similarity of these problems is obvious, but the methods employed to solve them differ significantly. Several numerical algorithms are represented: least squares, linear programming, Newton's method, and a barrier method. In most cases, transformations are required to produce an equivalent problem suitable for numerical solution. These transformations are not likely to be obvious to an applications-oriented user whose primary expertise is not optimization. As a result, those wishing to solve a norm minimization problem may, out of ignorance or practicality, restrict their view to norms for which solution methods are widely known, such as ℓ_2 or ℓ_1—even if doing so compromises the accuracy of their models. This might be understandable for cases like, say, $\ell_{1.5}$,

```
minimize     norm_inf( A x - b );
parameters   A[m,n], b[n];
variable     x[n];
include      "norms.cvx";

minimize     norm_p( A x - b, p );
parameters   A[m,n], b[n], p >= 1;
variable     x[n];
include     "norms.cvx";

minimize    norm_largest( A x - b, L );
parameters  A[m,n], b[n], L in #{1,2,...,n};
variable    x[n];
include     "norms.cvx";
```

Figure 1.1: `cvx` specifications for the Chebyshev, Hölder, and largest-L cases.

where the computational method employed is quite complex. But the true computational complexity may be far less severe than it seems on the surface, as is the case with the largest-L norm.

Even this simple example illustrates the high level of expertise needed to solve even basic convex optimization problems. Of course, the situation worsens if more complex problems are considered. In Chapter 2 we introduce constraints on x that eliminate analytical solutions for the ℓ_2 case and prevent the use of a simple Newton's method for the ℓ_p case.

Stated positively: if the process of specifying and solving these problems could be simplified, modelers would be free to create more accurate and more complex models.

1.1.7 Lowering the barrier

In Figure 1.1, `cvx` specifications for three of the problems presented here are given. In each case, the problem is in its original form; no transformations described above have been applied in advance to convert them into solvable form. Instead, the models utilize the functions `norm_inf`, `norm_p`, and `norm_largest` for the ℓ_∞, ℓ_p, and largest-L norms, respectively. The definitions of these functions have been stored in a separate file `norms.cvx` and are referred to by an `include` command. (For illustration, the models in Figure 1.1 take advantage of features in the `cvx` language that have been planned but not yet implemented. See §5.1 for versions of these problems implemented using the current version of `cvx`.)

In most modeling frameworks, function definitions consist of code to compute their values and derivatives. This method is not useful for the norms used here, because they are not differentiable. *Graph implementations*, which we describe in §3.5, provide a necessary alternative. For now, it is enough to know that they effectively describe the very transformations illustrated in §1.1.1-§1.1.5 above. For example, the definitions for the `norm_inf` and `norm_largest` provide the information necessary to convert their respective problems into LPs. The definition for `norm_p` includes a barrier function for its epigraph, which can be used to construct a barrier method for that problem.

So neither disciplined convex programming nor the `cvx` framework actually eliminates the transformations needed to solve these convex programs. But they do allow the transformations to be *encapsulated* within function definitions: that is, hidden from the user and performed without that user's intervention. The definitions are easily shared among users, suggesting a natural, collaborative environment where those with advanced expertise in convex programming can share their knowledge with less experienced modelers in a practical way. Therefore, the task of *solving* convex programs can be appropriately returned to experts, freeing applications-oriented users to focus confidently on creating and solving models.

1.2 Convex programming

A *mathematical program* is an optimization problem of the form

$$\begin{array}{rlll} \mathcal{P}_{\text{MP}}\text{: minimize} & f(x) & & \\ \text{subject to} & g_i(x) \leq 0 & i = 1, 2, \dots, n_g & \\ & h_j(x) = 0 & j = 1, 2, \dots, n_h & \\ & x \in X & & \end{array} \tag{1.19}$$

or one that can be readily converted into this form. The vector x is the *problem variable* drawn from a set $X \subseteq \mathbb{R}^n$. We will assume that $X \equiv \mathbb{R}^n$ for all of the problems considered here. The function $f : X \to \mathbb{R}$ is the *objective function*, and the relations $g_i(x) \leq 0$ and $h_j(x) = 0$ are the *inequality* and *equality constraints*, respectively. [76, 29]. In some contexts, including ours, the functions are permitted to be *extended-valued*; that is, obtaining the values $\pm\infty$.

Solving $\mathcal{P}_{\text{MP}}$ (1.19) involves making determinations such as the following:

- *Feasibility*: Determine if $\mathcal{P}_{\text{MP}}$ is *feasible*; that is, if there exists at least one *feasible point* $x \in X$ that satisfies the constraints. Otherwise, we say that the problem is *infeasible.*

- *Optimality*: Find the *optimal value* f^* of $\mathcal{P}_{\text{MP}}$—the infimum of $f(x)$ taken over all feasible points x—and, if possible, an *optimal point* x^* that achieves it; *i.e.*, $f(x^*) = f^*$. By convention, $f^* = +\infty$ if $\mathcal{P}_{\text{MP}}$ is infeasible, and $f^* = -\infty$ if $\mathcal{P}_{\text{MP}}$ is *unbounded.* An optimal point need not exist, even if f^* is finite—and if one does exist, it need not be unique.

- *Sensitivity*: Determine how the optimal value responds to perturbations in the constraints.

Typically these determinations are not expected to be made exactly, but rather to within predefined numerical tolerances. For example, instead of finding an optimal point x^*, we may find a *near-optimal* point $\bar{x}$ that is provably close to optimal; *e.g.*, satisfying $f(\bar{x}) - f^* \leq \epsilon$.

Certain special cases of (1.19) receive considerable dedicated attention. By far the most common is the linear program (LP), for which the functions f, g_i, h_j are all affine; *e.g.*,

$$\begin{array}{rlcll} \text{minimize} & c^T x & & f(x) \triangleq c^T x & \\ \text{subject to} & Ax = b & \Longrightarrow & h_j(x) \triangleq a_j^T x - b_j & j = 1, 2, \dots, m \\ & x \geq 0 & & g_i(x) \triangleq -x_i & i = 1, 2, \dots, n. \end{array} \tag{1.20}$$

Quadratic programs (QPs) and least-squares problems can be expressed in this form as well. The class of *nonlinear programs* (NLPs) includes nearly all problems that can be expressed in the form (1.19) with $X \equiv \mathbb{R}^n$, although generally there are assumptions made about the continuity and/or differentiability of the objective and constraint functions.

A *convex program* (CP) is yet another special case of (1.19): one in which the objective function f and inequality constraint functions g_i are convex, and the equality constraint functions h_j are affine. A function $f : \mathbb{R}^n \to (\mathbb{R} \cup +\infty)$ is convex if it satisfies

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha) f(y) \quad \forall\, x, y \in \mathbb{R}^n,\ \alpha \in (0,1). \tag{1.21}$$

The set of CPs is a strict subset of the set of NLPs, and includes all LPs, convex QPs (*i.e.*, QPs for which the Hessian $\nabla^2 f$ is positive semidefinite), and least-squares problems. Several other classes of CPs have been identified recently as standard forms. These include *semidefinite programs* (SDPs) [159], *second-order cone programs* (SOCPs) [103], and *geometric programs* (GPs) [49, 7, 144, 52, 95]. The work we present here applies to all of these special cases as well as to the general class of CPs.

The practice of modeling, analyzing, and solving CPs is known as *convex programming*. In this section we provide a survey of convex programming, including its theoretical properties, numerical algorithms, and applications.

1.2.1 Theoretical properties

A comprehensive theory of convex analysis was developed by the 1970s [142, 146], and advances have continued since [84, 85, 18, 29]. From this work we know that several powerful and practical theoretical conclusions can be drawn once it is established that a mathematical program is convex. The most fundamental of these is that for CPs, local optima are guaranteed to be global. Put another way, if local optimality can somehow be demonstrated (for example, using KKT conditions), then global optimality is assured. Except in certain special cases, no similar guarantees can be made for nonconvex NLPs. Such problems might exhibit multiple local optima, so an exhaustive search would be required to prove global optimality—an intractable task.

Convex programming also has a rich duality theory that is very similar to the duality theory that accompanies linear programming, though it is a bit more complex. The dual of a CP is itself a CP, and its solution often provides interesting and useful information about the original, or primal, problem. For example, if the dual problem is unbounded, then the primal must be infeasible. Under certain conditions, the reverse implication is also true: if the primal is infeasible, then its dual must be unbounded. These and other consequences of duality facilitate the construction of numerical algorithms with definitive stopping criteria for detecting infeasibility, unboundedness, and near-optimality. In addition, the dual problem provides valuable information about the sensitivity of the primal problem to perturbations in its constraints. For a more complete development of convex duality, see [142, 107].

Another important property of CPs is the provable existence of efficient algorithms for solving them. Nesterov and Nemirovsky proved that a polynomial-time barrier method can be constructed for any CP that meets certain technical conditions [123]. Other authors have shown that problems that do not meet those conditions can be embedded into larger problems that do—effectively making barrier methods universal [176, 107, 178].

Note that the theoretical properties discussed here, including the existence of efficient solution methods, hold even if a CP is nondifferentiable—that is, if one or more of the constraint or objective functions is nondifferentiable. The practical ramifications of this fact are discussed in §1.2.4.

1.2.2 Numerical algorithms

The existence of efficient algorithms for solving CPs has been known since the 1970s, but it is only through advances in the last two decades that this promise has been realized in practice. Much of the modern work in numerical algorithms has focused on *interior-point methods* [174, 143, 168]. Initially such work was limited to LPs [90, 139, 75, 94, 112, 116, 60], but was soon extended to encompass other CPs as well [123, 124, 5, 89, 125, 167, 8, 120]. Now a number of excellent solvers are readily available, both commercial and freely distributed.

Below we provide a brief survey of solvers for convex programming. We have separated them into two classes: those that rely on *standard forms*, and those that rely on *custom code*.

Standard forms

Most CP solvers are designed to handle certain prototypical problems known as *standard forms*. In other words, such solvers handle a limited family of problems with a very specific structure, or obeying certain conventions. The least squares problem and the LP are two common examples; we list several others below. These solvers trade generality for ease of use and performance.

It is instructive to think of the collection of standard form solvers as a "toolchest" of sorts. This toolchest is reasonably complete, in that most CPs that one might encounter can be transformed into one (or more) of these standard forms. However, the required transformations are often far from obvious, particularly for an applications-oriented user.

Smooth convex programs Several solvers have been designed to solve CPs in standard NLP form (1.19), under the added condition that the objective function f and inequality constraint functions g_i are *smooth*—that is, twice continuously differentiable—at least over the region that the algorithm wishes to search. We call such problems *smooth* CPs; conversely, we call CPs that do not meet this criteria *nonsmooth* CPs.

Software packages that solve smooth CPs include LOQO [162], which employs a primal/dual interior-point method, and the commercial package MOSEK [118], which implements the homogeneous algorithm. These solvers generally perform quite well in practice. Many systems designed for smooth *nonconvex* NLPs will often solve smooth CPs efficiently as well [36, 68, 119, 69, 44, 35, 163, 127, 31, 37, 17, 70, 58, 153]. This is not surprising when one considers that these algorithms typically exploit *local* convexity when computing search directions.

One practical difficulty in the use of smooth CP or NLP solvers is that the solver must be able to calculate the gradient and Hessian of the objective and constraint functions at points of its choosing. In some cases, this may require the writing of custom code to perform these computations. Many modeling frameworks (see §1.4) simplify this process greatly by allowing functions to be expressed in natural mathematical form and computing derivatives automatically (*e.g.*, [59, 30]).

Conic programs An entire family of standard forms that have become quite common are the primal and dual *conic forms*

$$\begin{array}{rl} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \in \mathcal{K} \end{array} \qquad \text{or} \qquad \begin{array}{rl} \text{maximize} & b^T y \\ \text{subject to} & A^T y + z = c \\ & z \in \mathcal{K}^* \end{array} \tag{1.22}$$

in which the sets $\mathcal{K}$ and $\mathcal{K}^*$ are closed, convex *cones* (*i.e.*, they satisfy $\alpha\mathcal{K} \equiv \mathcal{K}$ and $\alpha\mathcal{K}^* \equiv \mathcal{K}^*$ for all $\alpha > 0$). The most common conic form is the LP, for which $\mathcal{K} = \mathcal{K}^*$ is the nonnegative orthant

$$\mathcal{K} = \mathcal{K}^* = \mathbb{R}^n_+ \triangleq \{\, x \in \mathbb{R}^n \;\mid\; x_j \geq 0,\; j = 1, \ldots, n \,\}. \tag{1.23}$$

It can be shown that virtually any CP can be represented in conic form, with appropriate choice of $\mathcal{K}$ or $\mathcal{K}^*$ [124]. In practice, two conic forms (besides LP) dominate all recent study and implementation. One is the *semidefinite program* (SDP), for which $\mathcal{K} = \mathcal{K}^*$ is an isomorphism of the cone of positive semidefinite matrices

$$\mathcal{S}^n_+ \triangleq \left\{\, X = X^T \in \mathbb{R}^{n \times n} \;\mid\; \lambda_{\min}(X) \geq 0 \,\right\}. \tag{1.24}$$

The second is the *second-order cone program* (SOCP), for which $\mathcal{K} = \mathcal{K}^*$ is the Cartesian product of one or more second-order or Lorentz cones,

$$\mathcal{K} = \mathcal{Q}^{n_1} \times \cdots \times \mathcal{Q}^{n_K}, \quad \mathcal{Q}^n \triangleq \{\, (x, y) \in \mathbb{R}^n \times \mathbb{R} \;\mid\; \|x\|_2 \leq y \,\}. \tag{1.25}$$

SDP and SOCP receive this focused attention because many applications have been discovered for them, and because their geometry admits certain useful algorithmic optimizations [126, 78, 155, 54, 73, 107, 134]. Publicly available solvers for SDP and SOCP include SeDuMi [148], CDSP [22], SDPA [65], SDPT3 [158], and DSDP [16]. These solvers are generally quite efficient, reliable, and are entirely data-driven: that is, they require no external code to perform function calculations.

Geometric programs Another standard form that has been studied for some time and has generated renewed interest recently is the *geometric program* (GP). The GP is actually a bit of a unique case, in that it is in fact not convex—but a simple transformation produces an equivalent problem that is convex. In convex form, the objective and inequality constraint functions obtain a so-called "log-sum-exp" structure; for example,

$$f(x) \triangleq \log \sum_{i=1}^{M} e^{a_i^T x + b_i} \qquad a_i \in \mathbb{R}^n,\ b_i \in \mathbb{R},\ i = 1, 2, \dots, M. \tag{1.26}$$

GPs have been used in various fields since the late 1960s [49]. In convex form they are smooth CPs, and recent advances in algorithms have greatly improved the efficiency of their solution [95].

Custom code

There are instances where a CP cannot be transformed into one of the standard forms above—or perhaps the transformations cannot be determined. An alternative is to use one of the methods that we list here, which are *universal* in the sense that they can, in theory, be applied to *any* CP. The cost of this universality is that the user must determine certain mathematical constructions, and write custom code to implement them.

Barrier methods A barrier method replaces the inequality constraint set

$$\mathcal{S} \triangleq \{\, x \in \mathbb{R}^n \mid g_i(x) \le 0,\ i = 1, \dots, n_g \,\} \tag{1.27}$$

with a twice differentiable convex barrier function $\phi : \mathbb{R}^n \to \mathbb{R}$ satisfying $\mathbf{dom}\,\phi = \operatorname{Int}\mathcal{S}$, producing a modified problem

$$\begin{array}{ll} \text{minimize} & f(x) + \mu\phi(x) \\ \text{subject to} & h_j(x) = 0 \qquad j = 1, 2, \dots, n_h \end{array} \tag{1.28}$$

where μ is a scalar parameter. Each iteration of a barrier method performs Newton minimization steps on (1.28) for a particular value of μ. Under mild conditions, solutions to this modified problem converge to that of the original problem as $\mu \to 0$. A complete development of barrier methods, including proofs of universality and performance, as well as a number of complete algorithms, is given in [124].

There are several practical roadblocks to the use of a barrier method. First of all, the author knows of no publicly-available, *general-purpose* barrier solver; someone wishing to use this technique would have to write their own. Even if a barrier solver is found, the user must supply code to compute the value and derivatives of the barrier function. Furthermore, determining a valid barrier function is not always trivial, particularly if the inequality constraints are nondifferentiable.

Cutting plane methods A localization or cutting-plane method such as ACCPM [135] replaces the inequality constraint set $\mathcal{S}$ (1.27) with a *cutting-plane oracle*: a construct that, given a point $x \notin \mathcal{S}$, returns a separating hyperplane between x and $\mathcal{S}$. These methods require the computation of subgradient information from each of the functions f and g_i, and the user is expected to supply code to compute this information. The primary advantage of these methods is their immediate support for nondifferentiable functions. However, their performance is usually inferior to the other methods mentioned here, though they often lend themselves to distributed computation.

1.2.3 Applications

A wide variety of practical applications for convex programming have already been discovered, and the list is steadily growing. Perhaps the field in which the application of convex programming is the most mature and pervasive is control theory; see [50, 24, 25, 39, 114] for a sample of these applications. Other fields where applications for convex optimization are known include

- robotics [141, 32, 80];
- pattern analysis and data mining, including support vector machines [149, 181, 137, 166, 91];
- combinatorial optimization and graph theory [105, 3, 28, 71, 62, 55, 92, 177, 183, 108, 46, 57, 56, 81];
- structural optimization [2, 10, 15, 12, 13, 182, 88, 14, 11, 93];
- algebraic geometry [133, 97, 99, 98];
- signal processing [87, 170, 66, 147, 151, 165, 47, 152, 4, 51, 106, 145, 64, 156, 61, 154, 115, 109, 130, 48, 131, 43, 9, 67, 172, 164, 157, 180];
- communications and information theory [23, 100, 41, 138, 79, 150, 1];
- networking [19, 21];
- circuit design [160, 161, 83, 42, 82, 26];
- quantum computation [45];
- neural networks [132];
- chemical engineering [140];
- economics and finance [72, 175].

This list, while large, is certainly incomplete, and excludes applications where only LP or QP is employed. A list including LP and QP would be significantly larger; and yet convex programming is of course a generalization of these technologies.

One promising source of new applications for convex programming is the extension and enhancement of existing applications for linear programming. An example of this is *robust linear programming*, which allows uncertainties in the coefficients of an LP model to be accounted for in the solution of the problem, by transforming it into a nonlinear CP [103]. This approach produces robust solutions more quickly, and arguably more reliably, than using Monte Carlo methods. Presumably, robust linear programming would find application anywhere linear programming is currently employed, and where uncertainty in the model poses a significant concern.

Some may argue that our prognosis of the usefulness of convex programming is optimistic, but there is good reason to believe the number of applications is in fact being underestimated. We can appeal to the history of linear programming as precedent. George Dantzig first published his invention of the simplex method for linear programming in the 1947; and while a number of military applications were soon found, it was not until 1955-1960 that the field enjoyed robust growth [40]. Certainly, this delay was in large part due to the dearth of adequate computational resources; but that is the point: the discovery of new applications accelerated only once hardware and software advances made it truly practical to solve LPs.

Similarly, then, there is good reason to believe that the number of applications for convex programming will rise dramatically if it can be made easier to create, analyze, and solve CPs.

1.2.4 Convexity and differentiability

As mentioned in §1.2.2, many solvers for smooth (nonconvex) NLPs can be used effectively to solve many smooth CPs. An arguable case can be made that the advance knowledge of convexity is not critical in such cases. Dedicated solvers for smooth CPs do provide some advantages, such as reliable detection of infeasibility and degeneracy; see, for example, [178]. But such advantages may not immediately seem compelling to those accustomed to traditional nonlinear programming.

In the nonsmooth case, the situation is markedly different. Nondifferentiability poses a significant problem for traditional nonlinear programming. The best methods available to solve nondifferentiable NLPs are far less accurate, reliable, or efficient than their smooth counterparts. The documentation for the GAMS modeling framework "strongly discourages" the specification of nonsmooth problems, instead recommending that points of nondifferentiability be eliminated by replacing them with Boolean variables and expressions [30], thereby converting a nondifferentiable NLP to mixed-integer NLP (MINLP). But doing so is not always straightforward, and introduces significant practical complexities of a different sort.

In contrast, there is nothing in *theory* that prevents a nonsmooth CP from being solved as efficiently as a smooth CP. For example, as mentioned in §1.2.1, the proof provided by Nesterov and

Nemirovsky of the existence of barrier functions for CPs does not depend on smoothness considerations. And nonsmooth CPs can often be converted to an equivalent smooth problem with a carefully chosen transformation—consider the ℓ_∞, ℓ_1, and largest-L norm minimization problems presented in §1.1. Of course, neither the construction of a valid barrier function nor the smoothing transformation is always (or even often) obvious.

One might ask: just how often are the CPs encountered in practice nonsmooth? We claim that it is quite often. Non-trivial SDPs and SOCPs, for example, are nonsmooth. Common convex functions such as the absolute value and most norms are nonsmooth. Examining the current inventory of applications for convex programming, and excluding those that immediately present themselves as LPs and QPs, smoothness is the exception, not the rule.

Thus a convex programming methodology that provides truly practical support for nonsmooth problems is of genuine practical benefit. If such a solution can be achieved, then the *a priori* distinction between convexity and nonconvexity becomes far more important, because the need to avoid nondifferentiability remains only in the nonconvex case.

1.3 Convexity verification

Given the benefits of *a priori* knowledge of convexity, it follows that it would be genuinely useful to perform automatic *convexity verification*: that is, to determine whether or not a given mathematical program is convex. Unfortunately, in general, this task is at least as difficult as solving nonconvex problems: that is, it is theoretically intractable. Every practical approach to the problem necessarily involves a compromise of generality, reliability, automation, or some combination thereof:

- *generality*: the approach supports only a well-defined subset of convex programs.
- *reliability*: the approach fails to make conclusive determinations in some cases.
- *automation*: the approach requires at least some assistance from the user.

The approach taken in disciplined convex programming is no exception here. In this section, we introduce a number of other approaches, each involving a different set of these compromises.

1.3.1 Smooth NLP analysis

An ambitious approach to convexity detection for smooth nonlinear programs has been developed by Crusius [38] and Orban and Fourer [128]. The former has been refined and integrated into a commercial offering [121, 63]; the latter is part of a larger effort to analyze and classify many types of structure found in mathematical programs, not just convexity. While these systems were developed independently, and each includes unique features, they are similar enough in concept and operation that we discuss them together here.

These systems proceed in two stages. In the first, the constraints are analyzed using interval methods and other techniques to collect bounds on the feasible region. The process is iterative in nature: information from one constraint often allows tighter bounds to be computed from other constraints. This stage continues until the bounds can no longer be improved.

In the second stage, key subexpressions of the objective and constraint functions are analyzed to determine if they are convex over this approximate feasible region. This is accomplished by symbolically differentiating the expressions and bounding the values of their Hessians. As with any symbolic differentiation system, the systems are limited to those functions contained in a library with relevant derivative information. But in both cases, the library is reasonably large; and, in the Crusius case, relatively easy to expand.

The analysis produces one of three results:

- all Hessians are positive semidefinite over the feasible region: the problem is convex.
- any Hessian is negative definite over the feasible region: the problem is nonconvex.
- otherwise: no conclusive determination can be made.

Note that the third case does *not* mean that the problem is convex or nonconvex; just that the system was unable to reach a conclusion.

Limiting the convexity determination to the approximate feasible region allows a number of problems to be accepted as convex that do not satisfy a strict reading of the definition we give in §1.2, in which the overall feasible region is not considered when judging the convexity of a constraint. To illustrate the distinction, consider the trivial problem

$$\begin{array}{ll} \text{minimize} & x^3 \\ \text{subject to} & x \geq 0. \end{array} \tag{1.29}$$

In this case, the objective function is clearly not convex. However, it *is* convex over the interval enforced by the constraint $x \geq 0$, and the Crusius/DrAmpl systems would easily detect this fact. The problem would be accepted as convex by these systems, but not according to our definition in §1.2. This more inclusive definition is certainly attractive—as long as numerical methods applied to such problems are careful to remain within the region where local convexity is assured.

The Crusius/DrAmpl approach represents an impressive combination of computational techniques, and often achieves surprising results by determining convexity where it would not be reasonably expected. For example, the Crusius system successfully concludes that the problem

$$\begin{array}{ll} \text{minimize} & \sin(e^x) \\ \text{subject to} & 2\log(2x-1) \geq x \\ & x^3 \leq 1 \end{array} \tag{1.30}$$

is convex [38]—which, to the author at least, seems far from obvious. Nevertheless, the inability of this approach to reliably make conclusive determinations in all cases limits its usefulness in contexts where convexity is an advance requirement. It would seem that these systems are most useful when convexity is *not* a strict requirement, but can still be exploited when found. This interpretation coincides well with the way that the methods are being deployed.

As stated above, both systems are designed to work only with smooth NLPs. Crusius briefly discusses the nonsmooth case, providing a single example of how a problem can be transformed from nonsmooth form to smooth form. But he does not address the general problem or propose how this type of transformation could be automated. The techniques presented in this dissertation could perhaps be used to extend these methods to certain nonsmooth problems.

1.3.2 Parsing methods

Many modeling frameworks, including AMPL and GAMS, automatically and reliably determine if a model is a linear program, enabling specialized algorithms to be selected for them automatically [59, 30]. Similar approaches are employed by modeling tools such as SDPSOL and LMITOOL to verify SDPs automatically [169, 53].

While we do not know for sure how AMPL, GAMS, and LMITOOL accomplish this task, we do know that it can be easily performed by a *parser*, which is the approach SDPSOL takes. In simple terms, a *parser* is an algorithm that analyzes the structure of a text file or data structure to determine if it conforms to a set of pattern rules known as a *grammar*. For example, to determine that a mathematical program is an LP, it is necessary to confirm that the numerical expressions in the objective functions and constraints are affine in the parameters. A set of grammar rules to codify this might include:

- Variables are trivially affine, as are constant expressions.
- Sums and differences of affine expressions are affine.
- The product of a constant expression and an affine expression is affine.

In a parser grammar, what is *not* included is as important as what is: for example, the product of two non-constant affine expressions is *not* affine. So, for example, if `x` and `y` are variables, then the expression `2 x + 2 y / 3 - y` would satisfy the grammar rules and would be confirmed as affine; but the expression `x y + x` would not.

It is important to understand that parsing methods rely on an analysis of *textual* structure, and by themselves lack any sort of *mathematical* sophistication or knowledge. This can work quite well for standard forms like LP and SDP, where textual structure is sufficient to make conclusive determinations, but for more general problems it can be too limited. For example, a parser can easily be designed to recognize arbitrary quadratic expressions; but except in limited special cases, it will not be able to determine if the expression is convex. On the other hand, unlike the mathematically sophisticated approach in §1.3.1, a parser is deterministic, and can make conclusive determinations in every case: either a problem obeys the grammar, or it does not. Thus while the class of problems itself may be limited, true and reliable *verification* of that problem class is possible.

1.3.3 Empirical analysis

Yet another alternative is provided by MPROBE [33], a system that employs numerical sampling to determine *empirically* the shapes of constraint and objective functions. It will conclusively *disprove* linearity or convexity in many cases, but it can never conclusively prove convexity, because doing so would require an exhaustive search. To be fair, its author makes no claims to that effect, instead promoting MPROBE as a useful tool for interactively assisting the user to make his own decisions. It is a powerful tool for that purpose, and it is certainly conceivable that such an interactive approach could provide a practically sufficient determination of convexity in many cases.

1.4 Modeling frameworks

The purpose of a *modeling framework* is to enable someone to become a proficient *user* of a particular mathematical technology (*e.g.*, convex programming) without requiring that they become an expert in it (*e.g.*, interior-point methods). It accomplishes this by providing a convenient interface for specifying problems, and then by automating many of the underlying mathematical and computational steps for analyzing and solving them.

A number of excellent modeling frameworks for LP, QP, and NLP are in widespread use. The primary benefits of these frameworks include:

- *Convenient problem specification.* Most modeling frameworks, including including AMPL [59], GAMS [30], and LINGO [101], are built around a custom *modeling language* that allows models to be specified in a text file using a natural mathematical syntax. Other frameworks, such as What'sBest! [102] and Frontline [63], use spreadsheets as a natural, graphical user interface.

- *Standard form detection.* Most employ the techniques described in §1.3.2 to differentiate between LPs, QPs, and NLPs automatically, allowing specialized solvers to be selected.

- *Automatic differentiation.* Frameworks that support smooth NLPs can compute the derivatives of the objective and constraint functions for use by the solver.

- *Solver control.* The frameworks can automatically call the solver, passing it the problem data in proper form and supplying reasonable default values for the solver's configuration parameters. The user can override those parameters or force the use of a particular solver, if desired.

Thanks to these features, modeling frameworks greatly simplify the analysis and solution of LPs, QPs, and NLPs, and have had a broad and positive impact on the use of optimization in many application areas. These frameworks are well-suited for solving smooth CPs as well.

A few modeling tools designed for convex programming have been developed, but are limited to certain standard forms and tailored for particular applications. For example, several tools have been developed to simplify the specification and solution of semidefinite programs (SDPs) or linear matrix inequalities (LMIs), including SDPSOL [169], LMITool [53], MATLAB's LMI Control Toolbox [110],

```
maximize     entropy( x1, x2, x3, x4 );
subject to   a11 x1 + a12 x2 + a13 x3 + a14 x4 = b1;
             a21 x1 + a22 x2 + a23 x3 + a24 x4 = b2;
                 x1 +     x2 +     x3 +     x4 = 1;
parameters   a11, a12, a13, a14, b1,
             a21, a22, a23, a24, b2;
variables    x1 >= 0, x2 >= 0, x3 >= 0, x4 >= 0;
function entropy( ... ) concave;
```

Figure 1.2: An example CP in the `cvx` modeling language.

YALMIP [104], and SOSTOOLS [136]. These tools are used by thousands in control design, analysis, and research, and in other fields as well. Undoubtedly, semidefinite programming would not be so widely used without such tools, even though efficient codes for solving SDPs are widely available.

We are designing a new modeling framework, called `cvx`, to support disciplined convex programming, and have already completed a preliminary version which demonstrates the feasibility of the key principles of the methodology. `cvx` is built around a modeling language that allows optimization problems to be expressed using a relatively obvious mathematical syntax. The language shares a number of basic features with other modeling languages like AMPL or GAMS, such as parameter and variable declarations, common mathematical operations, and so forth. See Figure 1.2 for an example of a simple entropy maximization problem expressed in the `cvx` syntax.

Throughout this dissertation, we illustrate various concepts with examples rendered in the `cvx` modeling language, using a fixed-width font (`example`) to distinguish them. However, because `cvx` is still in development, it is possible that future versions of the language will use a slightly different syntax; and a few examples use features of the language that are planned but not yet implemented.

1.5 Synopsis

The purpose of this dissertation is to introduce disciplined convex programming, and to enumerate its advantages for both the applications-oriented modeler and the advanced convex optimization researcher. Several of the specific challenges discussed here are addressed, including automatic solver selection and construction (1.2.2), support for nonsmooth problems (1.2.4), and automatic convexity verification (1.3). These challenges are in fact advances toward a larger goal of *unification.* There are no less than *seven* commonly recognized standard forms for convex programming: LS, LP, QP, SDP, SOCP, GP, and smooth CP. Some are special cases of others; for example, SDP can be used to represent LS, LP, QP, and SOCP, though with some compromise in efficiency. Deciding which form best suits a given application is not always obvious; and for some applications, no standard form is suitable, and only a custom form will suffice. Thus a primary goal of disciplined convex programming is to eliminate the need for modelers to perform such classification themselves.

Chapter 2 extends the motivation provided in §1.1 by adding equality and bound constraints to the norm minimization problem (1.1). For each norm we determine a full solution to the problem, including numerical proofs of infeasibility and optimality. The aim of the development is to illustrate how solutions can be obtained using readily available numerical software. The chapter demonstrates that the task is genuinely feasible, but in many cases requires a significant amount of transformation and manipulation, just as in §1.1. This chapter provides further illustration of the expertise gap currently present in convex programming, and therefore of the challenges and opportunities that disciplined convex programming seeks to address.

Chapter 3 introduces the disciplined convex programming methodology in detail, from the perspective of the modeler: that is, the rules and conventions that must be followed to construct DCPs. We show that disciplined convex programming does not sacrifice generality; and we defend the notion that its design in fact supports the practices of those who regularly study and solve convex optimization problems today. Of particular note is §3.5, in which the concept of a *graph implementation* is introduced. Graph implementations are key to the support of nonsmooth convex programs. The chapter also shows how the conventions that DCPs follow simplify the verification of convexity.

Chapter 4 studies the analysis and solution of disciplined convex programs. Three primary challenges are identified and solved: conversion to canonical (*i.e.*, solvable) form, numerical solution, and recovery of dual information. In each case, we find that the conventions imposed by disciplined convex programming provide opportunities for automation and efficiency. We use concrete examples from the `cvx` modeling framework to illustrate the feasibility of the approaches taken.

Chapter 5 provides concluding remarks, including a summary of the results presented and a discussion of areas of promising future study and development.

1.6 Conventions

In this dissertation, we have chosen to follow the lead of Rockafellar [142] and adopted the extended-valued approach to defining convex and concave functions with limited domains; *e.g.*,

$$f : \mathbb{R} \to (\mathbb{R} \cup +\infty), \qquad f(x) = \begin{cases} +\infty & x < 0 \\ x^{1.5} & x \geq 0 \end{cases} \tag{1.31}$$

$$g : \mathbb{R} \to (\mathbb{R} \cup -\infty), \qquad g(x) = \begin{cases} -\infty & x \leq 0 \\ \log x & x > 0 \end{cases} \tag{1.32}$$

Using extended-valued functions simplifies many of the derivations and proofs. Still, we will on occasion use the **dom** operator to refer to the set of domain values that yield finite results:

$$\mathbf{dom}\, f = \{\, x \mid f(x) < +\infty \,\} = [0, +\infty) \tag{1.33}$$

$$\mathbf{dom}\, g = \{\, x \mid g(x) > -\infty \,\} = (0, +\infty) \tag{1.34}$$

Chapter 2

Motivation

2.1 Introduction

In this chapter, we expand the exercise undertaken in §1.1 by considering the *constrained* norm minimization problem

$$\begin{array}{rl} \mathcal{P}\text{: minimize} & f(x) \triangleq \|Ax - b\| \\ \text{subject to} & l \leq x \leq u \\ & Cx = d \end{array} \tag{2.1}$$

The optimization variable is the vector $x \in \mathbb{R}^n$, and the parameters are the quantities

$$A \in \mathbb{R}^{m \times n}, \quad b \in \mathbb{R}^m, \quad l \in \mathbb{R}^n, \quad u \in \mathbb{R}^n, \quad C \in \mathbb{R}^{p \times n}, \quad d \in \mathbb{R}^p \tag{2.2}$$

and a norm $\|\cdot\| : \mathbb{R}^m \to [0, +\infty)$. We consider the same set of norms $\|\cdot\|$ that we used in §1.1. We say that $\mathcal{P}$ is *solved* once the following determinations are made:

- *Feasibility.* Determine if $\mathcal{P}$ is *feasible*; that is, find at least one point $x \in \mathbb{R}^n$ that satisfies the constraints $l \leq x \leq u$ and $Cx = d$, or prove that such a point does not exist.

- *Optimality.* If $\mathcal{P}$ is feasible, find a value x that satisfies the constraints and that comes within a provable tolerance $\epsilon > 0$ of minimizing $\|Ax - b\|$.

- *Sensitivity.* If $\mathcal{P}$ is feasible, determine how the minimum norm would change if small changes were made to the values of b, l, u, and d.

Our goal is to illustrate how someone with an understanding of convex optimization—but *without* the ability or desire to implement numerical algorithms—can solve $\mathcal{P}$. Specifically, we assume that said person has access to a modeling framework such as AMPL [59] or GAMS [30] tied to a single *prototypical* solver—that is, a hypothetical solver chosen to represent the capabilities of common, readily available software packages.

We begin by providing a general analysis of $\mathcal{P}$, in the process providing a concrete numerical definition for each of the three determinations listed above. We then introduce the prototypical solver, describing its capabilities, its input format, and the results it returns. Then, for each norm $\|\cdot\|$, we describe in detail how to solve $\mathcal{P}$. Specifically, we describe in detail the transformations required to convert $\mathcal{P}$ into a form that the solver will accept, and how to map the results that the solver produces back to the original problem.

While this chapter is primarily a *Gedanken* (thought) experiment, we hope that the reader will ultimately agree that it provides a reasonable illustration of the practical intricacies of solving even a relatively simple convex program such as $\mathcal{P}$.

2.2 Analysis

As in §1.1, we assume that $m \geq n$ and that A has full rank. We also assume that $b \notin \text{Range}(A)$, so $Ax \neq b$ for all $x \in \mathbb{R}^m$. In addition, we partition the parameter matrices by rows; *i.e.*,

$$A \triangleq \begin{bmatrix} a_1^T \\ a_2^T \\ \vdots \\ a_m^T \end{bmatrix} \quad b \triangleq \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \quad l \triangleq \begin{bmatrix} l_1 \\ l_2 \\ \vdots \\ l_n \end{bmatrix} \quad u \triangleq \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ v_n \end{bmatrix} \quad C \triangleq \begin{bmatrix} c_1^T \\ c_2^T \\ \vdots \\ c_p^T \end{bmatrix} \quad d \triangleq \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_p \end{bmatrix}. \tag{2.3}$$

We continue to refer to the quantities $a_i^T x - b_i$ as the *residuals*.

2.2.1 The dual problem

Providing a full solution to $\mathcal{P}$ requires information from the dual problem, so let us derive it. The Lagrangian L is

$$L(x, \lambda_l, \lambda_u, \nu) = f(x) + \lambda_l^T(l - x) + \lambda_u^T(x - u) + \nu^T(Cx - d), \tag{2.4}$$

where $\lambda_l \in \mathbb{R}_+^n$, $\lambda_u \in \mathbb{R}_+^n$, and $\nu \in \mathbb{R}^p$ are Lagrange multipliers for the constraints $x \geq l$, $x \leq u$, and $Cx = d$, respectively. The dual function g is

$$\begin{aligned} g(\lambda_l, \lambda_u, \nu) &\triangleq \inf_x L(x, \lambda_l, \lambda_u, \nu) = l^T\lambda_l - u^T\lambda_u - d^T\nu - \left(\sup_x\, (\lambda_l - \lambda_u - C^T\nu)^T x - f(x)\right) \\ &= l^T\lambda_l - u^T\lambda_u - d^T\nu - f^*(\lambda_l - \lambda_u - C^T\nu) \end{aligned} \tag{2.5}$$

so the Lagrange dual problem is

$$\begin{array}{ll} \text{maximize} & g(\lambda_l, \lambda_u, \nu) = l^T\lambda_l - u^T\lambda_u - d^T\nu - f^*(\lambda_l - \lambda_u - C^T\nu) \\ \text{subject to} & \lambda_l, \lambda_u \geq 0. \end{array} \tag{2.6}$$

The function $f^* : \mathbb{R}^n \to \mathbb{R} \cup +\infty$ is the conjugate of f.

Lemma 1. *The conjugate function f^* is given by*

$$f^*(w) = \inf \left\{ b^T z \;\mid\; \|z\|_* \leq 1,\; A^T z = w \right\} \tag{2.7}$$

Proof. First let us recall the definition of the dual norm:

$$\|z\|_* \triangleq \sup_{\|x\| \leq 1} x^T z \quad \Longleftrightarrow \quad \|x\| = \sup_{\|z\|_* \leq 1} x^T z \tag{2.8}$$

The conjugate function is therefore

$$\begin{aligned} f^*(w) &\triangleq \sup_x w^T x - f(x) = \sup_x w^T x - \|Ax - b\| \\ &= \sup_x w^T x - \sup_{\|z\|_* \leq 1} z^T (Ax - b) = \sup_x \inf_{\|z\|_* \leq 1} w^T x - z^T (Ax - b) \\ &= \sup_x \inf_{\|z\|_* \leq 1} (w - A^T z)^T x + b^T z = \inf_{\|z\|_* \leq 1} \sup_x b^T z + (w - A^T z)^T x \\ &= \inf \left\{ b^T z \;\mid\; \|z\|_* \leq 1,\; A^T z = w \right\} \end{aligned} \tag{2.9}$$

The swap of inf and sup is justified because the optimization problem involved obtains strong duality via a Slater condition. □

Note that f^* and therefore g are extended-valued:

$$\mathbf{dom}\; g = \left\{ (\lambda_l, \lambda_u, \nu) \in \mathbb{R}^p \times \mathbb{R}^n \times \mathbb{R}^m \;\mid\; \exists z \; A^T z = \lambda_l - \lambda_u - C^T \nu,\; \|z\|_* \leq 1 \right\}. \tag{2.10}$$

Removing these implicit constraints from the objective and incorporating them into the problem yields a more practical form for the dual problem:

$$\begin{array}{rl} \mathcal{D}\text{: maximize} & \bar{g}(\lambda_l, \lambda_u, \nu, z) \triangleq l^T \lambda_l - u^T \lambda_u - d^T \nu - b^T z \\ \text{subject to} & \lambda_l - \lambda_u - C^T \nu - A^T z = 0 \\ & \lambda_l, \lambda_u \geq 0 \\ & \|z\|_* \leq 1 \end{array} \tag{2.11}$$

This is the form of the dual that we employ throughout this development.

2.2.2 Definitions

The *feasible set* $\mathcal{F}_\mathcal{P}$ is the set of *feasible points* of $\mathcal{P}$; that is, the set of points that satisfy the constraints of $\mathcal{P}$:

$$\mathcal{F}_\mathcal{P} \triangleq \{ x \in \mathbb{R}^n \;\mid\; l \leq x \leq u,\; Cx = d \}. \tag{2.12}$$

We say that $\mathcal{P}$ *feasible* if $\mathcal{F}_\mathcal{P} \neq \emptyset$, and *infeasible* otherwise. Similarly, the feasible set $\mathcal{F}_\mathcal{D}$ is the set of feasible points of $\mathcal{D}$; that is, the points that satisfy the constraints of $\mathcal{D}$:

$$\mathcal{F}_\mathcal{D} \triangleq \left\{ (\lambda_l, \lambda_u, \nu, z) \in \mathbb{R}^p \times \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \;\middle|\; \begin{array}{c} \lambda_l - \lambda_u - C^T\nu - A^T z = 0 \\ \lambda_l, \lambda_u \geq 0 \quad \|z\|_* \leq 1 \end{array} \right\}. \tag{2.13}$$

The all-zero vector is contained in $\mathcal{F}_\mathcal{D}$, so we know that $\mathcal{D}$ is feasible. Both $\mathcal{F}_\mathcal{D}$ and $\mathcal{F}_\mathcal{P}$ satisfy constraint qualifications—the former because it is polyhedral, the latter because it has a non-empty interior. These qualifications ensure strong duality and a number of other useful properties.

The *optimal value* p^* of $\mathcal{P}$ is

$$p^* = \inf_{x \in \mathcal{F}_\mathcal{P}} \|Ax - b\|. \tag{2.14}$$

By convention, $p^* = +\infty$ if $\mathcal{P}$ is infeasible. The nonnegativity of the objective function $\|Ax - b\|$ ensures that $p^* \geq 0$. The *optimal set* $\widehat{\mathcal{F}}_\mathcal{P}$ is the set of feasible points that achieve the optimal value:

$$\widehat{\mathcal{F}}_\mathcal{P} \triangleq \{\, x \in \mathcal{F}_\mathcal{P} \;|\; \|Ax - b\| = p^* \,\}. \tag{2.15}$$

A point $x^* \in \widehat{\mathcal{F}}_\mathcal{P}$ is called an *optimal point* of $\mathcal{P}$. Similarly, the optimal value d^* of $\mathcal{D}$ is

$$d^* = \sup_{(\lambda_l, \lambda_u, \nu, z) \in \mathcal{F}_\mathcal{D}} l^T\lambda_l - u^T\lambda_u - d^T\nu - b^T z. \tag{2.16}$$

The all-zero vector produces an objective value of 0, so we know that $d^* \geq 0$. $\mathcal{D}$ may also be unbounded above, in which case $d^* = +\infty$. The optimal set $\widehat{\mathcal{F}}_\mathcal{D}$ is the set of feasible points that achieve this optimal value:

$$\widehat{\mathcal{F}}_\mathcal{D} \triangleq \{\, (\lambda_l, \lambda_u, \nu, z) \in \mathcal{F}_\mathcal{D} \;|\; l^T\lambda_l - u^T\lambda_u - d^T\nu - b^T z = d^* \,\}. \tag{2.17}$$

2.2.3 Feasibility

As mentioned above, the dual problem $\mathcal{D}$ is feasible; but $\mathcal{P}$ is not necessarily so. A strict theorem of alternatives can be constructed regarding the feasibility of $\mathcal{P}$.

Lemma 2. *$\mathcal{P}$ is infeasible (i.e., $\mathcal{F}_\mathcal{P} = \emptyset$) if and only if $\mathcal{U}_\mathcal{D} \neq \emptyset$, where*

$$\mathcal{U}_\mathcal{D} \triangleq \left\{ (\lambda_l, \lambda_u, \nu) \in \mathbb{R}^p \times \mathbb{R}^n \times \mathbb{R}^n \;\middle|\; \begin{array}{c} l^T\lambda_l - u^T\lambda_u - d^T\nu > 0 \\ \lambda_l - \lambda_u - C^T\nu = 0,\ \lambda_l, \lambda_u \geq 0 \end{array} \right\}. \tag{2.18}$$

Proof. Consider the following primal/dual pair:

$$\begin{array}{llcll} \mathcal{P}_{\text{feas}}\text{: minimize} & 0 & & \mathcal{D}_{\text{feas}}\text{: minimize} & l^T\lambda_l - u^T\lambda_u - d^T\nu \\ \text{subject to} & l \leq x \leq u & & \text{subject to} & \lambda_l - \lambda_u - C^T\nu = 0 \\ & Cx = d & & & \lambda_l, \lambda_u \geq 0. \end{array} \tag{2.19}$$

$\mathcal{P}$ is feasible if and only if $\mathcal{P}_{\text{feas}}$ is feasible. Since $\mathcal{D}_{\text{feas}}$ is a feasible linear program, (2.19) is a strongly dual pair; so $\mathcal{P}_{\text{feas}}$ is feasible if and only if $\mathcal{D}_{\text{feas}}$ is bounded. The feasible set of $\mathcal{D}_{\text{feas}}$ is a cone—that is, it is closed under positive scalar multiplication—so $\mathcal{D}_{\text{feas}}$ is unbounded if and only if at least one feasible point has a positive objective—that is, if and only if $\mathcal{U}_{\mathcal{D}}$ is non-empty. □

We call a triple $(\lambda_l, \lambda_u, \nu) \in \mathcal{U}_{\mathcal{D}}$ a *certificate of infeasibility* for $\mathcal{P}$. If a numerical algorithm encounters such a point, it may immediately terminate, having obtained conclusive proof of infeasibility.

2.2.4 Optimality

Let us now assume that $\mathcal{P}$ is feasible. Given any point $x \in \mathcal{F}_{\mathcal{P}}$ and any point $(\lambda_l, \lambda_u, \nu, z) \in \mathcal{F}_{\mathcal{D}}$, the *duality gap* is the difference between their objective values, and is provably nonnegative:

$$\begin{aligned}
&f(x) - \bar{g}(\lambda_l, \lambda_u, \nu, z) \\
&\quad = \|Ax - b\| - l^T\lambda_l + u^T\lambda_u + d^T\nu + b^T z \\
&\quad = \|Ax - b\| - l^T\lambda_l + u^T\lambda_u + d^T\nu + b^T z + x^T(\lambda_l - \lambda_u - C^T\nu - A^T z) \\
&\quad = \|Ax - b\| - z^T(Ax - b) \qquad (z^T(Ax-b) \leq \|Ax-b\|\|z\|_*,\ \|z\|_* \leq 1) \\
&\qquad + (x - l)^T\lambda_l + (u - x)^T\lambda_u \qquad (l \leq x \leq u,\ \lambda_l, \lambda_u \geq 0) \\
&\qquad - \nu^T(Cx - d) \geq 0. \qquad (Cx = d).
\end{aligned} \tag{2.20}$$

This establishes the *weak duality* relationship between $\mathcal{P}$ and $\mathcal{D}$, and implies that $p^* \geq d^*$. Thus feasible dual points establish lower bounds on $\mathcal{P}$, and feasible primal points upper bounds on $\mathcal{D}$. Specifically, given any $x \in \mathcal{F}_{\mathcal{P}}$ and $(\lambda_l, \lambda_u, \nu, z) \in \mathcal{F}_{\mathcal{D}}$ satisfying $f(x) - \bar{g}(\lambda_l, \lambda_u, \nu, z) = \epsilon$,

$$f(x) = \|Ax - b\| \leq p^* + \epsilon \qquad \bar{g}(\lambda_l, \lambda_u, \nu, z) \geq d^* + \epsilon. \tag{2.21}$$

We call $(x, (\lambda_l, \lambda_u, \nu, z))$ a *certificate of ϵ-optimality* for $\mathcal{P}$.

In fact, as we have already stated, strong duality is achieved, so the optimal duality gap is zero: that is, $p^* = d^*$. Thus for *any* $\epsilon > 0$ there exists a certificate of ϵ-optimality $(x, (\lambda_l, \lambda_u, \nu, z)) \in \mathcal{F}_{\mathcal{P}} \times \mathcal{F}_{\mathcal{D}}$. (In fact, because the optimal sets $\widehat{\mathcal{F}}_{\mathcal{P}}$ and $\widehat{\mathcal{F}}_{\mathcal{D}}$ are non-empty, this holds for $\epsilon = 0$ as well.) Numerical algorithms can use this fact to create stopping criteria that guarantee a desired accuracy.

2.2.5 Sensitivity

Let $P : (\mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^p \times \mathbb{R}^m) \to (\mathbb{R} \cup +\infty)$ be a function whose value equals the solution to a perturbed version of $\mathcal{P}$; *i.e.*,

$$\begin{aligned}
P(\delta l, \delta u, \delta d, \delta b) &\triangleq \inf \{\, \|Ax - b - \delta b\| \;\mid\; l + \delta l \leq x \leq u + \delta u,\ Cx = d + \delta d \,\} \\
&= \sup_{(\lambda_l, \lambda_u, \nu, z) \in \mathcal{F}_{\mathcal{D}}} (l + \delta l)^T\lambda_l - (u + \delta u)^T\lambda_u - (d + \delta d)^T\nu - (b + \delta b)^T z
\end{aligned} \tag{2.22}$$

Note that $P(0,0,0,0) = p^*$, and $P(\cdot,\cdot,\cdot,\cdot) = +\infty$ whenever the given perturbation yields an infeasible problem. As the second form makes clear, P is convex, as it is the supremum of a (probably infinite) set of linear functions.

Now suppose that $P(0,0,0,0) < +\infty$ and $(\lambda_l, \lambda_u, \nu, z) \in \widehat{\mathcal{F}}_{\mathcal{D}}$; then we can calculate

$$\begin{aligned} P(\delta l, \delta u, \delta d, \delta b) &= \sup_{(\bar\lambda_l, \bar\lambda_u, \bar\nu, \bar z) \in \mathcal{F}_{\mathcal{D}}} (l+\delta l)^T \bar\lambda_l - (u+\delta u)^T \bar\lambda_u - (d+\delta d)^T \bar\nu - (b+\delta b)^T \bar z \\ &\geq (l+\delta l)^T \lambda_l - (u+\delta u)^T \lambda_u - (d+\delta d)^T \nu - (b+\delta b)^T z \\ &= p^* + \delta l^T \lambda_l - \delta u^T \lambda_u - \delta d^T \nu - \delta b^T z. \end{aligned} \tag{2.23}$$

So $(\lambda_l, -\lambda_u, -\nu, -z)$ is a subgradient of $P(l,u,d,b)$, and the optimal dual point provides a measure of sensitivity of the problem to perturbations in the parameter vectors. Of course, exact determination of an optimal dual point is usually not possible in practice; but for sufficiently small ϵ, a certificate of ϵ-optimality provides a useful approximation.

It is important to note that because (2.23) provides only a lower bound on the perturbed optimal value, it provides somewhat limited sensitivity information. As long as the original optimal point x^* remains feasible in the perturbed problem, then

$$\|Ax^* - b - \delta b\| \geq P(\delta l, \delta u, \delta d, \delta b) \geq P(0,0,0,0) + \delta l^T \lambda_l - \delta u^T \lambda_u - \delta d^T \nu - \delta b^T z$$

and a useful range of values is obtained. On the other hand, if x^* lies outside the new feasible set, then no upper bound on the increase can be obtained. In particular, if the feasible set for the perturbed problem is a strict subset of the original—say, if $\delta l \geq 0$ and $\delta u \leq 0$—then the optimal value must necessarily increase, and (2.23) provides a minimum, but not a maximum, for that increase.

2.2.6 Summary

We are now prepared to provide numerical definitions of the three determinations we listed in the introduction: feasibility, optimality, and sensitivity. Given numerical values of the parameters, a specification of the norm $\|\cdot\|$, and a small numerical tolerance ϵ, one of the following two *certificates* constitutes a complete solution:

- A *certificate of infeasibility* $(\lambda_l, \lambda_u, \nu) \in \mathcal{U}_{\mathcal{D}}$ satisfying

$$\lambda_l - \lambda_u - C^T \nu = 0 \qquad \lambda_l, \lambda_u \geq 0 \qquad l^T \lambda_l - u^T \lambda_u - d^T \nu > 0 \tag{2.24}$$

  and proving that $\mathcal{P}$ is infeasible.

- A *certificate of ϵ-optimality* $(x, (\lambda_l, \lambda_u, \nu, z)) \in \mathcal{F}_{\mathcal{P}} \times \mathcal{F}_{\mathcal{D}}$ satisfying

$$\begin{gathered} \lambda_l - \lambda_u - C^T \nu - A^T z = 0 \qquad \lambda_l, \lambda_u \geq 0 \qquad \|z\|_* \leq 1 \\ f(x) - \bar g(\lambda_l, \lambda_u, \nu, z) = \|Ax - b\| - l^T \lambda_l + u^T \lambda_u + d^T \nu + b^T z \leq \epsilon \end{gathered} \tag{2.25}$$

and proving that $\|Ax-b\|-p^* \leq \epsilon$. The values of λ_l, λ_u, ν, and z provide approximate costs of perturbing l, u, d, and b, respectively, according to formula (2.23).

2.3 The solver

In this chapter, we seek to illustrate the practice of solving $\mathcal{P}$ by using existing, readily available solver software. All such software requires that the problems they solve obey certain conventions, which invariably requires that the problems be *transformed* into a compliant form. Much of the development in the sections below centers around determining such transformations.

A wide variety of packages could be chosen for our task, and surveying them all would be well beyond the scope of our discussion. Instead we limit our view to a single prototypical solver, one that represents the capabilities of the majority of such packages. Our solver accepts problems in the following form:

$$\begin{array}{rll} \mathcal{P}_{\text{NLP}}\text{: minimize} & f_0(\bar{x}) & \\ \text{subject to} & f_k(\bar{x}) \leq 0, \ k=1,\dots,\bar{m} & \\ & \bar{C}\bar{x} = \bar{d} & \end{array} \tag{2.26}$$

We call (2.26) the *canonical form* for our solver. The problem variables for $\mathcal{P}_{\text{NLP}}$ are $\bar{x} \in \mathbb{R}^{\bar{n}}$, and the parameters include the quantities $\bar{C} \in \mathbb{R}^{\bar{p}\times\bar{n}}$, $\bar{d} \in \mathbb{R}^{\bar{p}}$, and $\bar{m}+1$ convex, *smooth* (*i.e.*, twice continuously differentiable) functions $f_k : \mathbb{R}^{\bar{n}} \rightarrow (\mathbb{R} \cup +\infty)$, $k=0,1,\dots,\bar{m}$.[1] The smoothness requirement is nearly universal among the best performing algorithms for nonlinear optimization, including the packages LOQO [162] and MOSEK [118]. The exceptions are for specific problem types that do not apply here. Indeed, it is this smoothness requirement that makes our development interesting, because it drives the transformations applied to $\mathcal{P}$.

Given a problem specification, the solver will determine solutions to $\mathcal{P}_{\text{NLP}}$ and its Wolfe dual,

$$\begin{array}{rl} \mathcal{D}_{\text{NLP}}\text{: maximize} & f_0(\bar{x}) + \sum_{k=1}^{m} \bar{\lambda}_k f_k(\bar{x}) + \bar{\nu}^T(\bar{C}\bar{x}-\bar{d}) \\ \text{subject to} & \nabla f_0(\bar{x}) + \sum_{k=1}^{m} \bar{\lambda}_k \nabla f_k(\bar{x}) + \bar{C}^T\bar{\nu} = 0 \\ & \bar{\lambda} \geq 0 \end{array} \tag{2.27}$$

The equality constraint allows the objective function to be expressed in the alternate form

$$f_0(\bar{x}) - \bar{x}^T\nabla f_0(\bar{x}) + \sum_{k=1}^{m} \bar{\lambda}_k (f_k(\bar{x}) - \bar{x}^T\nabla f_k(\bar{x})) - \bar{d}^T\bar{\nu} \tag{2.28}$$

This form often allows for significant simplification when many of the functions f_k are affine, so we use this alternate form in this chapter. The solver produces one of the following certificates:

[1]Our liberal use of bars over the variable and parameter names is to differentiate them from similar names used in $\mathcal{P}$ and $\mathcal{D}$.

- A *certificate of infeasibility*: a point $\bar{x}$ and a pair $(\bar{\lambda}, \bar{\nu})$ satisfying

$$\begin{gathered}\bar{\lambda} \geq 0 \qquad \sum_{k=1}^{m} \bar{\lambda}_k \nabla f_k(\bar{x}) + \bar{C}^T \bar{\nu} = 0 \\ \sum_{k=1}^{m} \bar{\lambda}_k f_k(\bar{x}) + \bar{\nu}^T (\bar{C}x - \bar{d}) = \sum_{k=1}^{m} \bar{\lambda}_k (f_k(\bar{x}) - \bar{x}^T \nabla f_k(\bar{x})) - \bar{d}^T \bar{\nu} > 0,\end{gathered} \tag{2.29}$$

  proving that $\mathcal{P}_{\text{NLP}}$ (2.26) is infeasible. If the constraint functions f_k are affine, then these conditions are actually independent of $\bar{x}$, so $\bar{x}$ need not be supplied.

- A *certificate of ϵ-optimality*: a feasible point $\bar{x}$ and a pair $(\bar{\lambda}, \bar{\nu})$ satisfying

$$\begin{gathered}\bar{\lambda} \geq 0 \qquad \nabla f_0(\bar{x}) + \sum_{k=1}^{m} \bar{\lambda}_k \nabla f_k(\bar{x}) + \bar{C}^T \bar{\nu} = 0 \\ -\sum_{k=1}^{m} \bar{\lambda}_k f_k(\bar{x}) = \bar{x}^T \nabla f_0(\bar{x}) - \sum_{k=1}^{m} \bar{\lambda}_k (f_k(\bar{x}) - \bar{x}^T \nabla f_k(\bar{x})) + \bar{d}^T \bar{\nu} \leq \epsilon,\end{gathered} \tag{2.30}$$

  proving that $f_0(\bar{x})$ is within ϵ of the optimal value.

Other scenarios are possible as well, such as an unbounded primal problem, but these are the only two scenarios that occur for the problems we are considering.

Despite the intentional similarity in notation, the dual information returned by the solver for the Wolfe dual will often be different from the dual information sought for the original problem $\mathcal{P}$. This is in part due to differences between the Wolfe and Lagrange duals, but primarily because many of the versions of $\mathcal{P}$ we are considering require significant transformation to convert them to canonical form, and those transformations alter the dual problem as well. So for each of the versions that we study, we show how to "recover" certificates of infeasibility (2.24) and ϵ-optimality (2.25) for $\mathcal{P}$ from corresponding certificates (2.29), (2.30) returned by the solver. For applications where dual information has a useful interpretation, this is an important step.

To solve a problem using this solver, the user must supply numerical values for $\bar{C}$ and $\bar{d}$, and definitions for each of the functions f_k, $k = 0, 1, \ldots, \bar{m}$. These definitions can take one of two forms:

- If f_k is *affine or quadratic*, the user may supply numerical values for a triple $(\bar{f}_k, \bar{g}_k, \bar{H}_k)$ such that $f_k(\bar{x}) \equiv \bar{f}_k + \bar{g}_k^T \bar{x} + \frac{1}{2} \bar{x}^T \bar{H}_k \bar{x}$. The matrix $\bar{H}_k$ must be symmetric and positive semidefinite. If f_k is affine, then $\bar{H}_k \equiv 0$.

- Otherwise, the user must supply *executable code*—say, written in C or FORTRAN—to compute the value $f_k(\bar{x})$, gradient $\nabla f_k(\bar{x})$, and Hessian $\nabla^2 f_k(\bar{x})$ at any desired point $\bar{x}$.

The requirement to supply executable code to compute the values and derivatives of nonlinear functions is a common and realistic one for standalone numerical solvers; and it is a requirement that would prove daunting for many users. Modeling frameworks such as AMPL [59] or GAMS [30] address this difficulty by providing a modeling language that allows the objective and constraint expressions to be written in a pseudo-mathematical syntax, and computes the derivatives automatically using symbolic differentiation methods.

Each of the sections §2.4-§2.9 is devoted to a single choice of the norm $\|\cdot\|$, and proceeds in a similar, proof-like fashion. First, we introduce the norm and its dual, and describe a set of transformations used to convert $\mathcal{P}$ into canonical form (2.26). Following this, we present the Wolfe dual (2.27), and prove by construction that certificates of infeasibility and ϵ-optimality for $\mathcal{P}$ can be recovered from the corresponding certificates for the Wolfe dual. In this way, we have effectively proven that the problem can be fully solved, as defined in §2.2.6, using our prototypical solver.

2.4 The smooth case

We begin with the Hölder or ℓ_p norm for $p \geq 2$, the dual of which is also a Hölder norm:

$$\|w\| \triangleq \|w\|_p = \left(\sum_{k=1}^m |w_k|^p\right)^{1/p} \qquad \|z\|_* \triangleq \|z\|_q = \left(\sum_{k=1}^m |z_k|^q\right)^{1/q} \qquad q \triangleq \frac{p}{p-1} \tag{2.31}$$

(We consider the $p < 2$ case in a later section.) This includes as a special case the most popular choice—the ℓ_2 or Euclidean norm, for which $\|z\|_* = \|z\|$. The norm $\|w\|_p$ is twice differentiable everywhere except $w = 0$. Because we have assumed that $b \notin \text{Range}(A)$, we know that this exception will not arise, so it is reasonable to solve the problem as if the objective is smooth. This approach is suggested for ℓ_2 problems by Vanderbei, the author of LOQO [162].

The conversion to canonical form (2.26) is rather trivial:

$$\begin{gathered} \bar{x} \triangleq x \qquad \bar{C} \triangleq C \qquad \bar{d} \triangleq d \\ f_0(x) \triangleq \|Ax - b\|_p \qquad \left.\begin{aligned} f_k(x) &\triangleq l_k - x_k \\ f_{k+n}(x) &\triangleq x_k - u_k \end{aligned}\right\} k = 1, 2, \dots, m \end{gathered} \tag{2.32}$$

Code must be supplied to compute the value and derivatives of the objective function $f_0(x)$. Let us define the following quantities for convenience,

$$w \triangleq Ax - b \qquad W \triangleq \mathbf{diag}(|w_1|^{p-2}, |w_2|^{p-2}, \dots, |w_m|^{p-2}) \qquad \tilde{w} \triangleq Ww, \tag{2.33}$$

enabling us to express the derivatives of f_0 as follows:

$$\begin{aligned} \nabla f_0(x) &= \|w\|_p^{1-p} A^T \tilde{w} \\ \nabla^2 f_0(x) &= (p-1)\|w\|_p^{1-p} A^T (W - \|w\|_p^{-p} \tilde{w}\tilde{w}^T) A \\ &= (p-1)\|w\|_p^{1-p} A^T W A - (p-1)\|w\|_p^{-1} \nabla f_0(x) \nabla f_0(x)^T. \end{aligned} \tag{2.34}$$

A `C++` code excerpt that uses the `BLAS` linear algebra library to compute f_0 and its derivatives for

```
dcopy_( m, b, 1, w, 1 );
dgemv_( "N", m, n, 1.0, A, m, x, 1, -1.0, w, 1 );
f = dnrm2_( m, w, 1 );
dgemv_( "T", m, n, 1.0 / f, A, m, w, 1, 0.0, g, 1 );
dsyrk_( "U", "N", n, m, 1.0 / f, A, m, 0.0, H, n );
dsyr_( "U", n, -1.0 / f, g, 1, H, n );
```

Figure 2.1: C++ code to compute the value, gradient, and Hessian of $f(x) = \|Ax - b\|_2$.

the $p = 2$ case is given in Figure 2.1. (Note that for $p = 2$, $W \equiv I$ and $\tilde{w} \equiv w$.) For simplicity we have given only the computational code, omitting interface and memory management.

By defining $\bar{\lambda} \triangleq [\lambda_l^T \quad \lambda_u^T]$ and noting that

$$\begin{aligned} f_0(x) - x^T \nabla f_0(x) &= \|w\|_p - \|w\|_p^{1-p} x^T A^T \tilde{w} \\ &= \|w\|_p - \|w\|_p^{1-p} (w+b)^T \tilde{w} = -\|w\|_p^{1-p} b^T \tilde{w} \end{aligned} \tag{2.35}$$

we may express the Wolfe dual (2.27) as follows:

$$\begin{array}{ll} \text{maximize} & -\|w\|_p^{1-p} b^T \tilde{w} + l^T \lambda_l^T - u^T \lambda_u - d^T \nu \\ \text{subject to} & \|w\|_p^{1-p} A^T \tilde{w} - \lambda_l + \lambda_u + C^T \nu = 0 \\ & \lambda_l, \lambda_u \geq 0 \end{array} \tag{2.36}$$

The conditions for the solver-generated certificate of infeasibility (2.29) become

$$\lambda_l, \lambda_u \geq 0 \qquad -\lambda_l + \lambda_u + C^T \nu = 0 \qquad l^T \lambda_u - u^T \lambda_u - d^T \nu > 0 \tag{2.37}$$

These coincide precisely with the conditions given in (2.24) for $\mathcal{P}$.

Similarly, the conditions for the solver-generated certificate of ϵ-optimality (2.30) become

$$\begin{gathered} \|w\|_p^{1-p} A^T \tilde{w} - \lambda_l + \lambda_u + C^T \nu = 0 \qquad \lambda_l, \lambda_u \geq 0 \\ \|w\|_p^{1-p} x^T A^T \tilde{w} - l^T \lambda_l^T + u^T \lambda_u + d^T \nu \leq \epsilon \end{gathered} \tag{2.38}$$

Defining $z \triangleq \|w\|_p^{1-p} \tilde{w}$ produces

$$\begin{gathered} -\lambda_l + \lambda_u + C^T \nu + A^T z = 0 \qquad \lambda_l, \lambda_u \geq 0 \\ z^T A x - l^T \lambda_l^T + u^T \lambda_u + d^T \nu \leq \epsilon \end{gathered} \tag{2.39}$$

Note that z satisfies

$$\begin{gathered} z^T (Ax - b) = \|w\|_p^{1-p} \tilde{w}^T w = \|w\|_p = \|Ax - b\|_p \\ \|z\|_q = \|w\|_p^{1-p} \left(\textstyle\sum_{k=1}^m |\tilde{w}_k|^q\right)^{1/q} = \|w\|_p^{1-p} \left(\textstyle\sum_{k=1}^m |w_k|^p\right)^{(p-1)/p} = 1, \end{gathered} \tag{2.40}$$

so the last inequality yields

$$\begin{aligned}\epsilon &\geq z^T Ax - l^T\lambda_l^T + u^T\lambda_u + d^T\nu \\ &= z^T(Ax-b) - l^T\lambda_l^T + u^T\lambda_u + d^T\nu + b^T z \\ &= \|Ax-b\|_p - l^T\lambda_l + u^T\lambda_u + d^T\nu + b^T z.\end{aligned} \tag{2.41}$$

So (2.39) and (2.41) coincide with the ϵ-optimality conditions conditions (2.25) for $\mathcal{P}$. In fact, they are slightly stricter because $\|z\|_q = 1$, even though only $\|z\|_q \leq 1$ is required. Then $(x, (\lambda_l, \lambda_u, \nu, z \triangleq \|w\|_p^{1-p}\tilde{w}))$ is a valid certificate of ϵ-optimality for $\mathcal{P}$.

2.5 The Euclidean norm

For the Euclidean norm, a common alternative is to square the objective function, producing the modified problem

$$\begin{array}{rl}\text{minimize} & \|Ax-b\|_2^2 = (Ax-b)^T(Ax-b) \\ \text{subject to} & l \leq x \leq u \\ & Cx = d.\end{array} \tag{2.42}$$

If a robust smooth CP solver is being employed, there is little reason to perform this transformation. However, (2.42) is a convex *quadratic* program, and a wider variety of solvers can handle such problems; so this form is a natural choice for many without access to more advanced solvers. Furthermore, this transformation simplifies the presentation of the problem to our prototypical solver, because it allows us to use a numerical representation of the objective function

$$f_0(x) = \bar{f}_0 + \bar{g}_0^T x + \frac{1}{2}x^T\bar{H}_0 x, \quad \bar{f}_0 \triangleq b^T b, \quad \bar{g}_0 \triangleq -2A^T b, \quad \bar{H}_0 \triangleq 2A^T A \tag{2.43}$$

instead of requiring that executable code be supplied to compute it. In all other respects, the conversion to canonical form matches the previous case (2.32).

If we define $\bar{\lambda} \triangleq [\lambda_l^T \quad \lambda_u]^T$ as before, and note that

$$f_0(x) - x^T\nabla f_0(x) = (Ax-b)^T(Ax-b) - 2x^T A^T(Ax-b) = -(Ax+b)^T(Ax-b), \tag{2.44}$$

then we may express the Wolfe dual (2.27) as follows:

$$\begin{array}{rl}\text{maximize} & -(Ax+b)^T(Ax-b) + l^T\lambda_l^T - u^T\lambda_u - d^T\nu \\ \text{subject to} & 2A^T(Ax-b) - \lambda_l + \lambda_u + C^T\nu = 0 \\ & \lambda_l, \lambda_u \geq 0.\end{array} \tag{2.45}$$

The certificates of infeasibility produced by the solver for this problem are identical to those obtained in §2.4; but the certificates of ϵ-optimality have changed because of the altered objective function.

The conditions (2.30) become

$$\begin{aligned} 2A^T(Ax-b) - \lambda_l + \lambda_u + C^T\nu = 0 \qquad \lambda_l, \lambda_u \geq 0 \\ 2x^TA^T(Ax-b) - l^T\lambda_l^T + u^T\lambda_u + d^T\nu \leq \epsilon. \end{aligned} \tag{2.46}$$

Defining $z \triangleq 2(Ax-b)$ produces

$$\begin{aligned} -\lambda_l + \lambda_u + C^T\nu + A^Tz = 0 \qquad \lambda_l, \lambda_u \geq 0 \\ z^TAx - l^T\lambda_l^T + u^T\lambda_u + d^T\nu \leq \epsilon. \end{aligned} \tag{2.47}$$

Note that $\|z\|_2 = 2\|Ax-b\|_2$ and $z^T(Ax-b) = 2\|Ax-b\|_2^2$, so the final inequality becomes

$$\begin{aligned} \epsilon &\geq z^TAx - l^T\lambda_l^T + u^T\lambda_u + d^T\nu \\ &= z^T(Ax-b+b) - l^T\lambda_l^T + u^T\lambda_u + d^T\nu \\ &= 2\|Ax-b\|_2^2 - l^T\lambda_l + u^T\lambda_u + d^T\nu + b^Tz. \end{aligned} \tag{2.48}$$

If we introduce a scaled version of these dual variables and of ϵ,

$$(\tilde{\lambda}_l, \tilde{\lambda}_u, \tilde{\nu}, \tilde{z}) \triangleq \alpha(\lambda_l, \lambda_u, \nu, z), \quad \tilde{\epsilon} \triangleq \alpha\epsilon \qquad \alpha \triangleq 1/(2\|Ax-b\|_2), \tag{2.49}$$

then we find that $\|\tilde{z}\|_2 = 1$ and

$$\begin{aligned} A^T\tilde{z} - \tilde{\lambda}_l + \tilde{\lambda}_u + C^T\tilde{\nu} = 0 \qquad \tilde{\lambda}_l, \tilde{\lambda}_u \geq 0 \\ \|Ax-b\|_2 - l^T\tilde{\lambda}_l + u^T\tilde{\lambda}_u + d^T\tilde{\nu} + b^T\tilde{z} \leq \tilde{\epsilon}. \end{aligned} \tag{2.50}$$

These conditions coincide with the ϵ-optimality conditions (2.25) for $\mathcal{P}$—except, again, the condition $\|\tilde{z}\|_2 = 1$ is in fact stricter. Thus we have produced a valid certificate of $\tilde{\epsilon}$-optimality for $\mathcal{P}$.

2.6 The Manhattan norm

Another popular choice for the norm $\|\cdot\|$ is the ℓ_1 norm, often called the *Manhattan* or *taxicab* norm. Its dual is the ℓ_∞ norm:

$$\|w\| \triangleq \|w\|_1 \triangleq \sum_{k=1}^m \|w_k\| \qquad \|z\|_* \triangleq \|z\|_\infty = \max_{1\leq k\leq m} |z_k|. \tag{2.51}$$

This norm is not differentiable and cannot easily be made so; a direct method is therefore impossible. There is, however, a simple and relatively common transformation that allows this problem to be solved efficiently. First, let us recognize that

$$\|w\|_1 \leq v \quad \Longleftrightarrow \quad \exists w_+, w_- \geq 0 \quad \begin{array}{c} w_+ - w_- = w \\ \vec{1}^T(w_+ + w_-) = v \end{array} \tag{2.52}$$

We can use this equivalence to transform $\mathcal{P}$ into a linear program:

$$\mathcal{P} \quad \Longrightarrow \quad \begin{array}{rl} \text{minimize} & v \\ \text{subject to} & \|Ax - b\|_1 \leq v \\ & Cx = d \\ & l \leq x \leq u \end{array} \quad \Longrightarrow \quad \begin{array}{rl} \text{minimize} & \vec{1}^T(w_+ + w_-) \\ \text{subject to} & Ax - b = w_+ - w_- \\ & Cx = d \\ & l \leq x \leq u \\ & w_+, w_- \geq 0 \end{array} \tag{2.53}$$

The transformation is shown in two steps. The first exploits the equivalence between a function and its epigraph $f(x) \triangleq \inf \{ v \mid (x, v) \in \operatorname{epi} f \}$ to move the nonlinearity out of the objective function and into a nonlinear inequality constraint. The second uses (2.52) to replace this constraint with an alternate form, in this case $m + 1$ linear equations and $2m$ nonnegative variables. This two-step process is duplicated for several of the norms in this chapter.

Because the objective and constraint functions are affine, no code writing is required; and the conversion to canonical form is relatively simple:

$$\bar{x} = \begin{bmatrix} x \\ w_+ \\ w_- \end{bmatrix} \quad \begin{array}{l} f_0(\bar{x}) = \vec{1}^T(w_+ + w_-) \\ \left.\begin{array}{r} f_k(\bar{x}) = l_k - x_k \\ f_{k+n}(\bar{x}) = x_k - u_k \end{array}\right\} k = 1, 2, \dots, n \\ \left.\begin{array}{r} f_{k+2n}(\bar{x}) = -w_{+,k} \\ f_{k+2n+m}(\bar{x}) = -w_{-,k} \end{array}\right\} k = 1, 2, \dots, m \end{array} \quad \bar{C} \triangleq \begin{bmatrix} A & -I & I \\ C & 0 & 0 \end{bmatrix}, \quad \bar{d} \triangleq \begin{bmatrix} b \\ d \end{bmatrix} \tag{2.54}$$

If we define $\bar{\lambda} \triangleq [\lambda_l^T \quad \lambda_u^T \quad \lambda_+^T \quad \lambda_-^T]^T$ and $\bar{\nu} \triangleq [\nu^T \quad z^T]^T$, the Wolfe dual (2.27) becomes

$$\begin{array}{rl} \text{maximize} & l^T\lambda_l - u^T\lambda_u - d^T\nu - b^Tz \\ \text{subject to} & -\lambda_l - \lambda_u + C^T\nu + A^Tz = 0 \\ & \vec{1} - \lambda_+ - z = 0 \\ & \vec{1} - \lambda_- - z = 0 \\ & \lambda_l, \lambda_u, \lambda_+, \lambda_- \geq 0 \end{array} \tag{2.55}$$

The solver-generated certificate of infeasibility (2.29) satisfies

$$\begin{array}{cc} -\lambda_l + \lambda_u + A^Tz + C^T\nu = 0 & \lambda_l, \lambda_u, \lambda_+, \lambda_- \geq 0 \\ -\lambda_+ - z = -\lambda_- + z = 0 & l^T\lambda_l - u^T\lambda_u - b^Tz - d^T\nu > 0 \end{array} \tag{2.56}$$

The conditions on z, λ_+, and λ_- reduce as follows:

$$-\lambda_+ - z = -\lambda_- + z = 0 \quad\Longrightarrow\quad 0 \leq \lambda_- = z = -\lambda_+ \leq 0 \quad\Longrightarrow\quad z = \lambda_+ = \lambda_- = 0 \tag{2.57}$$

Incorporating these results produces the simplified conditions

$$-\lambda_l + \lambda_u + C^T\nu = 0 \qquad \lambda_l, \lambda_u \geq 0 \qquad l^T\lambda_l - u^T\lambda_u - d^T\nu > 0 \tag{2.58}$$

These coincide precisely with (2.24), so $(\lambda_l, \lambda_u, \nu)$ is a valid certificate of infeasibility for $\mathcal{P}$.

Similarly, the solver-generated certificate of ϵ-optimality for (2.30) satisfies

$$\begin{array}{rc} -\lambda_l + \lambda_u + C^T\nu + A^Tz = 0 & \lambda_l, \lambda_u, \lambda_+, \lambda_- \geq 0 \\ \vec{1} - \lambda_+ - z = 0 & \vec{1}^T(w_+ + w_-) - l^T\lambda_l + u^T\lambda_u + d^T\nu + b^Tz \leq \epsilon \\ \vec{1} - \lambda_- + z = 0 & \end{array} \tag{2.59}$$

The conditions on z, λ_+, and λ_- reduce as follows:

$$\begin{array}{l} \vec{1} - \lambda_+ - z = 0 \\ \vec{1} - \lambda_- + z = 0 \end{array} \quad\Longrightarrow\quad -\vec{1} + \lambda_- = z = \vec{1} - \lambda_+ \quad\Longrightarrow\quad -\vec{1} \leq z \leq \vec{1} \quad\Longrightarrow\quad \|z\|_\infty \leq 1 \tag{2.60}$$

Incorporating these results and the fact that $\vec{1}^T(w_+ + w_-) \geq \|Ax - b\|_1$ yields

$$\begin{array}{cc} -\lambda_l + \lambda_u + C^T\nu + A^Tz = 0 & \lambda_l, \lambda_u \geq 0 \\ \|z\|_\infty \leq 1 & \|Ax - b\|_1 - l^T\lambda_l + u^T\lambda_u + d^T\nu + b^Tz \leq \epsilon \end{array} \tag{2.61}$$

which coincides with (2.25); so $(x, (\lambda_l, \lambda_u, \nu, z))$ is a valid certificate of ϵ-optimality for $\mathcal{P}$.

2.7 The Chebyshev norm

Another popular choice for the norm $\|\cdot\|$ is the ℓ_∞ norm, often called the *Chebyshev* or *peak* norm, the dual of which is the ℓ_1 norm:

$$\|w\| \triangleq \|w\|_\infty = \max_{1\leq k\leq m} |w_k| \qquad \|z\|_* \triangleq \|w\|_1 = \sum_{k=1}^m |w_k|. \tag{2.62}$$

Like the ℓ_1 case, this problem can be solved as a linear program. First, let us recognize that

$$\|w\|_\infty \leq q \quad\Longleftrightarrow\quad |w_k| \leq q,\ \forall 1 \leq k \leq n \quad\Longleftrightarrow\quad \exists w_+, w_- \geq 0 \quad \begin{array}{l} w_+ - w_- = w \\ w_+ + w_- = q\vec{1}. \end{array} \tag{2.63}$$

We can use this equivalence to transform $\mathcal{P}$ into a linear program:

$$\mathcal{P} \implies \begin{array}{rl} \text{minimize} & q \\ \text{subject to} & \|Ax-b\|_\infty \leq q \\ & Cx = d \\ & l \leq x \leq u \end{array} \implies \begin{array}{rl} \text{minimize} & q \\ \text{subject to} & Ax - b = w_+ - w_- \\ & w_+ + w_- = q\vec{1} \\ & Cx = d \\ & l \leq x \leq u \\ & w_+, w_- \geq 0. \end{array} \tag{2.64}$$

The nonlinearity has been replaced with 1 free variable, $2m$ nonnegative variables, and $2m+1$ equations. Conversion to canonical form (2.26) is similar to the ℓ_1 case:

$$\bar{x} \triangleq \begin{bmatrix} x \\ q \\ w_+ \\ w_- \end{bmatrix} \quad \begin{array}{r} f_0(\bar{x}) \triangleq q \\ \left.\begin{array}{r} f_k(\bar{x}) \triangleq l_k - x_k \\ f_{k+n}(\bar{x}) \triangleq x_k - u_k \end{array}\right\} k = 1,2,\dots,n \\ \left.\begin{array}{r} f_{k+2n}(\bar{x}) \triangleq -w_{+,k} \\ f_{k+2n+m}(\bar{x}) \triangleq -w_{-,k} \end{array}\right\} k = 1,2,\dots,m \end{array} \quad \bar{C} \triangleq \begin{bmatrix} A & 0 & -I & +I \\ 0 & -\vec{1} & I & -I \\ C & 0 & 0 & 0 \end{bmatrix} \quad \bar{d} = \begin{bmatrix} b \\ 0 \\ d \end{bmatrix}. \tag{2.65}$$

If we define $\bar{\lambda} \triangleq [\lambda_l^T \quad \lambda_u^T \quad \lambda_+^T \quad \lambda_-^T]^T$ and $\bar{\nu} \triangleq [\nu^T \quad r \quad z^T]^T$, the Wolfe dual (2.27) becomes

$$\begin{array}{rl} \text{maximize} & l^T\lambda_l - u^T\lambda_u - d^T\nu - b^Tz \\ \text{subject to} & -\lambda_l - \lambda_u + C^T\nu + A^Tz = 0 \\ & 1 - \vec{1}^Tr = 0 \\ & -\lambda_+ - z + r = 0 \\ & -\lambda_- - z + r = 0 \\ & \lambda_l, \lambda_u, \lambda_+, \lambda_- \geq 0. \end{array} \tag{2.66}$$

The solver-generated certificate of infeasibility (2.29) satisfies

$$\begin{array}{rr} -\lambda_l + \lambda_u + A^Tz + C^T\nu = 0 & \\ -\vec{1}^Tr = 0 & \lambda_l, \lambda_u, \lambda_-, \lambda_+ \geq 0 \\ -\lambda_+ - z + r = 0 & l^T\lambda_u - u^T\lambda_u - b^Tz - d^T\nu > 0. \\ -\lambda_- + z + r = 0 & \end{array} \tag{2.67}$$

The conditions on z, λ_+, λ_-, and r reduce as follows:

$$\begin{array}{r} \vec{1}^Tr = 0 \\ -\lambda_+ - z + r = 0 \\ -\lambda_- + z + r = 0 \end{array} \implies \begin{array}{c} \vec{1}^Tr = 0 \\ -r \leq z \leq +r \end{array} \implies r = z = \lambda_+ = \lambda_- = 0. \tag{2.68}$$

Incorporating these results produces

$$-\lambda_l + \lambda_u + A^T z + C^T \nu = 0 \qquad \lambda_l, \lambda_u \geq 0 \qquad l^T \lambda_l - u^T \lambda_u - d^T \nu > 0 \tag{2.69}$$

which coincides with (2.24); so $(\lambda_l, \lambda_u, \nu)$ is a valid certificate of infeasibility for $\mathcal{P}$.

Similarly, the solver-generated certificate of ϵ-optimality (2.30) satisfies

$$\begin{array}{rr} -\lambda_l + \lambda_u + A^T z + C^T \nu = 0 & \\ 1 - \vec{1}^T r = 0 & \lambda_l, \lambda_u, \lambda_-, \lambda_+ \geq 0 \\ -\lambda_+ - z + r = 0 & q - l^T \lambda_l + u^T \lambda_u + d^T \nu + b^T z \leq \epsilon. \\ -\lambda_- + z + r = 0 & \end{array} \tag{2.70}$$

The conditions on z, λ_+, λ_-, and r reduce as follows:

$$\begin{array}{r} 1 - \vec{1}^T r = 0 \\ -\lambda_+ - z + r = 0 \\ -\lambda_- + z + r = 0 \end{array} \implies \begin{array}{c} \vec{1}^T r = 1 \\ -r \leq z \leq +r \end{array} \implies \|z\|_1 \leq 1. \tag{2.71}$$

Incorporating these results and the fact that $q \geq \|Ax - b\|_\infty$ yields

$$\begin{array}{cc} -\lambda_l + \lambda_u + C^T \nu + A^T z = 0 & \lambda_l, \lambda_u \geq 0 \\ \|z\|_1 \leq 1 & \|Ax - b\|_\infty - l^T \lambda_l + u^T \lambda_u + d^T \nu + b^T z \leq \epsilon \end{array} \tag{2.72}$$

which coincides with (2.25); so $(x, (\lambda_l, \lambda_u, \nu, z))$ is a valid certificate of ϵ-optimality for $\mathcal{P}$.

2.8 The Hölder l_p norm

Now we revisit the Hölder or ℓ_p norm, this time allowing $p < 2$:

$$\|w\| \triangleq \|w\|_p = \left(\sum_{k=1}^{m} |w_k|^p \right)^{1/p} \qquad \|z\|_* \triangleq \|z\|_q = \left(\sum_{k=1}^{m} |z_k|^q \right)^{1/q} \qquad q \triangleq \frac{p}{p-1}. \tag{2.73}$$

The Hölder norm is common in facility location problems, and optimization methods involving such norms are the subject of considerable independent study; see, for example, [171].

For $1 < p < 2$, neither $f(x)$ nor $g(x) \triangleq f(x)^p$ is differentiable whenever any of the residuals are zero. A more aggressive transformation is required to eliminate this nondifferentiability and thereby convert the problem to solvable form. To begin, let us raise the objective to the p power, and rewrite the result as follows:

$$\mathcal{P} \quad \Longrightarrow \quad \begin{array}{ll} \text{minimize} & \|Ax-b\|_p^p \\ \text{subject to} & Cx = d \\ & l \leq x \leq u \end{array} \quad \Longrightarrow \quad \begin{array}{ll} \text{minimize} & \vec{1}^T v \\ \text{subject to} & Cx = d \\ & |a_k^T x - b_k|^p \leq v_k, \;\; k = 1,2,\dots,m \\ & l \leq x \leq u. \end{array} \tag{2.74}$$

The quantities $|a_k^T x - b_k|^p$ are not differentiable, but note that if $v > 0$,

$$|\bar{w}|^p \leq \bar{v} \quad \Longleftrightarrow \quad (\bar{w}^2)^p \leq \bar{v}^2 \quad \Longleftrightarrow \quad \phi_p(\bar{w}, \bar{v}) \leq 0 \tag{2.75}$$

where

$$\phi_p : (\mathbb{R} \times \mathbb{R}) \to (\mathbb{R} \cup +\infty), \quad \phi_p(\bar{v}, \bar{w}) \triangleq \begin{cases} (\bar{w}^2)^p/\bar{v} - \bar{v} & \bar{v} > 0 \\ +\infty & \bar{v} \leq 0. \end{cases} \tag{2.76}$$

It is not difficult to verify that ϕ_p is convex and twice differentiable. This result produces the form,

$$\begin{array}{ll} \text{minimize} & \vec{1}^T v \\ \text{subject to} & Ax - b = w \\ & Cx = d \\ & \phi_p(v_k, w_k) \leq 0, \;\; k = 1,2,\dots,m \\ & l \leq x \leq u. \end{array} \tag{2.77}$$

This transformed problem replaces the nondifferentiable objective with m additional variables and m smooth nonlinearities. Now if any $v_k = 0$, then the equivalency in (2.75) does not hold, so this problem is not precisely equivalent to $\mathcal{P}$. However, as we will see, this problem can still be used to construct solutions for $\mathcal{P}$. This is not surprising if one realizes that interior-point methods will ensure that $v > 0$ at all times as a natural consequence, so the equivalency (2.75) does hold in practice.

The converted problem (2.74) maps to the canonical form as follows:

$$\bar{x} \triangleq \begin{bmatrix} x \\ v \\ w \end{bmatrix} \quad \begin{array}{c} f_0(\bar{x}) = \vec{1}^T v \\ \left. \begin{array}{r} f_k(\bar{x}) = l_k - x_k \\ f_{k+n}(\bar{x}) = x_k - u_k \end{array} \right\} k = 1,2,\dots,n \\ f_{k+2n}(\bar{x}) = \phi(v_k, w_k) \quad k = 1,2,\dots,m \end{array} \quad \bar{C} \triangleq \begin{bmatrix} C & 0 \end{bmatrix} \quad \bar{d} \triangleq \begin{bmatrix} d \end{bmatrix}. \tag{2.78}$$

The constraint functions $f_{k+2n}(x)$, $k = 1,2,\dots,m$ are nonlinear, so their derivatives must be supplied. These can be determined rather simply from the derivatives of ϕ_p,

$$\nabla \phi_p(v,w) = \begin{bmatrix} -|w|^{2p}/v^2 - 1 \\ 2p|w|^{2p-2}w/v \end{bmatrix} \qquad \nabla^2 \phi_p(v,w) = \frac{2|w|^{2p-2}}{v} \begin{bmatrix} (w/v)^2 & -pw/v \\ -pw/v & p(2p-1) \end{bmatrix}. \tag{2.79}$$

If we define $\bar{\lambda} \triangleq [\lambda_w^T \quad \lambda_l^T \quad \lambda_u^T]^T$ and $\bar{\nu} \triangleq [z^T \quad \tilde{\nu}^T]^T$, and note that

$$\begin{aligned} \phi(v,w) - \begin{bmatrix} v & w \end{bmatrix} \nabla\phi(v,w) &= |w|^{2p}/v - b + |w|^{2p}/v + v - 2p|w|^{2p}/v \\ &= -2(p-1)|w|^{2p}/v, \end{aligned} \tag{2.80}$$

then the Wolfe dual (2.27) becomes

$$\begin{array}{ll} \text{maximize} & l^T\lambda_l - u^T\lambda_u - b^Tz - d^T\tilde{\nu} - \sum_{k=1}^m 2(p-1)\lambda_{w,k}|w_k|^{2p}/v_k \\ \text{subject to} & -\lambda_l + \lambda_u + C^T\tilde{\nu} + A^Tz = 0 \\ & \left.\begin{array}{l} 1 - \lambda_{w,k}(w_k^{2p}/v_k^2 + 1) = 0 \\ -z_k + 2p\lambda_{w,k}w_k^{2p-1}/v_k = 0 \end{array}\right\} k = 1,2,\dots,m \\ & \lambda_w, \lambda_l, \lambda_u \geq 0. \end{array} \tag{2.81}$$

For infeasible problems, the solver-generated certificate of infeasibility (2.29) satisfies

$$\begin{gathered} -\lambda_l + \lambda_u + C^T\tilde{\nu} + A^Tz = 0, \quad \lambda_l, \lambda_u, \lambda_w \geq 0 \\ \left.\begin{array}{r} -\lambda_{w,k}(|w_k|^{2p}/v_k^2 + 1) = 0 \\ -z_k + 2p\lambda_{w,k}|w_k|^{2p-2}w_k/v_k = 0 \end{array}\right\} k = 1,2,\dots,m \\ l^T\lambda_l - u^T\lambda_u - b^Tz - d^T\tilde{\nu} - \sum_{k=1}^m 2(p-1)\lambda_{w,k}|w_k|^{2p}/v_k > 0. \end{gathered} \tag{2.82}$$

It is not difficult to verify that the conditions on λ_w and z reduce as follows:

$$-\lambda_{w,k}(|w_k|^{2p}/v_k^2 + 1) = 0 \quad \Longrightarrow \quad \lambda_{w,k} = 0 \tag{2.83}$$

$$\lambda_{w,k} = 0, \quad -z_k + 2p\lambda_{w,k}|w_k|^{2p-2}w_k/v_k = 0 \quad \Longrightarrow \quad z_k = 0. \tag{2.84}$$

Incorporating these results produces the simplified conditions

$$-\lambda_l + \lambda_u + C^T\tilde{\nu} = 0 \qquad \lambda_l, \lambda_u \geq 0 \qquad l^T\lambda_l - u^T\lambda_u - d^T\tilde{\nu} - b^Tz > 0 \tag{2.85}$$

which coincide with (2.24); so $(\lambda_l, \lambda_u, \tilde{\nu})$ is a valid certificate of infeasibility for $\mathcal{P}$.

For feasible problems, the solver-generated certificate of infeasibility (2.30) satisfies

$$\begin{gathered} -\lambda_l + \lambda_u + C^T\tilde{\nu} + A^Tz = 0, \quad \lambda_l, \lambda_u, \lambda_w \geq 0 \\ \left.\begin{array}{r} 1 - \lambda_{w,k}(|w_k|^{2p}/v_k^2 + 1) = 0 \\ -z_k + 2p\lambda_{w,k}|w_k|^{2p-2}w_k/v_k = 0 \end{array}\right\} k = 1,2,\dots,m \\ -l^T\lambda_l + u^T\lambda_u + b^Tz + d^T\tilde{\nu} + \sum_{k=1}^m v_k + 2(p-1)\lambda_{w,k}|w_k|^{2p}/v_k \leq \epsilon. \end{gathered} \tag{2.86}$$

Completing this analysis requires three observations. First, the values of $\lambda_{w,k}$ are bounded:

$$|w_k|^{2p} \leq v_k^2, \quad (|w_k|^{2p}/v_k^2+1)\lambda_{w,k} = 1 \quad \Longrightarrow \quad 1/2 \leq \lambda_{w,k} \leq 1. \tag{2.87}$$

Second, $\|z\|_q$ may be bounded as follows:

$$\begin{aligned} &\lambda_{w,k}|w_k|^{2p} = (1-\lambda_{w,k})v_k^2, \quad 2p\lambda_{w,k}|w_k|^{2p-2}w_k = z_k v_k \\ &\quad\Longrightarrow \quad z_k = 2p\sqrt{\lambda_{w,k}(1-\lambda_{w,k})}|w_k|^{p-2}w_k \\ &\quad\Longrightarrow \quad |z_k| = 2p\sqrt{\lambda_{w,k}(1-\lambda_{w,k})}|w_k|^{p-1} \leq p|w_k|^{p-1} \\ &\quad\Longrightarrow \quad \|z\|_q^2 \leq \sum_{k=1}^m (p|w_k|^{p-1})^q = p^q\|w\|_p^p \quad \Longrightarrow \quad \|z\|_q \leq p\|w\|_p^{p-1}. \end{aligned} \tag{2.88}$$

The upper bound on $|z_k|$ is possible because the expression $\sqrt{\lambda_{w,k}(1-\lambda_{w,k})}$ achieves a maximum of $1/2$ at $\lambda_{w,k} = 1/2$. Third, another important quantity may be bounded:

$$\begin{aligned} &|w_k|^{2p} \leq v_p^2, \quad 1-\lambda_{w,k}(|w_k|^{2p}/v_k^2+1) = 0 \\ &\quad\Longrightarrow \quad v_k + 2(p-1)\lambda_{w,k}|w_k|^{2p}/v_k = (1+2(p-1)(1-\lambda_{w,k}))v_k \geq pv_k \geq p|w_k|^p \\ &\quad\Longrightarrow \quad \textstyle\sum_{k=1}^m v_k + 2(p-1)\lambda_{w,k}|w_k|^{2p}/v_k \geq p\|w\|_p^p. \end{aligned} \tag{2.89}$$

Using these results, and noting that $w = Ax-b$, we may rewrite the final inequality in (2.86) as

$$\epsilon \geq -l^T\lambda_l + u^T\lambda_u + d^T\tilde{\nu} + b^Tz + p\|Ax-b\|_p^p. \tag{2.90}$$

Now consider a simple scaling:

$$(\tilde{\lambda}_l, \tilde{\lambda}_u, \tilde{\nu}, \tilde{z}) \triangleq \alpha(\lambda_l, \lambda_u, \nu, z), \quad \tilde{\epsilon} \triangleq \alpha\epsilon, \quad \alpha \triangleq 1/(p\|Ax-b\|_p^{p-1}). \tag{2.91}$$

These scaled quantities satisfy the conditions

$$\begin{aligned} -\tilde{\lambda}_l + \tilde{\lambda}_u + C^T\tilde{\nu} + A^T\tilde{z} = 0, \quad \tilde{\lambda}_l, \tilde{\lambda}_u \geq 0, \qquad \|\tilde{z}\|_q \leq 1 \\ -l^T\tilde{\lambda}_l + u^T\tilde{\lambda}_u + d^T\tilde{\nu} + b^T\tilde{z} + \|Ax-b\|_p \leq \tilde{\epsilon}. \end{aligned} \tag{2.92}$$

which coincide with (2.25); so $(x, (\tilde{\lambda}_l, \tilde{\lambda}_u, \tilde{\nu}, \tilde{z}))$ is a valid certificate for $\tilde{\epsilon}$-optimality for $\mathcal{P}$.

2.9 The largest-L norm

Now let us consider the *largest-L* norm that we introduced in §1.1.5:

$$\|w\|_{[|L|]} \triangleq |w_{[|1|]}| + |w_{[|2|]}| + \cdots + |w_{[|L|]}|, \tag{2.93}$$

where $w_{[|k|]}$ is the k-th element of the vector after it has been sorted in descending order of absolute value. As already stated, this norm reduces to the ℓ_∞-norm when $L = 1$, and the ℓ_1-norm when $L = m$. In effect, it defines a "spectrum" of norm minimization problems with the ℓ_1 and ℓ_∞ norms at the extremes—one extreme (ℓ_1) favoring residual vectors with larger numbers of zero elements, the other (ℓ_∞) favoring residual vectors with more even distribution of values. The ℓ_p norm provides a similar tradeoff between these extremes, except in this case the norms are polyhedral.

The largest-L norm is nondifferentiable, but its polyhedral nature suggests a potential conversion to LP as with the ℓ_1 and ℓ_∞ norms. The conversion is not straightforward, however. Consider, for example, the following "exhaustive" definition:

$$\|w\|_{[L]} = \sup_{z\in\mathcal{E}_L} z^T w, \quad \mathcal{E}_L \triangleq \left\{\, z \in \mathbb{R}^n \ \middle|\ \textstyle\sum_{k=1}^n |z_k| = L,\ z_k \in \{-1, 0, +1\} \,\right\}. \tag{2.94}$$

This expression effectively takes the supremum over all possible combinations of L elements of the residual. But since $|\mathcal{E}_L| = 2^L n!/L!(n-L)!$, creating a separate inequality for each entry is hardly practical for problems of reasonable size. However, the set $\mathcal{E}_L$ is finite and the objective function is linear, so the value of the supremum remains the same when taken over the convex hull; *i.e.*

$$\|w\|_{[L]} = \sup_{z\in\mathbf{Conv}\,\mathcal{E}_L} z^T w = \sup\left\{\, w^T z \ \middle|\ \|z\|_1 \le L,\ \|z\|_\infty \le 1 \,\right\}. \tag{2.95}$$

This result makes evident that the dual norm is

$$\|z\|_{[L]*} \triangleq \max\{\|z\|_\infty, \|z\|_1/L\}, \tag{2.96}$$

and shows that $\|w\|_{[L]}$ is the optimal value of the convex program

$$\begin{array}{rl} \text{maximize} & w^T e \\ \text{subject to} & \|e\|_\infty \le 1 \\ & \|e\|_1 \le L. \end{array} \tag{2.97}$$

which can be converted to the linear program

$$\begin{array}{rl} \text{maximize} & w^T e \\ \text{subject to} & \vec{1}^T e_+ = L \\ & -e_+ \le e \le +e_+ \le \vec{1}. \end{array} \tag{2.98}$$

As a feasible linear program, strong duality is assured, so its dual obtains the same optimal value:

$$\begin{array}{rl} \text{minimize} & \vec{1}^T v + Lq \\ \text{subject to} & w_+ - w_- = w \\ & w_+ + w_- = v + \vec{1}q \\ & w_+, w_-, v \ge 0. \end{array} \tag{2.99}$$

We can now express the sorted value norm minimization problem as a linear program:

$$\begin{array}{ll} \text{minimize} & \vec{1}^T v + Lq \\ \text{subject to} & Ax - b = w_+ - w_- \\ & w_+ + w_- = v + \vec{1}q \\ & l \leq x \leq u, \;\; v, w_+, w_- \geq 0 \\ & Cx = d. \end{array} \tag{2.100}$$

This problem is only slightly more complex than the ℓ_∞ case—n new nonnegative variables have been added, and the number of equality constraints and other inequalities remains the same. Representing this problem in canonical form (2.26) requires

$$\begin{gathered} f_0(\bar{x}) = \vec{1}^T v + Lq \\ \bar{x} \triangleq \begin{bmatrix} x \\ v \\ q \\ w_+ \\ w_- \end{bmatrix} \quad \begin{array}{r} \left.\begin{array}{r} f_k(\bar{x}) = l_k - x_k \\ f_{k+n}(\bar{x}) = x_k - u_k \end{array}\right\} k = 1, 2, \dots, n \\ \left.\begin{array}{r} f_{k+2n}(\bar{x}) = -v_k \\ f_{k+2n+m}(\bar{x}) = -w_{+,k} \\ f_{k+2n+2m}(\bar{x}) = -w_{-,k} \end{array}\right\} k = 1, 2, \dots, m \end{array} \\ \bar{C} \triangleq \begin{bmatrix} A & 0 & 0 & -I & +I \\ 0 & -I & -\vec{1} & I & I \\ C & 0 & 0 & 0 & 0 \end{bmatrix} \quad \bar{d} \triangleq \begin{bmatrix} b \\ 0 \\ d \end{bmatrix}. \end{gathered} \tag{2.101}$$

If we define $\bar{\lambda} \triangleq [\lambda_l^T \quad \lambda_u^T \quad \lambda_v^T \quad \lambda_+^T \quad \lambda_-^T]^T$ and $\bar{\nu} \triangleq [z^T \quad r^T \quad \nu^T]^T$, the Wolfe dual (2.27) becomes

$$\begin{array}{ll} \text{maximize} & l^T\lambda_l - u^T\lambda_u - d^T\nu - b^T z \\ \text{subject to} & -\lambda_l - \lambda_u + C^T\nu + A^T z = 0 \\ & \vec{1} - \lambda_v - r = 0 \\ & L - \vec{1}^T r = 0 \\ & -\lambda_+ - z + r = 0 \\ & -\lambda_- + z + r = 0 \\ & \lambda_l, \lambda_u, \lambda_v, \lambda_+, \lambda_- \geq 0. \end{array} \tag{2.102}$$

The solver-generated certificate of infeasibility (2.29) satisfies

$$\begin{array}{rc} -\lambda_l + \lambda_u + A^T z + C^T \nu = 0 & \\ -\lambda_v - r = 0 & \lambda_l, \lambda_u, \lambda_v, \lambda_+, \lambda_- \geq 0 \\ -\vec{1}^T r = 0 & l^T\lambda_l - u^T\lambda_u - d^T\nu - b^T z > 0. \\ -\lambda_+ - z + r = 0 & \\ -\lambda_- + z + r = 0 & \end{array} \tag{2.103}$$

The conditions on z, r, λ_v, λ_+, and λ_- reduce as follows:

$$\begin{array}{c}\lambda_v \geq 0 \\ -\lambda_v - r = 0 \\ -\vec{1}^T r = 0\end{array} \implies \lambda_v = r = 0 \implies \begin{array}{c}-\lambda_+ - z = 0 \\ -\lambda_- - z = 0\end{array} \implies z = \lambda_+ = \lambda_- = 0. \tag{2.104}$$

Incorporating these results produces

$$-\lambda_l + \lambda_u + C^T\nu = 0 \qquad \lambda_l, \lambda_u \geq 0 \qquad -l^T\lambda_l + u^T\lambda_u - d^T\nu > 0 \tag{2.105}$$

which coincides with (2.24); so $(\lambda_l, \lambda_u, \nu)$ is a valid certificate of infeasibility for $\mathcal{P}$.

Similarly, the solver-generated certificate of optimality (2.30) satisfies

$$\begin{array}{c}-\lambda_l + \lambda_u + A^Tz + C^T\nu = 0 \\ -\lambda_v - r = 0 \\ -\vec{1}^T r = 0 \\ -\lambda_+ - z + r = 0 \\ -\lambda_- + z + r = 0\end{array} \qquad \begin{array}{c}\lambda_l, \lambda_u, \lambda_v, \lambda_+, \lambda_- \geq 0 \\ \vec{1}^T v + Lq - l^T\lambda_l + u^T\lambda_u + d^T\nu + b^Tz \leq \epsilon.\end{array} \tag{2.106}$$

The conditions on r and λ_v reduce as follows:

$$\begin{array}{c}\lambda_v \geq 0 \\ \vec{1} - \lambda_v - r = 0 \\ L - \vec{1}^T r = 0\end{array} \implies \begin{array}{c}\max_k r_k \leq 1 \\ \sum_k r_k = L.\end{array} \tag{2.107}$$

With this we can reduce the conditions on z, λ_+, and λ_- as follows:

$$\begin{array}{c}-\lambda_+ - z + r = 0 \\ -\lambda_- + z + r = 0\end{array} \implies -r \leq z \leq r \implies \begin{array}{c}\max_k |z_k| = \|z\|_\infty \leq 1 \\ \sum_k |z_k| = \|z\|_1 \leq L\end{array} \implies \|z\|_{[L]*} \leq 1. \tag{2.108}$$

This last step comes from a recognition of the dual norm in (2.96). Using this result and the recognition that $\vec{1}^T v + Lq \geq \|Ax - b\|_{[L]}$, yields

$$\begin{array}{cc}-\lambda_l + \lambda_u + C^T\nu + A^Tz = 0 & \lambda_l, \lambda_u \geq 0 \\ \|z\|_{[L]*} \leq 1 & \|Ax - b\|_{[L]} - l^T\lambda_l + u^T\lambda_u + d^T\nu + b^Tz \leq \epsilon\end{array} \tag{2.109}$$

which coincides with (2.25); so $(x, (\lambda_l, \lambda_u, \nu, z))$ is a valid certificate of ϵ-optimality for $\mathcal{P}$.

2.10 Conclusion

In this chapter we provided an analysis of a typical norm minimization problem, and considerable detail on how it can be solved by existing numerical software for a variety of choices of the norm.

In all but the first case, a non-trivial transformation is required to produce an equivalent problem that can be easily and efficiently solved. Some of those transformations may not necessarily be obvious to applications-oriented users. Such persons may, out of ignorance or practicality, restrict their view to norms for which solution methods are more widely known, such as ℓ_2 or ℓ_∞, even if doing so compromises the quality of their models.

The transformed versions of the norm minimization problem also present different challenges to the underlying numerical solvers:

- Linear programs (ℓ_∞, ℓ_1, sorted value);
- Quadratic programs (ℓ_2 in QP form),
- Linearly constrained smooth NLPs (ℓ_2 in direct form, ℓ_p $p \geq 2$);
- Nonlinearly constrained smooth NLPs (ℓ_p $p \geq 1$).

While we provided a prototypical solver that could handle all of these cases, in practice there is a definite hierarchy of preference and availability. Software packages for solving LPs and QPs are ubiquitous, well known, and mature. Promising options do exist for the other cases, but are less well known and less mature. So again, modelers may restrict their consideration to more common norms for which they have software that is familiar and simple to use.

The steps required to recover dual information for the original problem from the certificates generated by the prototypical solver vary in difficulty. In particular, the author spent quite some time proving the equivalence of the certificates for the ℓ_p case in §2.8. Modeling frameworks such as AMPL and GAMS perform a limited version of this dual recovery for problems that they support, which are converted internally to a canonical form. But no framework can be expected to take into account transformations performed *in advance* by users themselves. Our examples, unfortunately, demanded this sort of manual pre-processing. To be fair, not every application, nor every user, requires the use of dual information. But it begs the question: would it be used more frequently if it were easier to retrieve?

Finally, recall that the first example—the ℓ_p, $p \geq 2$ case in §2.4—required only trivial transformations to convert it to canonical form. This was a direct result of the fact that we were able to assume that the objective function was smooth. Conversely, the reason transformations were required for the remaining examples was that their objective functions were *not* smooth. In theory, convex optimization problems need not be differentiable in order for them to be solved efficiently. But as these examples illustrate, nondifferentiability remains a significant challenge in practice, specifically in the conversion to solvable form and the recovery of dual information. Disciplined convex programming provides a solution to this challenge.

Chapter 3

Disciplined convex programming

Disciplined convex programming is a methodology for constructing, analyzing, and solving convex optimization problems. It is inspired by the practices of those who regularly study and use convex optimization in research and applications. They do not simply construct constraints and objective functions without advance regard for convexity; rather, they draw from a mental library of functions and sets whose convexity properties are already known, and combine and manipulate them in ways that convex analysis ensures will produce convex results. If it proves necessary to determine the convexity properties of a new function or set from basic principles, that function or set is added to the mental library to be reused in other models.

Disciplined convex programming formalizes this strategy, and includes two key components:

- An *atom library* (§3.2): an *extensible* collection of functions and sets, or *atoms*, whose properties of shape (convex/concave/affine), monotonicity, and range are explicitly declared.
- A *ruleset* (§3.1), drawn from basic principles of convex analysis, that governs how atoms, variables, parameters, and numeric values can be combined to produce convex results.

A valid *disciplined convex program*, or DCP, is simply a mathematical program built in accordance with the ruleset using elements from the atom library. This methodology provides a teachable conceptual framework for people to use when studying and using convex programming, as well as an effective platform for building software to analyze and solve problems.

The ruleset, which we call the *DCP ruleset*, has been designed to be easy to learn and apply. As we show in §3.3, the task of verifying that a problem complies to the complexity rules can be performed in a straightforward and reliable manner. The rules constitute a set of sufficient conditions to guarantee convexity; any problem constructed in accordance with the convexity ruleset is guaranteed to be convex. The converse, however, is not true: it is possible to construct problems that do not obey the rules, but are convex nonetheless. Such problems are *not* valid DCPs, and the methodology does not attempt to accommodate them. This does not mean that they cannot be solved, but it does mean that they will have to be rewritten to comply with the ruleset.

In §3.4, we examine the practical consequences of the restricted approach that disciplined convex programming adopts. In particular, we provide some examples of some common and useful convex programs that, in their native form, are not *disciplined* convex programs. We show that the these limitations are readily remedied by appropriate additions to the atom library. We argue that the remedies are, in fact, consistent with the very thought process that disciplined convex programming is attempting to formalize.

In order to solve DCPs, the functions and sets in the atom library must each be provided with an *implementation*, a computer-friendly description of the atom that allows certain calculations to be performed upon it. The exact calculations depend upon the numerical algorithm employed, and might include derivatives, subgradients, cutting planes, barrier functions, and so forth. In §3.5, we describe such implementations in more detail, and introduce a new concept called a *graph implementation*. Graph implementations effectively allow functions and sets to be described in terms of other DCPs. This concept has some genuine tangible benefits, the most important being that it provides a natural way to define nondifferentiable functions, without the loss in performance typically associated with them, and without requiring specialized algorithms to be developed.

Much of disciplined convex programming can be understood and practiced at a purely conceptual level; DCPs can be completely described on paper. But of course, its ultimate purpose is to simplify the analysis and solution of convex programs via computer. For this reason, in this chapter we provide examples of various constructs in the `cvx` modeling language. However, some of the concepts introduced here have not yet been implemented, so we have made some sensible extensions as well. For this reason, the examples in this chapter should not be considered authoritative representations of the language, but rather more like artists' renditions.

3.1 The ruleset

The DCP ruleset governs how variables, parameters, and atoms (functions and sets) may be combined to form DCPs. DCPs are a strict subset of general CPs, so another way to say this is that the ruleset imposes a set of conventions or restrictions on CPs. The ruleset can be separated into four categories: *top-level rules*, *product-free rules*, *sign rules*, and *composition rules*.

3.1.1 Top-level rules

As the name implies, top-level rules govern the top-level structure of DCPs. These rules are more descriptive than they are restrictive, in the sense that nearly all *general* CPs follow these conventions anyway. But for completeness they must be explicitly stated.

Problem types. A valid DCP can be:

T1 a *minimization*: a convex objective and zero or more convex constraints;

T2 a *maximization*: a concave objective and zero or more convex constraints; or

$$\begin{array}{ll}
\textit{affine} \texttt{ = } \textit{affine} & \textbf{(T4)} \\
\textit{convex} \texttt{ <= } \textit{concave} \quad \text{or} \quad \textit{convex} \texttt{ < } \textit{concave} & \textbf{(T5)} \\
\textit{concave} \texttt{ >= } \textit{convex} \quad \text{or} \quad \textit{concave} \texttt{ > } \textit{convex} & \textbf{(T6)} \\
\texttt{(}\textit{affine}\texttt{,}\textit{affine}\texttt{,}\ldots\texttt{,}\textit{affine}\texttt{) in } \textit{convex set} & \textbf{(T7)}
\end{array}$$

Figure 3.1: Valid constraints.

T3 a *feasibility problem*: no objective and one or more convex constraints.

A valid DCP may also include any number of *assertions*; see rule **T9**.

At the moment, support for multiobjective problems and games has not been developed, but both are certainly reasonable choices for future work.

Constraints. See Figure 3.1. Valid constraints include:

T4 an equality constraint with affine left- and right-hand expressions.

T5 a less than ($<$,$\leq$) inequality, with a convex left-hand expression and a concave right-hand expression;

T6 a greater than ($>$,$\geq$) inequality, with a concave left-hand expression and a convex right-hand expression; or

T7 a set membership constraint $(lexp_1, \ldots, lexp_m) \in cset$, where $m \geq 1$, $lexp_1, \ldots, lexp_m$ are affine expressions, and *cset* is a convex set.

Non-equality ($\neq$) constraints and set non-membership ($\notin$) constraints are not permitted, because they are convex only in exceptional cases—and support for exceptional cases is anathema to the philosophy behind disciplined convex programming.

Constant expressions and assertions.

T8 Any well-posed numeric expression consisting only of numeric values and parameters is a valid constant expression.

T9 Any Boolean expression performing tests or comparisons on valid constant expressions is a valid assertion.

T10 If a function or set is parameterized, then those parameters must be valid constant expressions.

A *constant expression* is a numeric expression involving only numeric values and/or parameters; a *non-constant* expression depends on the value of at least one problem variable. Obviously a constant expression is trivially affine, convex, and concave. Constant expressions must be well-posed, which for our purposes means that they produce well-defined results for any set of parameter values that satisfy a problem's assertions.

An *assertion* resembles a constraint, but involves only constant expressions. They are not true constraints *per se*, because their truth or falsity is determined entirely by the numerical values supplied for a model's parameters, *before* the commencement of any numerical optimization algorithm. Assertions are not restricted in the manner that true constraints are; for example, non-equality ($\neq$) and set non-membership ($\notin$) operations may be freely employed. Assertions serve as *preconditions*, guaranteeing that a problem is numerically valid or physically meaningful. There are several reasons why an assertion may be wanted or needed; for example:

- to represent physical limits dictated by the model. For example, if a parameter w represents the physical weight of an object, an assertion $w > 0$ enforces the fact that the weight must be positive.
- to ensure numeric well-posedness. For example, if x, y, and z are variables and a, b, and c are parameters, then the inequality constraint $ax + by + z/c \leq 1$ is well-posed only if c is nonzero; this can be ensured by an assertion such as $c \neq 0$ or $c > 0$.
- to guarantee compliance with the preconditions attached to a function or set in the atom library. For example, a function $f_p(x) = \|x\|_p$ is parameterized by a value $p \geq 1$. If p is supplied as a parameter, then an assertion such as $p \geq 1$ would be required to guarantee that the function is being properly used. See §3.2.1 for information on how such preconditions are supplied in the atom library.
- to ensure compliance with the sign rules §3.1.3 or composition rules §3.1.4 below; see those sections and §3.3 for more details.

The final rule **T10** refers to functions or sets that are parameterized; *e.g.*,

$$f_p : \mathbb{R}^n \to \mathbb{R}, \quad f_p(x) = \|x\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{1/p} \tag{3.1}$$

$$B_p = \{\, x \in \mathbb{R}^n \mid \|x\|_p \leq 1 \,\} \tag{3.2}$$

and simply states that parameters such as p above must be constant. Of course this is generally assumed, but we must make it explicit for the purposes of computer implementation.

3.1.2 The product-free rules

Some of the most basic principles of convex analysis govern the sums and scaling of convex, concave, and affine expressions; for example:

- The sum of two or more convex (concave, affine) expressions is convex (concave, affine).
- The product of a convex (concave) expression and a nonnegative constant expression is convex (concave).

- The product of a convex (concave) expression and a nonpositive constant expression, or the simple negation of the former, is concave (convex).
- The product of an affine expression and any constant is affine.

Conspicuously absent from these principles is any mention of the *product* of convex or concave expressions. The reason for this is simple: there is no simple, general principle that can identify the curvature in such cases. For instance, suppose that x is a scalar variable; then:

- The expression $x \cdot x$, a product of two affine expressions, is convex.
- The expression $x \cdot \log x$, a product between an affine and a concave expression, is convex.
- The expression $x \cdot e^x$, a product between an affine and a convex expression, is neither convex nor concave.

For this reason, the most prominent structural convention enforced by disciplined convex programming is the prohibition of products (and related operations, like exponentiation) between non-constant expressions. The result is a set of rules appropriately called the *product-free rules*:

> *Product-free rule for numeric expressions:* All valid numeric expressions must be product-free; such expressions include:
>
> **PN1** A simple variable reference.
>
> **PN2** A *constant* expression.
>
> **PN3** A call to a function in the atom library. Each argument of the function must be a product-free expression.
>
> **PN4** The sum of two or more product-free expressions.
>
> **PN5** The difference of product-free expressions.
>
> **PN6** The negation of a product-free expression.
>
> **PN7** The product of a product-free expression and a constant expression.
>
> **PN8** The division of a product-free expression by a constant expression.
>
> We assume in each of these rules that the results are well-posed; for example, that dimensions are compatible.

In the scalar case, a compact way of restating these rules is to say that a valid numeric expression can be reduced to the form

$$a + \sum_{i=1}^{n} b_i x_i + \sum_{j=1}^{L} c_j f_j(arg_{j,1}, arg_{j,2}, \ldots, arg_{j,m_j}) \tag{3.3}$$

where a, b_i, c_j are constants; x_i are the problem variables; and $f_j : \mathbb{R}^{m_j} \rightarrow \mathbb{R}$ are functions from the atom library, and their arguments $arg_{j,k}$ are product-free expressions themselves. Certain special cases of (3.3) are notable: if $L = 0$, then (3.3) is a simple affine expression; if, in addition, $b_1 = b_2 = \cdots = b_n = 0$, then (3.3) is a constant expression.

For an illustration of the use of these product-free rules, suppose that a, b, and c are parameters; x, y, and z are variables; and $f(\cdot)$, $g(\cdot,\cdot)$, and $h(\cdot,\cdot,\cdot)$ are functions from the atom library. Then the expression

$$af(x) + y + (h(x, bg(y, z), c) - z + b)/a \tag{3.4}$$

satisfies the product-free rule, which can be seen by rewriting it as follows:

$$(b/a) + y + (-1/a)z + af(x) + (1/a)h(x, bg(y, z)c) \tag{3.5}$$

On the other hand, the following expression does not obey the product-free rule:

$$axy/2 - f(x)g(y, z) + h(x, y^b, z, c) \tag{3.6}$$

Now certainly, because these rules prohibit *all* products between non-constant expressions, some genuinely useful expressions such as quadratic forms like $x \cdot x$ are prohibited; see §3.4 for further discussion on this point.

For set expressions, a similar set of product-free rules apply:

> *Product-free rules for set expressions:* All valid set expressions used in constraints must be product-free; such expressions include:
>
> **PS1** A call to a convex set in the atom library.
>
> **PS2** A call to a function in the atom library. Each argument of the function must be a product-free set expression or a constant (numeric) expression.
>
> **PS3** The sum of product-free expressions, or of a product-free expression and a constant expression, or vice versa.
>
> **PS4** The difference between two product-free set expressions, or between a product-free set expression and a constant expression, or vice versa.
>
> **PS5** The negation of a product-free set expression.
>
> **PS6** The product of a product-free set expression and a constant expression.
>
> **PS7** The division of a product-free set expression by a constant expression.
>
> **PS8** The intersection of two or more product-free set expressions.
>
> **PS9** The Cartesian product of two or more product-free set expressions.
>
> We also assume in each of these rules that the results are well-posed; for example, that dimensions are compatible.

In other words, valid set expressions are reducible to the form

$$a + \sum_{i=1}^{n} b_i S_i + \sum_{j=1}^{L} c_j f_j(\mathit{arg}_{j,1}, \mathit{arg}_{j,2}, \ldots, \mathit{arg}_{j,m_j}) \tag{3.7}$$

where a, b_i, c_j are constants, and the quantities S_k are either sets from the atom library, or intersections and/or Cartesian products of valid set expressions. The functions $f_j : \mathbb{R}^{m_j} \to \mathbb{R}$ are functions in the atom library, and their arguments $\mathit{arg}_{j,k}$ are product-free set expressions themselves. As we see in §3.1.3 below, set expressions are more constrained than numerical expressions, in that the functions f_j must be affine.

It is well understood that the intersection of convex sets is convex, as is the direct product of convex sets; and that unions and set differences generally are not convex. What may not be clear is why **PS8**-**PS9** are considered "product-free" rules. The link becomes clear if we examine these rules in terms of indicator functions. Consider, for example, the problem

$$\begin{array}{ll} \text{minimize} & ax + by \\ \text{subject to} & (x, y) \in (S_1 \times S_1) \cup S_2 \end{array} \tag{3.8}$$

If $\phi_1 : \mathbb{R} \to \mathbb{R}$ and $\phi_2 : (\mathbb{R} \times \mathbb{R}) \to \mathbb{R}$ are convex indicator functions for the sets S_1 and S_2, respectively, then the problem can be reduced to

$$\text{minimize} \quad ax + by + (\phi_1(x) + \phi_2(y))\phi_2(x, y) \tag{3.9}$$

and the objective function now violates the product-free rule. What has occurred, of course, is that the union operation became a forbidden product.

3.1.3 The sign rules

Once the product-free conventions are established, the sum and scaling principles of convex analysis can be used to construct a simple set of sufficient conditions to establish whether or not expressions are convex, concave, or affine. These conditions form what we call the *sign rules*, so named because their purpose is to govern the signs of the quantities $c_1, \ldots, c_L$ in (3.3). We can concisely state the sign rules for numeric expressions in the following manner.

Sign rules for numeric expressions. Given a product-free expression, the following must be true of its reduced form (3.3):

SN1 If the expression is expected to be convex, then each term $c_j f_j(\ldots)$ must be convex; hence one of the following must be true:

- $f_j(\mathit{arg}_{j,1}, \mathit{arg}_{j,2}, \ldots, \mathit{arg}_{j,m_j})$ is affine;
- $f_j(\mathit{arg}_{j,1}, \mathit{arg}_{j,2}, \ldots, \mathit{arg}_{j,m_j})$ is convex and $c_j \geq 0$;
- $f_j(\mathit{arg}_{j,1}, \mathit{arg}_{j,2}, \ldots, \mathit{arg}_{j,m_j})$ is concave and $c_j \leq 0$.

SN2 If the expression is expected to be concave, then each term $c_j f_j(\dots)$ must be concave; hence one of the following must be true:

- $f_j(arg_{j,1}, arg_{j,2}, \dots, arg_{j,m_j})$ is affine;
- $f_j(arg_{j,1}, arg_{j,2}, \dots, arg_{j,m_j})$ is concave and $c_j \geq 0$;
- $f_j(arg_{j,1}, arg_{j,2}, \dots, arg_{j,m_j})$ is convex and $c_j \leq 0$.

SN3 If the expression is expected to be affine, then each function f_j must be affine, as must each of its arguments $arg_{j,1}, arg_{j,2}, \dots, arg_{j,m_j}$.

SN4 If the expression is expected to be constant, then it must be true that $L = 0$ and $b_1 = b_2 = \dots = b_n = 0$.

All function arguments must obey these rules as well, with their expected curvature dictated by the composition rules (§3.1.4).

For example, suppose that the expression (3.4) is expected to be convex, and that the atom library indicates that the function $f(\cdot)$ is convex, $g(\cdot,\cdot)$ is concave, and $h(\cdot,\cdot,\cdot)$ is convex. Then the sign rule dictates that

$$a f(x) \ \text{convex} \quad \Longrightarrow \quad a \geq 0 \tag{3.10}$$

$$(1/a) h(x, bg(y,z), c) \ \text{convex} \quad \Longrightarrow \quad 1/a \geq 0 \tag{3.11}$$

Function arguments must obey the sign rule as well, and their curvature is dictated by the composition rules discussed in the next section. So, for example, if the second argument of h is required to be convex, then

$$bg(y,z) \ \text{convex} \quad \Longrightarrow \quad b \leq 0 \tag{3.12}$$

It is the responsibility of the modeler to ensure that the values of the coefficients $c_1, \dots, c_L$ obey the sign rules; that is, that conditions such as those generated in (3.10)-(3.12) are satisfied. This can be accomplished by adding appropriate assertions to the model; see §3.3 for an example of this.

There is only one "sign rule" for set expressions:

The sign rule for set expressions. Given a product-free set expression, the following must be true of its reduced form (3.7):

SS1 Each function f_j, and any functions used in their arguments, must be affine.

Unlike the product-free rule for numerical expressions, functions involved in set expressions are *required* to be affine. To understand why this must be the case, one must understand how an expression of the form (3.7) is interpreted. A simple example should suffice; for the $L = 0$ case,

$$x \in a + \sum_{i=1}^{n} b_i S_i \quad \Longleftrightarrow \quad \exists\, (t_1, t_2, \dots, t_n) \in S_1 \times S_2 \times \dots \times S_n \quad x = a + \sum_{i=1}^{n} b_i t_i \tag{3.13}$$

When $L > 0$, similar substitutions are made recursively in the function arguments as well, producing a similar result: a series of simple set membership constraints of the form $t_k \in S_k$, and a single equality constraint. Thus in order to ensure that this implied equality constraint is convex, set expressions (specifically, those used in constraints) must reduce to affine combinations of sets. (Of course, set expressions used in assertions are not constrained in this manner.)

3.1.4 The composition rules

A basic principle of convex analysis is that the composition of a convex function with an affine mapping remains convex. In fact, under certain conditions, similar guarantees can be made for compositions with nonlinear mappings as well. The ruleset incorporates a number of these conditions, and we have called them the *composition rules*.

Designing the composition rules required a balance between simplicity and expressiveness. In [29], a relatively simple composition rule for convex functions is presented:

Lemma 3. *If $f : \mathbb{R} \to (\mathbb{R} \cup +\infty)$ is convex and nondecreasing and $g : \mathbb{R}^n \to (\mathbb{R} \cup +\infty)$ is convex, then $h = f \circ g$ is convex.*

So, for example, if $f(y) = e^y$ and $g(x) = x^2$, then the conditions of the lemma are satisfied, and $h(x) = f(g(x)) = e^{x^2}$ is convex. Similar composition rules are given for concave and/or nonincreasing functions as well:

- If $f : \mathbb{R} \to (\mathbb{R} \cup +\infty)$ is convex and nonincreasing and $g : \mathbb{R}^n \to (\mathbb{R} \cup -\infty)$ is concave, then $f \circ g$ is convex.
- If $f : \mathbb{R} \to (\mathbb{R} \cup -\infty)$ is concave and nondecreasing and $g : \mathbb{R}^n \to (\mathbb{R} \cup -\infty)$ is concave, then $f \circ g$ is concave.
- If $f : \mathbb{R} \to (\mathbb{R} \cup -\infty)$ is concave and nonincreasing and $g : \mathbb{R}^n \to (\mathbb{R} \cup +\infty)$ is convex, then $f \circ g$ is concave.

In addition, similar rules are described for functions with multiple arguments.

One way to interpret these composition rules is that they only allow nonlinear compositions that can be *separated* or *decomposed*. To explain, consider a nonlinear inequality $f(g(x)) \leq y$, where f is convex and nondecreasing and g is convex, thus satisfying the conditions of Lemma 3. Then it can be shown that

$$f(g(x)) \leq y \quad \iff \quad \exists z \; f(z) \leq y, \; g(x) \leq z \tag{3.14}$$

Similar decompositions can be constructed for the other composition rules as well. Decompositions serve as an important component in the conversion of DCPs into solvable form. Thus the composition rules guarantee that equivalency is reserved when these decompositions are performed.

The composition rules suggested by Lemma 3 and its related corollaries are a good start. But despite their apparent simplicity, they require a surprising amount of care to apply. In particular,

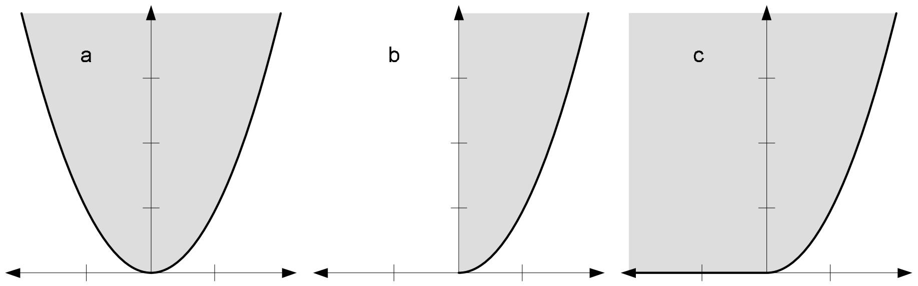

Figure 3.2: Three different "versions" of $f(y) = y^2$.

the use of extended-valued functions is a necessary part of the lemma and has a subtle impact. For example, consider the functions

$$f(y) = y^2, \quad g(x) = \|x\|_2 \tag{3.15}$$

Certainly, $h(x) = f(g(x)) = \|x\|_2^2$ is convex; but Lemma 3 would not predict this, because $f(y)$ is not monotonic. A sensible attempt to rectify the problem would be to restrict the domain of the f to the nonnegative orthant, where it is non-decreasing. Under the extended-valued valued conventions we have adopted, this produces the function

$$\bar{f}(y) = \begin{cases} y^2 & y \geq 0 \\ +\infty & y < 0 \end{cases} \tag{3.16}$$

But while $\tilde{f}$ is nondecreasing over its domain, it is *nonmonotonic* in the extended-valued sense, so the lemma does not apply. The only way to reconcile Lemma 3 with this example is to introduce yet another version of f that extends $\bar{f}$ in a nondecreasing fashion:

$$\bar{\bar{f}}(y) = \begin{cases} y^2 & y \geq 0 \\ 0 & y < 0 \end{cases} \tag{3.17}$$

Figure 3.2 provides a graph of each "version" of f. Forcing users of disciplined convex programming to consider such technical conditions seems an unnecessary complication, particularly when the goal is to *simplify* the construction of CPs.

To simplify the use of these composition rules, we begin by recognizing something that seems intuitively obvious: f need only be nondecreasing over the range of g. We can formalize this intuition as follows:

Lemma 4. *Let $f : \mathbb{R} \to (\mathbb{R} \cup +\infty)$ and $g : \mathbb{R}^n \to (\mathbb{R} \cup +\infty)$ be two convex functions. If f is nondecreasing over the range of g—i.e., the interval $g(\mathbb{R}^n)$—then $h = f \circ g$ is convex.*

Proof. Let $x_1, x_2 \in \mathbb{R}^n$ and let $\theta \in [0,1]$. Because g is convex,

$$g(\theta x_1 + (1-\theta)x_2) \leq \theta g(x_1) + (1-\theta)g(x_2) \leq \max\{g(x_1), g(x_2)\} \tag{3.18}$$

The right-hand inequality has been added to establish that

$$\begin{aligned}[g(\theta x_1 + (1-\theta)x_2), \theta g(x_1) + (1-\theta)g(x_2)] \subseteq \\ [g(\theta x_1 + (1-\theta)x_2), \max\{g(x_1), g(x_2)\}] \subseteq g(\mathbb{R}^n)\end{aligned} \tag{3.19}$$

f is therefore nondecreasing over this interval; so

$$\begin{aligned} f(g(\theta x_1 + (1-\theta)x_2)) &\leq f(\theta g(x_1) + (1-\theta)g(x_2)) && (f \text{ nondecreasing}) \\ &\leq \theta f(g(x_1)) + (1-\theta)f(g(x_2)) && (f \text{ convex}) \end{aligned} \tag{3.20}$$

establishing that $h = f \circ g$ is convex. □

This lemma does indeed predict the convexity of (3.15): f is nondecreasing over the interval $[0, +\infty)$, and $g(\mathbb{R}^n) = [0, +\infty)$, which coincide perfectly; hence, $f \circ g$ is convex.

And yet this revised lemma, while more inclusive, presents its own challenge. A critical goal for these composition rules is that adherence can be quickly, reliably, and automatically verified; see §3.3. The simple composition rules such as Lemma 3 plainly satisfy this condition; be can we do the same with the more complex rules? We claim that it is simpler than it may first appear. Note that our example function $f(x) = x^2$ is nondecreasing over a *half-line*; specifically, for all $x \in [0, +\infty)$. This is actually true for *any* nonmonotonic scalar function:

Lemma 5. *Let $f : \mathbb{R} \to \mathbb{R} \cup +\infty$ be a convex function that is nondecreasing over some interval $\bar{F} \subset \mathbb{R}$, $\operatorname{Int} \bar{F} \neq \emptyset$. Then it is, in fact, nonincreasing over the entire half-line $F = \bar{F} + [0, +\infty)$; i.e.,*

$$F = (F_{\min}, +\infty) \quad \textit{or} \quad F = [F_{\min}, +\infty). \tag{3.21}$$

Proof. If $\bar{F}$ already extends to $+\infty$, we are done. Otherwise, select any two points $x_1, x_2 \in \bar{F}$ and a third point $x_3 > x_2$. Then

$$\begin{aligned} &f(x_2) \leq \alpha f(x_1) + (1-\alpha)f(x_3), \quad \alpha \triangleq (x_2 - x_1)/(x_3 - x_1) \\ &\implies \quad f(x_2) \leq \alpha f(x_2) + (1-\alpha)f(x_3) \quad \implies \quad f(x_2) \leq f(x_3). \end{aligned} \tag{3.22}$$

So $f(x_3) \geq f(x_2)$ for all $x_3 > x_2$. Now consider another point $x_4 > x_3$; then

$$\begin{aligned} &f(x_3) \leq \bar{\alpha} f(x_2) + (1-\bar{\alpha})f(x_4), \quad \bar{\alpha} \triangleq (x_3 - x_2)/(x_4 - x_2) \\ &\implies \quad f(x_3) \leq \bar{\alpha} f(x_3) + (1-\bar{\alpha})f(x_4) \quad \implies \quad f(x_4) \leq f(x_3) \end{aligned} \tag{3.23}$$

So $f(x_4) \geq f(x_3)$ for all $x_4 > x_3 > x_2$; that is, f is nondecreasing for all $x > x_2 \in \bar{F}$. □

In other words, *any* scalar convex function that is nondecreasing between two points is so over an entire half-line. So determining whether f is nondecreasing over $g(\mathbf{dom}\, g)$ reduces to a single comparison between $F_{\min}$ and $\inf_x g(x)$. For concave or nonincreasing functions, similar intervals can be constructed:

- f convex, nonincreasing: $F = (-\infty, F_{\max}]$ or $F = (-\infty, F_{\max})$
- f concave, nondecreasing: $F = (-\infty, F_{\max}]$ or $F = (-\infty, F_{\max})$
- f concave, nonincreasing: $F = [F_{\min}, +\infty)$ or $F = (F_{\min}, +\infty)$

As we show in §3.2, it is straightforward to include such intervals in the atom library, so that they are readily available to verify compositions.

The task of determining $\inf_x g(x)$ or $\sup_x g(x)$ remains. In §3.2, we show that function ranges are *included* in the atom library for just this purpose, alongside information about their curvature and monotonicity properties. But often the inner expression $g(x)$ is not a simple function call, but an expression reducible to the form (3.3):

$$g(x) = \inf_x a + b^T x + \sum_{j=1}^{L} c_j f_j(\dots) \tag{3.24}$$

We propose the following heuristic in such cases. Let $X_i \subseteq \mathbb{R}$, $i = 1, 2, \dots, n$, be simple interval bounds on the variables, retrieved from any simple bounds present in the model. In addition, let $F_j = f_j(\mathbb{R}) \subseteq \mathbb{R}$, $j = 1, 2, \dots, L$, be the range bounds retrieved from the atom library. Then

$$g(\mathbb{R}) \subseteq a + \sum_{i=1}^{n} b_i X_i + \sum_{j=1}^{L} c_j F_j \tag{3.25}$$

So (3.25) provides a conservative bound on $\inf_x g(x)$ or $\sup_x g(x)$, as needed. In practice, this heuristic proves sufficient to support the composition rules in nearly all circumstances. In those exceptional circumstances where the bound is too conservative, and the heuristic fails to detect a valid composition, a model may have to be rewritten—say, by manually performing the decomposition (3.14) above. It is a small price to pay for added expressiveness in the vast majority of cases.

Generalizing these composition rules to functions with multiple arguments is straightforward, but requires a bit of technical care. The result is as follows:

The composition rules. Consider a numerical expression of the form

$$f(\mathit{arg}_1, \mathit{arg}_2, \dots, \mathit{arg}_m) \tag{3.26}$$

where f is a function from the atom library. For each argument arg_k, construct a bound $G_k \subseteq \mathbb{R}$ on the range using the heuristic described above, so that

$$(\mathit{arg}_1, \mathit{arg}_2, \dots, \mathit{arg}_m) \in G = G_1 \times G_2 \times \cdots \times G_m \tag{3.27}$$

Given these definitions, (3.26) must satisfy exactly one of the following rules:

C1-C3 If the expression is expected to be convex, then f must be affine or convex, and one of the following must be true for each $k = 1, \dots, m$:

- **C1** f is nondecreasing in argument k over G, and arg_k is convex; or
- **C2** f is nonincreasing in argument k over G, and arg_k is concave; or
- **C3** arg_k is affine.

C4-C6 If the expression is expected to be concave, then f must be affine or concave, and one of the following must be true for each $k = 1, \dots, m$:

- **C4** f is nondecreasing in argument k over G, and arg_k is concave; or
- **C5** f is nonincreasing in argument k over G, and arg_k is convex; or
- **C6** arg_k is affine.

C7 If the expression is expected to be affine, then f must be affine, and each arg_k is affine for all $k = 1, \dots, m$.

3.2 The atom library

The second component of disciplined convex programming is the *atom library.* As a concept, the atom library is relatively simple: an extensible list of functions and sets whose properties of curvature/shape, monotonicity, and range are known. The description of the DCP ruleset in §3.1 shows just how this information is utilized.

As a tangible entity, the atom library requires a bit more explanation. In `cvx`, the library is a collection of text files containing descriptions of functions and sets. Each entry is divided into two sections: the *declaration* and the *implementation.* The declaration is divided further into two components:

- the *prototype*: the name of the function or set and the and structure of its inputs.
- the *attribute list*: a list of descriptive statements concerning the curvature, monotonicity, and range of the function; or the shape of the set.

The *implementation* is a computer-friendly description of the function or set that enables it to be used in numerical solution algorithms. What is important to note here is that the implementation section is *not used* to determine whether or not a particular problem is a DCP. Instead, it comes into play only *after* a DCP has been verified, when one wishes to compute a numerical solution. For this reason, we postpone the description of the implementation until §3.5.

3.2.1 The prototype

A function prototype models the usage syntax for that function, and in the process describes the number and dimensions of its arguments; *e.g.*,

`function sq( x );` $\qquad f(x) = x^2 \qquad (3.28)$

`function max( x, y );` $\qquad g(x,y) = \max\{x,y\} \qquad (3.29)$

Some functions are parameterized; *e.g.*, $h_p(x) = \|x\|_p$. In `cvx`, parameters are included in the argument list along with the arguments; *e.g.*,

`function norm_p( p, x[n] ) if p >= 1;` $\qquad h_p(x) = \|x\|_p \quad (p \geq 1) \qquad (3.30)$

Parameters are then distinguished from variables through the curvature attribute; see §3.2.2 below.

The `norm_p` example also illustrates the ability to place *preconditions* on a function's parameters using an `if` construct. To use a function with preconditions, they must somehow enforced—perhaps by using one or more assertions. For example, `norm_p( 2.5, x )` would be verified as valid; but if `b` is a parameter, `norm_p( b, x )` would not be, unless the value of `b` could somehow be guaranteed to be greater than 1; for example, unless an assertion like `b > 1` were provided in the model.

Most set prototypes look identical to those of functions:

`set integers( x );` $\qquad A = \mathbb{Z} = \{\dots,-2,-1,0,1,2,\dots\} \qquad (3.31)$

`set simplex( x[n] );` $\qquad B = \{\, x \in \mathbb{R}^n \mid x \geq 0,\ \sum_i x_i = 1 \,\} \qquad (3.32)$

`set less_than( x, y );` $\qquad C = \{\, (x,y) \in \mathbb{R} \times \mathbb{R} \mid x < y \,\} \qquad (3.33)$

Unlike functions, the actual *usage* of a set differs from its prototype—the arguments are in fact the components of the set, and therefore appear to the left of a set membership expression: *e.g.*,

`x in integers;` $\qquad x \in A \qquad (3.34)$

`y in simplex;` $\qquad y \in B \qquad (3.35)$

`(x,y) in less_than;` $\qquad (x,y) \in C \qquad (3.36)$

For parameterized sets, there is yet another difference: the parameters are supplied in a *separate* parameter list, preceding the argument list: *e.g.*,

`set ball_p( p )( x[n] ) if p >= 1;` $\qquad B_p(x) = \{\, x \in \mathbb{R}^n \mid \|x\|_p \leq 1 \,\} \quad (p \geq 1) \qquad (3.37)$

This parameter list remains on the right-hand side of the constraint along with the name of the set:

`z in ball_p( q );` $\qquad z \in B_q \qquad (3.38)$

3.2.2 Attributes

As we saw in §3.1, the DCP ruleset depends upon one or more pieces of information about each function and set utilized in a DCP. For sets, it utilizes just one item:

- *shape*: specifically, whether or not the set is convex.

For functions, a bit more information is used:

- *curvature*: whether the function is convex, concave, affine, or otherwise.
- *monotonicity*: whether the functions are nonincreasing or nondecreasing; and over what subsets of their domain they are so.
- *range*: the minimum of convex functions and the maximum of concave functions.

The `cvx` framework allows this information to be provided through the use of *attributes*: simple text tags that allow each of the above properties to be identified as appropriate.

Shape

For sets, only one attribute is recognized: `convex`. A set is either convex, in which case this attribute is applied, or it is not. Given the above four examples, only `integers` is not convex:

```
set integers( x );                                (3.39)
set simplex( x[n] ) convex;                       (3.40)
set less_than( x, y ) convex;                     (3.41)
set ball_p( p )( x[n] ) convex if p >= 1;         (3.42)
```

Sets that are not convex are obviously of primary interest for DCP, but non-convex sets may be genuinely useful, for example, for restricting the values of parameters to realistic values.

Curvature

Functions can be declared as `convex`, `concave`, or `affine`, or none of the above. Clearly, this last option is the least useful; but such functions can be used in constant expressions or assertions. No more than one curvature keyword can be supplied. For example:

`function max( x, y ) convex;`	$g(x, y) = \max\{x, y\}$	(3.43)
`function min( x, y ) concave;`	$f(x, y) = \min\{x, y\}$	(3.44)
`function plus( x, y ) affine;`	$p(x, y) = x + y$	(3.45)
`function sin( x );`	$g(x) = \sin x$	(3.46)

By default, a function declared as convex, concave, or affine is assumed to be *jointly* so over all of its arguments. It is possible to specify that it is so only over a subset of its arguments by listing those arguments after the curvature keyword; for example, the function

```
function norm_p( p, x[n] ) convex( x ) if p >= 1;
```

$$h_p(x) = \|x\|_p \quad (p \geq 1) \tag{3.47}$$

is convex only in its second argument `x`. This convention allows parameterized functions to be declared: arguments omitted from the list are the parameters of the function and must be constant.

Disciplined convex programming allows only functions that are *globally* convex or concave to be specified as such. For example, the simple inverse function

$$f : \mathbb{R} \to \mathbb{R}, \quad f(x) \triangleq 1/x \qquad (x \neq 0) \tag{3.48}$$

is neither convex nor concave, and so cannot be used to construct a DCP. However, we commonly think of f as convex *if x is known to be positive.* In disciplined convex programming, this understanding must be realized by defining a *different function*

$$f_{\text{cvx}} : \mathbb{R} \to \mathbb{R}, \quad f(x) \triangleq \begin{cases} 1/x & x > 0 \\ +\infty & x \leq 0 \end{cases} \tag{3.49}$$

which is globally convex, and can therefore be used in DCPs. Similarly, the power function

$$g : \mathbb{R}^2 \to \mathbb{R}, \quad g(x,y) \triangleq x^y \qquad \text{(when defined)} \tag{3.50}$$

is convex or concave on certain subsets of $\mathbb{R}^2$, such as:

- convex for $x \in [0, \infty)$ and fixed $y \in [1, \infty)$.
- concave for $x \in [0, \infty)$ and fixed $y \in (0, 1]$;
- convex for fixed $x \in (0, \infty)$ and $y \in \mathbb{R}$

In order to introduce nonlinearities such as $x^{2.5}$ or $x^{0.25}$ into a DCP, there must be appropriate definitions of these "restricted" versions of the power function:

$$f_y : \mathbb{R} \to \mathbb{R}, \quad f_y(x) = \begin{cases} x^y & x \geq 0 \\ +\infty & x < 0 \end{cases} \quad (y \geq 1) \tag{3.51}$$

$$g_y : \mathbb{R} \to \mathbb{R}, \quad g_y(x) = \begin{cases} x^y & x \geq 0 \\ -\infty & x < 0 \end{cases} \quad (0 < y < 1) \tag{3.52}$$

$$h_y : \mathbb{R} \to \mathbb{R}, \quad h_y(x) = y^x \quad (y > 0) \tag{3.53}$$

Thus the disciplined convex programming approach forces the user to consider convexity more carefully. We consider this added rigor an advantage, not a liability.

Monotonicity

The monotonicity of a function with respect to its arguments proves to be a key property exploited by the ruleset. For this reason, the `cvx` atom library provides the keywords `increasing`, `nondecreasing`, `nonincreasing`, or `decreasing` in each of its arguments. Each argument can be given a separate declaration:

```
function exp( x ) convex increasing;
```

$$f(x) = e^x \tag{3.54}$$

The ruleset does not treat strict monotonicity specially; so, for example, the attributes `increasing` and `nondecreasing` are effectively synonymous, as are `decreasing` and `nonincreasing`.

There is one somewhat technical but critical detail that must be adhered to when declaring a function to be monotonic. Specifically, monotonicity must be judged in the *extended-valued* sense. For example, given $p \geq 1$, the function

```
function pow_p( p, x ) convex( x ) if p >= 1;
```

$$f_p(x) = \begin{cases} x^p & x \geq 0 \\ +\infty & x < 0 \end{cases} \tag{3.55}$$

is increasing over its (real-valued) domain. However, in the extended-valued sense, the function is *nonmonotonic*, so f_p cannot be declared as globally monotonic.

As suggested in §3.1.4, the `cvx` atom library allows *conditions* to be placed on monotonicity. So, for example, $\tilde{f}_p(x)$ is, of course, nondecreasing over $x \in [0, +\infty)$, suggesting the following declaration:

```
function pow_p( p, x ) convex( x ) if p >= 1, increasing( x ) if x >= 0;    (3.56)
```

Multiple declarations are possible: for example, the function $f(x) = x^2$ is both nonincreasing over $x \in (-\infty, 0]$ and nondecreasing over $x \in [0, +\infty)$:

```
function sq( x ) convex, decreasing if x <= 0, increasing if x >= 0;    (3.57)
```

Each argument of a function with multiple inputs can be given a separate, independent monotonicity declaration. For example, $f(x, y) = x - y$ is increasing in x and decreasing in y:

```
function minus( x, y ) affine increasing( x ) decreasing( y );    (3.58)
```

Range

A function definition can include a declaration of its its *range*, using a simple inequality providing a lower or upper bound for the function. For example,

```
function exp( x ) convex, increasing, >= 0;
```

$$f(x) = e^x \tag{3.59}$$

```
minimize    c x;
subject to  exp( y ) <= log( a sqrt( x ) + b );
            a x + b y = d;
variables   x, y;
parameters  a, b, c, d;
function    exp( x )  convex  increasing >= 0;
function    sqrt( x ) concave nondecreasing;
function    log( x )  concave increasing;
```

Figure 3.3: The `cvx` specification for (3.60).

As with the monotonicity operations, the range must be specified in the extended-valued sense, so it will inevitably be one-sided: that is, all convex functions are unbounded above, and all concave functions are unbounded below.

3.3 Verification

In order to solve a problem as a DCP, one must first establish that it is indeed a valid DCP—that is, that it involves only functions and sets present in the atom library, and combines them in a manner compliant with the DCP ruleset. A proof of validity is necessarily hierarchical in nature, reflecting the structure of the problem and its expressions. To illustrate the process, consider the simple convex optimization problem

$$\begin{array}{rl} \text{minimize} & cx \\ \text{subject to} & \exp(y) \leq \log(a\sqrt{x}+b) \\ & ax+by=d \end{array} \tag{3.60}$$

where a, b, c, d are parameters, and x, y are variables. A `cvx` version of this model is given in Figure 3.3. Note in particular the explicit declarations of the three atoms `exp`, `log`, and `sqrt`. Usually these declarations will reside in an external file, but we include them here to emphasize that every atom used in a model must be accounted for in the atom library.

It is helpful to divide the proof into two stages. The first stage verifies that each of the expressions involved is product-free. Below is a textual description of this stage. The lines are indented in order to represent the hierarchy present in the proof. In addition, each line line includes the number of the rule employed.

cx is product-free, because (**PN7**)
    c is a constant expression(**T8**)
    x is product-free (**PN1**)
$\exp(y)$ is product-free, because (**PN3**)
    y is product-free (**PN2**)
$\log(a\sqrt{x}+b)$ is product-free, because (**PN3**)

$a\sqrt{x}+b$ is product-free, because (**PN4**)
$a\sqrt{x}$ is product-free, because (**PN7**)
a is a constant expression (**T8**)
$\sqrt{x}$ is product-free, because (**PN3**)
x is product-free (**PN2**)
b is product-free (**PN2**,**T8**)
$ax+by$ is product-free, because (**PN4**)
ax is product-free, because (**PN7**)
a is a constant expression (**T8**)
x is product-free (**PN1**)
by is product-free, because (**PN7**)
b is a constant expression (**T8**)
y is product-free (**PN1**)
d is product-free (**PN2**)

The second stage proceeds by verifying the top-level, sign, and composition rules in a similarly hierarchical fashion:

The minimization problem is valid if $a \geq 0$, because (**T1**)
The objective function is valid, because (**T1**)
cx is convex (**SN1**)
The first constraint is valid if $a \geq 0$, because (**T1**)
$\exp(y) \leq \log(a\sqrt{x}+b)$ is convex if $a \geq 0$, because (**T5**)
$\exp(y)$ is convex, because (**SN1**)
$\exp(y)$ is convex, because (**C1**)
$\exp(\cdot)$ is convex and nondecreasing (atom library)
y is convex (**SN1**)
$\log(a\sqrt{x}+b)$ is concave if $a \geq 0$, because (**SN2**)
$\log(a\sqrt{x}+b)$ is concave if $a \geq 0$, because (**C4**)
$\log(\cdot)$ is concave and nondecreasing (atom library)
$a\sqrt{x}+b$ is concave if $a \geq 0$, because (**SN2**)
$\sqrt{x}$ is concave, because (**C4**)
$\sqrt{\cdot}$ is concave and nondecreasing (atom library)
x is concave (**SN2**)
The second constraint is valid, because (**T1**)
$ax+by=d$ is convex, because (**T4**)
$ax+by$ is affine (**SN3**)
d is affine (**SN3**)

It can be quite instructive to view this proof of validity graphically. Figure 3.4 presents an *expression tree* of the problem (3.60), annotated with the relevant rules verified at each position in the tree.

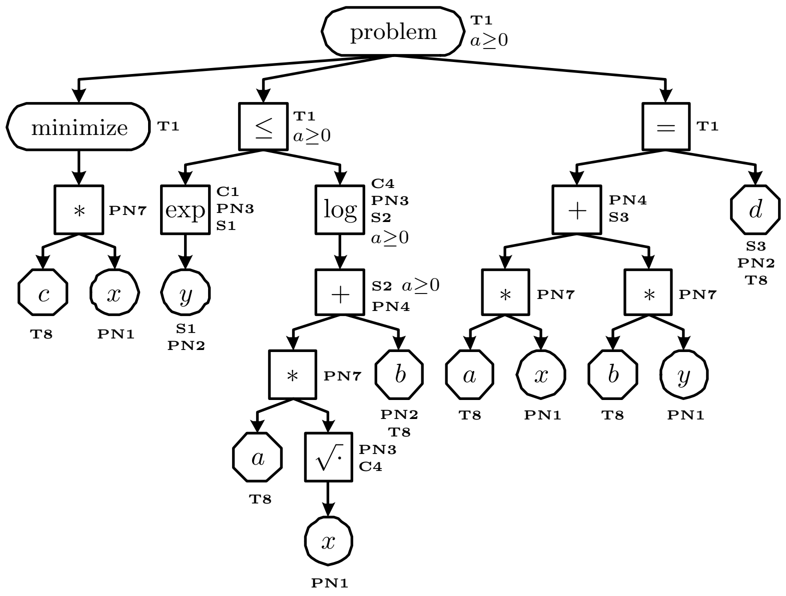

Figure 3.4: An expression tree for (3.60), annotated with applicable convexity rules.

The verification process is guaranteed to yield one of three conclusions:

- *valid*: the rules are fully satisfied.
- *conditionally valid*: the rules will be fully satisfied if one or more additional preconditions on the parameters are satisfied.
- *invalid*: one or more of the convexity rules has been violated.

In this case, a conclusion of *conditionally valid* has been reached: the analysis has revealed that an additional condition $a \geq 0$ must satisfied. If this precondition were somehow assured, then the proof would have conclusively determined that the problem is a valid DCP. One simple way to accomplish is to add the assertion `a >= 0` to the problem. If, on the other hand, we were to do the opposite and add an assertion `a < 0`, the sign rule **SN2** would be violated; in fact, the expression $a\sqrt{x}+b$ would be verifiably convex.

The task of verifying DCPs comprises yet another approach to the challenge of automatic convexity verification described in §1.3. Like the methods used to verify LPs and SDPs, a certain amount of structure is assumed via the convexity rules that enables the verification process to proceed in a reliable and deterministic fashion. However, unlike these more limited methods, disciplined convex programming maintains generality by allowing new functions and sets to be added to the atom library. Thus disciplined convex programming provides a sort of *knowledgebase* environment for convex programming, in which human-supplied information about functions and sets is used to expand the body of problems that can be recognized as convex.

```
maximize    - x1 log( x1 ) - x2 log( x2 ) - x3 log( x3 ) - x4 log( x4 );
subject to  a1 x1 + a2 x2 + a3 x3 + a4 x4 = b;
               x1 +    x2 +    x3 +    x4 = 1;
parameters  a1, a2, a3, a4, b;
variables   x1 >= 0, x2 >= 0, x3 >= 0, x4 >= 0;
function    log( x ) concave nondecreasing;
```

Figure 3.5: An entropy maximization problem.

```
maximize    log( exp( a11 x1 + x12 x2 + a13 x3 - b1 ) +
                 exp( a21 x1 + a22 x2 + a23 x3 - b2 ) );
subject to  log( exp( a31 x1 + a32 x2 + a33 x3 - b3 ) +
                 exp( a41 x1 + a42 x2 + a43 x3 - b4 ) <= 0;
            a51 x1 + a52 x2 + a53 x3 = b5;
parameters  a11, a12, a13, b1, a21, a22, a23, b2,a31, a32, a33, b3,
            a41, a42, a43, b4, a51, a52, a53, b5;
variables   x1, x2, x3;
function    log( x ) concave nondecreasing;
function    exp( x ) convex nondecreasing >= 0;
```

Figure 3.6: A geometric program in convex form.

3.4 Navigating the ruleset

Adherence to the DCP ruleset is sufficient but not necessary to ensure convexity. It is possible to construct mathematical programs that are indeed convex, but which fail to be DCPs because one or more of the expressions involved violate the ruleset. In fact, it is quite simple to do so: for example, consider the entropy maximization problem

$$\begin{array}{ll} \text{maximize} & -\sum_{j=1}^n x_j \log x_j \\ \text{subject to} & Ax = b \\ & \vec{1}^T x = 1 \\ & x \geq 0 \end{array} \tag{3.61}$$

where $x \in \mathbb{R}^n$ is the problem variable and $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$ are parameters; and $\log(\cdot)$ is defined in the atom library. A `cvx` version of this problem, with $(m,n) = (2,4)$, is given in Figure 3.5. The expression $x_j \log x_j$ violates the product-free rule **PS6**—and as a result, (3.61) is not a DCP, even though it is a well-known CP. Alternatively, consider a geometric program in convex form,

$$\begin{array}{lll} \text{minimize} & \log \sum_{i=1}^{m_0} \exp(a_{0i}^T x + b_{0i}) & \\ \text{subject to} & \log \sum_{i=1}^{m_i} \exp(a_{ki}^T x + b_{ki}) \leq 0 & k = 1,2,\ldots,M \\ & A^{(M+1)}x + b^{(M+1)} = 0 & \end{array} \tag{3.62}$$

where $x \in \mathbb{R}^n$ is the problem variable, the quantities

$$A^{(k)} = [a_{k1} \quad a_{k2} \quad \ldots \quad a_{km_k}]^T \in \mathbb{R}^{m_k\times n} \quad b^{(k)} = [b_{k1} \quad b_{k2} \quad \ldots \quad b_{km_k}]^T \in \mathbb{R}^{m_k} \tag{3.63}$$

```
maximize    entr( x1, x2, x3, x4 );
subject to  a1 * x1 + a2 * x2 + a3 * x3 + a4 * x4 = b;
                 x1 +      x2 +      x3 +      x4 = 1;
parameters  a1, a2, a3, a4, b;
variables   x1 >= 0, x2 >= 0, x3 >= 0, x4 >= 0;
function    entr( ... ) concave;
```

Figure 3.7: An entropy maximization problem, expressed as a DCP.

for $k = 1, 2, \ldots, M+1$ are parameters; and both $\log(\cdot)$ and $\exp(\cdot)$ are defined in the atom library. An example `cvx` model of this problem is given in Figure 3.6. This problem satisfies the product-free rules, but the objective function and inequality constraints fail either the sign rule **SN1** or the composition rule **C4**, depending on how you verify them. But of course, (3.62) is a CP.

These violations do not mean that the problems cannot be solved as disciplined convex programs, just that they must be rewritten in a compliant manner. The simplest way to do so is to add new functions to the atom library that encapsulate the offending nonlinearities:

$$f_{\text{entr}} : \mathbb{R}^n \to (\mathbb{R} \cup -\infty), \quad f_{\text{entr}}(x) \triangleq \sum_{j=1}^n \bar{f}(x_j), \quad \bar{f}(x_j) \triangleq \begin{cases} -x_j \log x_j & x_j > 0 \\ 0 & x_j = 0 \\ -\infty & x_j < 0 \end{cases} \tag{3.64}$$

$$f_{\text{lse}} : \mathbb{R}^n \to \mathbb{R}, \quad f_{\text{lse}}(y) = \log \sum_{j=1}^n e^{y_j}. \tag{3.65}$$

Using these functions, we can rewrite both problems as DCPs; (3.61) as

$$\begin{array}{ll} \text{maximize} & f_{\text{entr}}(x) \\ \text{subject to} & Ax = b \\ & \vec{1}^T x = 1 \\ & x \geq 0 \end{array} \tag{3.66}$$

and the GP (3.62) as

$$\begin{array}{ll} \text{minimize} & f_{\text{lse}}(A^{(0)}x + b^{(0)}) \\ \text{subject to} & f_{\text{lse}}(A^{(k)}x + b^{(k)}) \leq 0, \quad i = 1, 2, \ldots, M \\ & A^{(M+1)}x + b^{(M+1)} = 0. \end{array} \tag{3.67}$$

`cvx` versions of these problems are given in Figures 3.7 and 3.8, respectively.

The ability to extend the atom library as needed has the potential to be taken to an inelegant extreme. For example, consider the problem

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_k(x) \leq 0, \quad k = 1, 2, \ldots, m \end{array} \tag{3.68}$$

```
maximize    lse( a11 * x1 + x12 * x2 + a13 * x3 - b1,
                 a21 * x1 + a22 * x2 + a23 * x3 - b2 );
subject to  lse( a31 * x1 + a32 * x2 + a33 * x3 - b3,
                 a41 * x1 + a42 * x2 + a43 * x3 - b4 ) <= 0;
            a51 * x1 + a52 * x2 + a53 * x3 = b5;
parameters  a11, a12, a13, b1, a21, a22, a23, b2,a31, a32, a33, b3,
            a41, a42, a43, b4, a51, a52, a53, b5;
variables   x1, x2, x3;
function    lse( ... ) convex nondecreasing;
```

Figure 3.8: A geometric program in convex form, expressed as a DCP.

where the functions $f_0, f_1, \dots, f_m$ are convex. One way to cast this problem as a DCP would simply be to add all $m+1$ of the functions to the atom library. The convexity rules would then be satisfied rather trivially; and yet this would probably require *more*, not *less*, effort than a more traditional NLP modeling method. In practice, however, the functions f_k are rarely monolithic, opaque objects. Rather, they are constructed from components such as affine forms, norms, and other known functions, combined in ways consistent with the principles of convex analysis captured by the DCP ruleset. It is *those* functions that are suitable for addition into the atom library.

Once an atom is defined and implemented, it can be freely reused across many DCPs. The atoms can be shared with other users as well. The effort involved in adding a new function to the atom library, then, is significantly amortized. A collaborative hierarchy is naturally suggested, wherein more advanced users can create new atoms for application-specific purposes, while novice users can employ them in their models without regard for how they were constructed.

We argue, therefore, that (3.61) and (3.62) are ideal examples of the kinds of problems that disciplined convex programming is intended to support, so that the DCP ruleset poses little practical burden in these cases. While it is true that the term $-x_j \log x_j$ violates the product-free rules, someone interested in entropy maximization does not consider this expression as a product of nonlinearities but rather as a single, encapsulated nonlinearity—as represented by the function f_{entr}. In a similar manner, those studying geometric programming treat the function f_{lse} as a monolithic convex function; it is irrelevant that it happens to be the composition of a concave function and a convex function. Thus the addition of these functions to the atom library coincides with the intuitive understanding of the problems that employ them.

Still, the purity of the convexity rules prevents even the use of obviously convex quadratic forms such as $x^2 + 2xy + y^2$ in a model. It could be argued that this is impractically restrictive, since quadratic forms are so common. Indeed, an extension of the convexity rules to include quadratic forms is worthy of consideration. But in many cases, a generic quadratic form may in fact represent a quantity with more structure or meaning. For example, traditionally, the square of a Euclidean norm $\|Ax+b\|_2^2$ might be converted to a quadratic form

$$\|Ax+b\|_2 = x^T P x + q^T x + r, \quad P \triangleq A^T A, \quad q \triangleq A^T b, \quad r \triangleq b^T b \tag{3.69}$$

But within a DCP, this term can instead be expressed as a composition:

$$\|Ax+b\|_2 = f(g(Ax+b)), \quad f(y) \triangleq y^2, \quad g(z) \triangleq \|z\|_2 \tag{3.70}$$

In disciplined convex programming, there is no natural bias against (3.70), so it should be preferred over the converted form (3.69) simply because it reflects the original intent of the problem. So we argue that support for generic quadratic forms would be at least somewhat less useful than in a more traditional modeling framework. Furthermore, we can easily support quadratic forms with the judicious addition of functions to the atom library, such as the function $f(y) = y^2$ above, or a more complex quadratic form such as

$$f_Q : \mathbb{R}^n \to \mathbb{R}, \quad f_Q(x) = x^T Q x \qquad (Q \succeq 0). \tag{3.71}$$

Thus even support for quadratic forms is a matter of convenience, not necessity.

3.5 Implementing atoms

In §1.2.2, we enumerated a variety of numerical methods for solving CPs. Many of those methods can be readily adapted to solve DCPs, as we shall see in Chapter 4. For now, it is sufficient to know that each of these methods performs certain computations involving each of the atoms (functions and sets) they encounter in problems they solve. As we briefly introduced in §3.2, the purpose of the *implementation* of an atom is to provide the means to perform these calculations. In this section, we provide a more detailed description of atom implementations, once again using the `cvx` modeling language to illustrate our examples.

Disciplined convex programming and the `cvx` modeling framework distinguish between two different types of implementations:

- a *simple* implementation, which provides traditional calculations such as derivatives, subgradients and supergradients for functions; and indicator functions, barrier functions, and cutting planes for sets; and
- a *graph* implementation, in which the function or set is defined *as the solution to another DCP.*

The computations supported by simple implementations should be quite familiar to anyone who studies the numerical solution of optimization problems. To the best of our knowledge, however, the concept of graph implementations is new, and proves to be an important part of the power and expressiveness of disciplined convex programming.

In `cvx`, an implementation is surrounded by curly braces, and consists of a list of key/value pairs with the syntax *key* `:=` *value*. See Figures 3.9 and 3.10 for examples. It is also possible for an implementation to be constructed in a lower-level language like C, but we do not consider that feature here.

3.5.1 Simple implementations

Any continuous function can have a simple implementation. Simple function implementations use the following *key* `:=` *value* entries:

- `value`: the value of the function.
- `domain_point`: a point on the interior of the domain of the function; the origin is assumed if this is omitted.
- For differentiable functions:
  - `gradient`: the first derivative.
  - `Hessian` (if twice differentiable): the second derivative.
- For nondifferentiable functions:
  - `subgradient` (if convex): a subgradient of a function f at point $x \in \mathbf{dom}\, f$ is any vector $v \in \mathbb{R}^n$ satisfying
  $$f(y) \geq f(x) + v^T(y - x) \quad \forall y \in \mathbb{R}^n \tag{3.72}$$
  - `supergradient` (if concave): a supergradient of a function g at point $x \in \mathbf{dom}\, g$ is any vector $v \in \mathbb{R}^n$ satisfying
  $$g(y) \leq g(x) + v^T(y - x) \quad \forall y \in \mathbb{R}^n \tag{3.73}$$

It is not difficult to see how different algorithms might utilize this information. Almost every method would use `value` and `domain_point`, for example. A smooth CP method would depend on the entries `gradient` and `Hessian` to calculate Newton search directions. A localization method would use the entries `gradient`, `subgradient`, and `supergradient` to compute cutting planes.

Any set with a non-empty interior can also have a simple implementation. Simple set implementations use the following *key* `:=` *value* entries:

- `interior_point`: a point on the interior of the set.
- `indicator`: an expression returning 0 for points inside the set, and $+\infty$ for points outside it.
- At least one, but ideally both, of the following:
  - `barrier`: a reference to a convex, twice differentiable *barrier function* for the set, declared separately as a function atom with a direct implementation.
  - `oracle`: a cutting plane oracle for the set. Given a set $S \subset \mathbb{R}^n$, the cutting plane oracle accepts as input a point $x \in \mathbb{R}^n$; and, if $x \notin S$, returns a separating hyperplane; that is, a pair $(a, b) \in \mathbb{R}^n \times \mathbb{R}$ satisfying
  $$a^T x \leq b, \quad S \subseteq \left\{\, y \;\middle|\; a^T y \geq b \,\right\} \tag{3.74}$$

```
function min( x, y ) concave, nondecreasing {
    value := x < y ? x : y;
    supergradient := x < y ? ( 1, 0 ) : ( 0, 1 );
}
set pos( x ) convex {
    interior_point := 1.0;
    indicator := x < 0 ? +Inf : 0;
    oracle := x < 0 ? ( 1, 0 ) : ( 0, 0 );
    barrier := neglog( x );
}
function neglog( x ) convex {
    domain_point := 1.0;
    value := x <= 0 ? +Inf : - log( x );
    gradient := - 1 / x;
    Hessian := 1 / x^2;
}
```

Figure 3.9: Simple implementations.

If $x \in S$, then the oracle returns $(a, b) = (0, 0)$.

Figure 3.9 presents several examples of simple implementations. Again, we do not wish to document the `cvx` syntax here, only illustrate the feasibility of this approach. Note that the set `pos` has both a barrier function and a cutting plane generator, allowing it to be used in both types of algorithms.

3.5.2 Graph implementations

A fundamental principle in convex analysis is the very close relationship between convex and concave functions and convex sets. A function $f : \mathbb{R}^n \to (\mathbb{R} \cup +\infty)$ is convex if and only if its *epigraph*

$$F = \operatorname{epi} f = \{\, (x, y) \in \mathbb{R}^n \times \mathbb{R} \;\mid\; f(x) \leq y \,\} \tag{3.75}$$

is a convex set. Likewise, a function $g : \mathbb{R}^n \to \mathbb{R} \cup +\infty$ is concave if and only if its *hypograph*

$$G = \operatorname{hypo} g = \{\, (x, y) \in \mathbb{R}^n \times \mathbb{R} \;\mid\; g(x) \geq y \,\} \tag{3.76}$$

is a convex set. These relationships can be expressed in reverse fashion as well:

$$f(x) = \inf \{\, y \;\mid\; (x, y) \in F \,\} \tag{3.77}$$

$$g(x) = \sup \{\, y \;\mid\; (x, y) \in G \,\} \tag{3.78}$$

A *graph implementation* of a function is effectively a representation of the epigraph or hypograph of a function, as appropriate, as a disciplined convex feasibility problem (top-level rule **T3**). The `cvx` framework supports this approach by allowing epigraphs or hypographs to be represented in *key* `:=` *value* pairs `epigraph` and `hypograph`, respectively.

```
function abs( x ) convex, >= 0 {
    value := x < 0 ? -x : x;
    epigraph := { -abs <= x <= +abs; }
}
function min( x, y ) concave, nondecreasing {
    value := x < y ? x : y;
    supergradient := x < y ? ( 1, 0 ) : ( 0, 1 );
    hypograph := { min <= x; min <= y; }
}
function entropy( x ) concave {
    value := x < 0 ? -Inf : x = 0 ? 0 : - x log( x );
    hypograph := { ( x, y ) in hypo_entropy; }
}
set simplex( x[n] ) convex {
    constraints := { sum( x ) = 1; x >= 0; }
}
```

Figure 3.10: Graph implementations.

For a simple example, consider the absolute value function $f(x) = |x|$, the epigraph of which is

$$\operatorname{epi} f = \{\,(x,y) \;|\; |x| \leq y\,\} = \{\,(x,y) \;|\; -y \leq x \leq y\,\} \tag{3.79}$$

In Figure 3.10, we show how this epigraph is represented in `cvx`. Notice that the name of the function `abs` is used to represent the epigraph variable.

The primary benefit of graph implementations is that they provide an elegant means to define nondifferentiable functions. The absolute value function above is one such example; another is the two-argument minimum $g(x,y) = \min\{x,y\}$. This function is concave, and its hypograph is

$$\operatorname{hypo} g = \{\,(x,y,z) \in \mathbb{R}\times\mathbb{R}\times\mathbb{R} \;|\; z \leq x,\; z \leq y\,\} \tag{3.80}$$

Figure 3.10 shows how the function `min` represents this hypograph in `cvx`. Notice that this function has a simple implementation as well—allowing the underlying solver to decide which it prefers to use. Of course, both `abs` and `min` are rather simple, but far more complex functions are possible. For example, graph implementations can be constructed for each of the norms examined in §1.1.

A subtle but important instance of nondifferentiability occur in functions that are discontinuous at the boundaries of their domains. These functions require special care as well, and graph implementations provide that. For example, consider the scalar entropy function

$$f : \mathbb{R} \to \mathbb{R}, \quad f(x) \triangleq \begin{cases} -x\log x & x > 0 \\ 0 & x = 0 \\ -\infty & x < 0 \end{cases} \tag{3.81}$$

This function is smooth over the positive interval, but it is discontinuous at the origin. This discontinuity causes difficulties for some numerical methods [96]. Using the hypograph

$$\operatorname{hypo} f = \mathbf{cl}\,\{\,(x,y)\in\mathbb{R}\times\mathbb{R}\;\mid\; x>0,\;\; -x\log x > y\,\}\,, \tag{3.82}$$

these difficulties can be solved. In Figure 3.10, we show the definition of a function `entropy` that refers to a set `hypo_entropy` representing this hypograph. We have chosen to omit the implementation of this set here, but it would certainly contain a definition of the barrier function

$$\phi:\mathbb{R}\times\mathbb{R}\to(\mathbb{R}\cup+\infty),\qquad \phi(x,y)=\begin{cases}-\log(-y-x\log x)-\log x & (x,y)\in\operatorname{Int}\operatorname{epi} f\\ +\infty & \text{otherwise}\end{cases} \tag{3.83}$$

[124], as well as an oracle to compute cutting planes $a_1x+a_2y\le b$, where

$$(a_1,a_2,b)\triangleq\begin{cases}(0,0,0) & (x,y)\in\operatorname{hypo} f\\ (-1,0,0) & x<0\\ (\log(y/2)+1,1,y/2) & x=0,\;y>0\\ (\log x+1,1,x) & x>0\end{cases} \tag{3.84}$$

Graph implementations can also be used to unify traditional, inequality-based nonlinear programming with conic programming. For example, consider the maximum singular value function

$$f:\mathbb{R}^{m\times n}\to\mathbb{R},\quad f(X)=\sigma_{\max}(X)=\sqrt{\lambda_{\max}(X^TX)} \tag{3.85}$$

This function is convex, and in theory could be used in a disciplined convex programming framework. The epigraph of f is

$$\begin{aligned}\operatorname{epi} f &= \left\{\,(X,y)\in\mathbb{R}^{m\times n}\times\mathbb{R}\;\mid\;\sigma_{\max}(X)\le y\,\right\}\\ &= \left\{\,(X,y)\;\middle|\;\begin{bmatrix} yI & X\\ X^T & yI\end{bmatrix}\in\mathcal{S}_+^{m+n},\;y\ge 0\right\}\end{aligned} \tag{3.86}$$

where $\mathcal{S}_+^{m+n}$ is the set of $(m+n)\times(m+n)$ positive semidefinite matrices. For someone who is familiar with semidefinite programming, (3.86) is probably quite familiar. Burying this construct within an implementation in the atom library, however, enables people who are *not* comfortable with semidefinite programming to take advantage of its benefits in a traditional NLP-style problem.

Graph implementations are possible for sets as well, using a single *key* `:=` *value* pair `constraints`, which contains the set described as a disciplined convex feasibility problem (again, top-level rule **T3**). There are several reasons why this might be used. For example, graph implementations can be used to represent sets with non-empty interiors, such as the set of n-element probability distributions

$$S=\left\{x\in\mathbb{R}^n\;\mid\;x\ge 0,\;\vec{1}^Tx=1\right\} \tag{3.87}$$

A cvx version of this set is given in Figure 3.10. Graph implementations can also be used to represent sets using a sequence of smooth inequalities so that smooth CP solvers can support them. For example, the second-order cone

$$\mathcal{Q}^n = \{\, (x,y) \in \mathbb{R}^n \times \mathbb{R} \;\mid\; \|x\|_2 \leq y \,\} \tag{3.88}$$

can be represented by smooth inequalities as follows:

$$\mathcal{Q}^n = \mathbf{cl}\left\{\, (x,y) \in \mathbb{R}^n \times \mathbb{R} \;\mid\; x^T x/y - y \leq 0,\; y > 0 \,\right\} \tag{3.89}$$

The concept of graph implementations is based on relatively basic principles of convex analysis. And yet, an applications-oriented user—someone who is not expert in convex optimization—is not likely to be constructing new atoms with graph implementations. Indeed they are not likely to be constructing new simple implementations either. Thankfully, they do not need to *build* them in order to *use* them. The implementations themselves can be built by those with more expertise in such details, and shared with applications-oriented users. As the development of the cvx framework continues, its authors will build a library of common and useful functions; and we hope that others will do so and share them with the community of users.

In §4.2.3, we show how graph implementations are used when solving problems. In short, their DCP representations are incorporated into the models that call them, not unlike an inline function in C++. Graph implementations are *algorithm agnostic*, in that any numerical method for convex programming can utilize graph implementations. So algorithms that do not "natively" support non-differentiable functions or sets with non-empty interiors can do so using of graph implementations.

3.6 Conclusion

We have introduced a new methodology for constructing convex programs called *disciplined convex programming*. The key components of the methodology are an extensible *atom library* of functions and sets with known properties of shape, monotonicity, and range; and a *ruleset* that governs how atoms may be combined with parameters, variables, and numeric constants to construct valid constraints and objective functions. Disciplined convex programs are a strict subset of convex programs, and yet the extensibility of the atom library ensures that this subset is representative: that is, that all finite-dimensional convex programs can be expressed in disciplined form.

An important feature of disciplined convex programming is the concept of *graph implementations* for functions and sets. Graph implementations allow functions and sets to be expressed in terms of other disciplined convex programs, and expand the class of functions and sets that can be practically represented in a modeling framework; in particular, nondifferentiable (convex and concave) functions are fully supported, as are (convex) sets with non-empty interiors. As a result, the modeler is free to disregard details such as nondifferentiability when constructing disciplined convex programs.

Chapter 4

Solving disciplined convex programs

In the previous chapter, we introduced the disciplined convex programming methodology as a tool for constructing convex programs. In this chapter, we discuss how disciplined convex programs can be solved. As depicted in Figure 4.1, the task can be divided into four stages:

- verification of adherence to the DCP ruleset (§3.3);
- *canonicalization*; that is, conversion to a solvable form (§4.2);
- numerical solution (§4.3); and
- *post processing*; that is, recovering primal and dual information for the original model from the solution to the canonicalized problem (§4.2.4).

We have already addressed the first of these stages in the previous chapter. There it was shown that the conventions enforced by disciplined convex programming replaced the intractable problem of determining if a nonlinear program is convex with a simpler, automatable task: verifying compliance to the DCP ruleset. In this chapter, we examine the remaining three stages of the solution process, and find that disciplined convex programming provides significant opportunities for simplification, automation, and improvement as well.

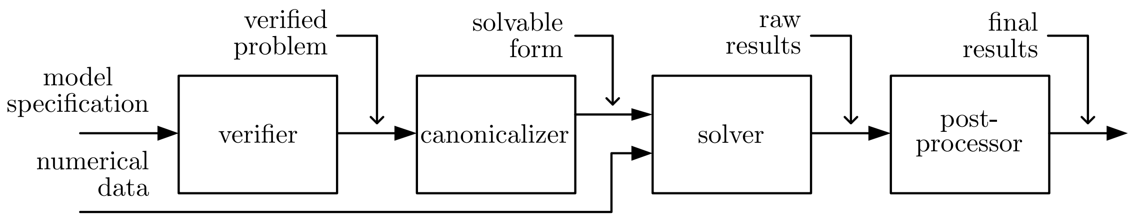

Figure 4.1: The DCP workflow.

4.1 The canonical form

A *canonical form* is a prototypical problem that represents an entire class of problems for the purposes of study or numerical solution. For example, much of the study of linear programming involves the so-called primal and dual canonical forms,

$$\begin{array}{rl} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array} \qquad \begin{array}{rl} \text{maximize} & b^T y \\ \text{subject to} & A^T y \leq c. \\ & \end{array} \tag{4.1}$$

A canonical form must be *universal*, in that every problem in the class it represents must be equivalent to a problem in canonical form. Indeed, all linear programs can be represented using *either* of the canonical forms in (4.1). (When we use the term *equivalent* here, we rely on an informal but practical definition given in [29]: two problems are equivalent if a solution to one readily yields a solution to the other, and vice versa.)

4.1.1 Our choice

We propose a single canonical form for disciplined convex programs:

$$\begin{array}{rl} \mathcal{P}_{\text{DCP}}\text{: minimize} & c^T x + d \\ \text{subject to} & Ax = b \\ & x \in S. \end{array} \tag{4.2}$$

The problem variable is $x \in \mathbb{R}^n$, and the parameters are $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $c \in \mathbb{R}^n$, $d \in \mathbb{R}$, and a convex set $S \subset \mathbb{R}^n$. Corresponding forms for maximization and feasibility logically follow:

$$\begin{array}{rl} \text{maximize} & -c^T x - d \\ \text{subject to} & Ax = b \\ & x \in S \end{array} \qquad \begin{array}{rl} \text{find} & x \in \mathbb{R}^n \\ \text{such that} & Ax = b \\ & x \in S, \end{array} \tag{4.3}$$

but the maximization can be solved as a minimization simply by negating the objective, and the feasibility problem by selecting $c = 0$, $d = 0$. So we focus on the minimization problem above.

If $S \triangleq \mathbb{R}^n_+$ and $d = 0$, the primal LP form in (4.1) is recovered—and this was not accidental. But it is the structure of the set S that distinguishes this canonical form from (4.1) or other canonical forms for mathematical programming. S is the Cartesian product of one or more sets S_k; *i.e.*,

$$S = S_1 \times S_2 \times \cdots \times S_L, \quad S_k \subseteq \mathbb{R}^{n_k}, \quad k = 1, \dots, L, \quad \textstyle\sum_{k=1}^L n_k \triangleq n. \tag{4.4}$$

The sets S_k, which we call the *nonlinearities* of the problem, are drawn from one of five categories:

- a group of free variables: $S_k \triangleq \mathbb{R}^{n_k}$;

- a group of nonnegative variables: $S_k \triangleq \mathbb{R}^{n_k}_+ = \{\, x \in \mathbb{R}^{n_k} \mid x_j \geq 0,\ j = 1, 2, \dots, n_k \,\}$;
- an *epigraph nonlinearity*: $S_k \triangleq \operatorname{epi} f = \left\{\, (u, v) \in \mathbb{R}^{n_k - 1} \times \mathbb{R} \mid f(u) \leq v \,\right\}$, where f is a convex function from the atom library;
- a *hypograph nonlinearity*: $S_k \triangleq \operatorname{hypo} g = \left\{\, (u, v) \in \mathbb{R}^{n_k - 1} \times \mathbb{R} \mid g(u) \geq v \,\right\}$, where g is a concave function from the atom library;
- a *set nonlinearity*: S_k is a convex set from the atom library.

(We hope the reader will forgive that the term "nonlinearity" is somewhat of a misnomer in the first two cases.) As we see in §4.3, numerical algorithms place further restrictions upon the nonlinearities.

It should be clear that $\mathcal{P}_{\text{DCP}}$ is a valid disciplined convex program, but it adopts a number of conventions not shared by the larger class of DCPs:

- The objective function is affine.
- There are no linear inequalities, except for simple nonnegative variables.
- There are no nonlinear inequalities, except for epigraph/hypograph nonlinearities.
- Each nonlinearity is *atomic*, which means three things:
  - it involves only a single function or set from the atom library;
  - each argument is a simple variable—*i.e.*, no affine combinations;
  - each variable appears exactly once across all nonlinearities.

  The effects of the nonlinearities are coupled through the equality constraints $Ax = b$.

The atomic property of the nonlinearities is the defining structural characteristic of the DCP canonical form, and has significant practical consequences that we explore in this chapter.

We often find it useful to consider two alternate forms for (4.2). First, let us partition x, A, and c to coincide with the Cartesian structure of S:

$$x \triangleq \begin{bmatrix} x_{(1)} \\ x_{(2)} \\ \vdots \\ x_{(L)} \end{bmatrix} \qquad A \triangleq \begin{bmatrix} A_{(1)} & A_{(2)} & \dots & A_{(L)} \end{bmatrix} \qquad c \triangleq \begin{bmatrix} c_{(1)} \\ c_{(2)} \\ \vdots \\ c_{(L)} \end{bmatrix}. \tag{4.5}$$

This produces what we call the *partitioned form* for $\mathcal{P}_{\text{DCP}}$:

$$\begin{array}{rl} \mathcal{P}_{\text{DCP}}\text{: minimize} & \sum_{k=1}^{L} c_{(k)}^T x_{(k)} + d \\ \text{subject to} & \sum_{k=1}^{L} A_{(k)} x_{(k)} = b \\ & x_{(k)} \in S_k \quad k = 1, 2, \dots, L. \end{array} \tag{4.6}$$

Some analysis requires the selection of a specific combination of nonlinearities. When that occurs we use this example:

$$\begin{array}{ll} \text{minimize} & c_x^T x + c_z^T z + c_u^T u + c_v^T v + d \\ \text{subject to} & A_x x + A_z z + A_u u + A_v v = b \\ & z \geq 0 \\ & f_k(u_{(k)}) \leq v_k, \quad k = 1, 2, \dots, N. \end{array} \tag{4.7}$$

The problem variables are

$$\begin{array}{c} x \in \mathbb{R}^{n_x}, \quad z \in \mathbb{R}^{n_z}, \quad u \in \mathbb{R}^{n_u}, \quad v \in \mathbb{R}^N \\ u \triangleq \begin{bmatrix} u_{(1)}^T & u_{(2)}^T & \dots & u_{(N)}^T \end{bmatrix}^T \quad u_{(k)} \in \mathbb{R}^{n_{f,k}},\ k = 1, 2, \dots, N \quad (\sum_{k=1} n_{f,k} \triangleq n_u) \end{array} \tag{4.8}$$

and the functions $f_k : \mathbb{R}^{n_{f,k}} \to \mathbb{R}$, $k = 1, 2, \dots, N$ are convex. It is not difficult to see that, with a simple rearrangement of the variables, this problem is a DCP in canonical form with n_x free variables, n_z nonnegative variables, and N epigraph nonlinearities $\operatorname{epi} f_k$.

4.1.2 The dual problem

The dual problem figures into much of our analysis so let us introduce it here. The exact form depends upon the specific structure of the nonlinearities, so let us use (4.7) above. The Lagrangian is

$$\begin{array}{l} L(x, z, u, v, y, s, \lambda) = c_x^T x + c_z^T z + c_u^T u + c_v^T v + d \\ \qquad - y^T (A_x x + A_z z + A_u u + A_v v - b) - s^T z - \sum_{k=1}^N \lambda_k (v_k - f_k(u_{(k)})), \end{array} \tag{4.9}$$

where $y \in \mathbb{R}^m$, $s \in \mathbb{R}^{n_z}$, and $\lambda \in \mathbb{R}^N$ are Lagrange multipliers for the equality constraints, the nonnegative variables, and the nonlinear inequalities, respectively. The Lagrange dual problem is

$$\begin{array}{ll} \text{maximize} & g(y, s, \lambda) \triangleq \inf_{x,z,u,v} L(x, z, u, v, y, s, \lambda) \\ \text{subject to} & s \geq 0,\ \lambda \geq 0, \end{array} \tag{4.10}$$

where $g(y, s, \lambda)$ is the dual function. Eliminating the primal variables yields

$$\begin{array}{ll} \text{maximize} & b^T y + d - \sum_{k=1}^N \lambda_k f_k^*(\lambda_k^{-1} p_{(k)}) \\ \text{subject to} & c_x - A_x^T y = 0 \\ & c_z - A_z^T y = s \\ & \left. \begin{array}{l} c_{u,k} - A_{u,k}^T y = -p_{(k)} \\ c_{v,k} - A_{v,k}^T y = \lambda_k \end{array} \right\} k = 1, 2, \dots, N \\ & z \geq 0,\ \lambda \geq 0, \end{array} \tag{4.11}$$

where f_k^* is the conjugate function for f_k; *i.e.*,

$$f^* : \mathbb{R}^n \to (\mathbb{R} \cup +\infty), \quad f_k^*(p) \triangleq \sup_u p^T u - f_k(u), \tag{4.12}$$

and new variables $p_{(k)} \in \mathbb{R}^{n_{f,k}}$, $k = 1, 2, \ldots, N$ have been defined for convenience. Because the functions f_k^* may be extended-valued, additional constraints $p_{(k)} \in \lambda_k \,\mathbf{dom}\, f_k^*$, $k = 1, 2, \ldots, N$ are implied here as well. If the functions f_k are differentiable, then the Wolfe dual

$$\begin{array}{ll} \text{maximize} & b^T y + d + \sum_{k=1}^N \lambda_k (f_k(u_{(k)}) - u_{(k)}^T \nabla f_k(u_{(k)})) \\ \text{subject to} & c_x - A_x^T y = 0 \\ & c_z - A_z^T y = s \\ & \left.\begin{array}{l} c_{u,k} - A_{u,k}^T y = -\lambda_k \nabla f_k(u_{(k)}) \\ c_{v,k} - A_{v,k}^T y = \lambda_k \end{array}\right\} k = 3, 4, \ldots, L \\ & z \geq 0, \;\; \lambda \geq 0 \end{array} \tag{4.13}$$

provides more explicit conditions for dual feasibility.

Given a feasible primal point (x, z, u, v) and a feasible dual point (y, s, λ), the duality gap $\eta(x, z, u, v, y, s, \lambda)$ is the difference between the primal and dual objectives. This quantity is provably nonnegative:

$$\begin{aligned} \eta(x, z, u, v, y, s, \lambda) &= c_x^T x + c_z^T z + c_u^T u + c_v^T v - b^T y - \textstyle\sum_{k=1}^N \lambda_k f_k^*(p_{(k)}) \\ &= s^T z + \textstyle\sum_{k=1}^N \lambda_k (v_k - u_{(k)}^T p_{(k)}) - \sum_{k=1}^N \lambda_k f_k^*(p_{(k)}) \\ &\geq s^T z + \textstyle\sum_{k=1}^N \lambda_k (v_k - f_k(u_{(k)})) \geq 0. \end{aligned} \tag{4.14}$$

The duality gap cannot be exactly determined from the primal and feasible dual points unless the conjugates are computed exactly, or if u satisfies the equality constraints of the Wolfe dual (if applicable). In such cases, the duality gap is exactly

$$\hat{\eta}(x, z, u, v, y, s, \lambda) \triangleq s^T z + \sum_{k=1}^N \lambda_k (v_k - f_k(u_{(k)})). \tag{4.15}$$

This quantity is often called the *surrogate duality gap* [29] and is sometimes employed in numerical methods, even when $\eta \neq \hat{\eta}$. Strong duality is not assured unless one has a constraint qualification such as the appropriate Slater condition,

$$\exists (x, z, u, v) \quad A_x x + A_z z + A_u u + A_v v = b, \quad z \geq 0, \quad f_k(u_{(k)}) < v_{(k)}, \quad k = 1, 2, \ldots, N. \tag{4.16}$$

Without such a qualification, it is possible for the duality gap to be bounded away from zero.

4.2 Canonicalization and dual recovery

We call the process of finding a problem in canonical form equivalent to a given DCP *canonicalization*. Closely tied to canonicalization is *dual recovery*, the recovery of dual information for the original DCP given dual information computed for its canonical form.

One can think of canonicalization as the application of a series of equivalency-preserving transformations to a DCP. We can divide these transformations into three groups:

- *linearization* (§4.2.1): removing nonlinear functions from inequality constraints and objective functions, and adding slack variables to convert inequalities to equalities;
- *set decomposition* (§4.2.2): converting set membership constraints into equality constraints and atomic nonlinearities;
- *graph expansion* (§4.2.3): replacing a function or set with its graph implementation, as directed by the requirements of a solver.

Of course, there are more trivial reformatting issues as well, such as moving all variable expressions to the left-hand sides of equality constraints, and all constant expressions to the right. We do not address them here because they lack significant mathematical consequence.

With the exception of the use of slack variables, these canonicalization operations are largely unique to disciplined convex programming. In §4.2.1-§4.2.3, we describe these operations more thoroughly. It is important to note that many of the transformations *assume* adherence to the DCP ruleset; that is, the ruleset provides sufficient conditions to guarantee the equivalency of these transformations. Put another way, once a problem has been verified as a DCP, canonicalization may be applied automatically and mechanically, without further verification of equivalency.

In §4.2.4, we examine the issue of dual recovery. We claim that canonicalization has a relatively benign effect on dual information, and a straightforward process recovers Lagrange multiplier information for the original problem.

4.2.1 Linearization

Linearization involves three tasks: removing nonlinearities from objective functions and inequality constraints; insuring that nonlinearities are atomic; and adding slack variables as needed to eliminate linear inequalities. It is the first task that gives this operation its name.

To illustrate this first task, suppose that $x \in \mathbb{R}^n$ is a variable, a, $b \in \mathbb{R}^n$, and $c \in \mathbb{R}^L$ are parameters, and f_j, $j = 1, 2, \dots, L$ are convex functions from the atom library; and consider the nonlinear inequality

$$a + b^T x + \sum_{j=1}^{L} c_j f_j(\mathit{arg}_{j,1}, \mathit{arg}_{j,2}, \dots, \mathit{arg}_{j,m_j}) \leq 0. \tag{4.17}$$

The quantities $\mathit{arg}_{j,i}$ are expressions whose exact form is not relevant here—except that they obey the DCP ruleset. The reader may recognize the left-hand side of (4.17) from (3.3), the prototypical form for all numeric expressions in DCPs, as dictated by the no-product rules (§3.1.2). Because the functions f_j are convex, the sign rules (§3.1.3) dictate that $c \geq 0$. Then (4.17) is equivalent to the system of $L + 1$ inequalities

$$a + b^T x + c^T y \leq 0 \qquad f_j(\mathit{arg}_{j,1}, \mathit{arg}_{j,2}, \dots, \mathit{arg}_{j,m_j}) \leq y_j, \quad j = 1, 2, \dots, L. \tag{4.18}$$

where $y \in \mathbb{R}^L$ are new variables created just for this purpose. Adding a slack variable $z \in \mathbb{R}$ to the linear inequality yields the result,

$$a + b^T x + c^T y + z = 0 \qquad z \geq 0 \qquad f_j(\mathit{arg}_{j,1}, \mathit{arg}_{j,2}, \ldots, \mathit{arg}_{j,m_j}) \leq y_j, \quad j = 1, 2, \ldots, L. \tag{4.19}$$

The equivalence of (4.19) to (4.17), under the stated conditions that the functions f_j are convex and $c \geq 0$, should be obvious. However, this example does illustrate the point made above concerning the dependence of these conversion steps on adherence to the DCP ruleset. The preservation of equivalency in the transformation of (4.17) to (4.19) *depends* upon the adherence to the sign rules—*i.e.*, the fact that $c \geq 0$. But since compliance to the ruleset is verified before canonicalization starts, there is no need to check the signs of the elements of c at all.

To linearize an objective function, one can simply move the nonlinearity into an inequality constraint using the classic epigraph conversion, and linearize that constraint. But it is not difficult to see that the conversion process can be applied directly to the objective function; *e.g.*,

$$\text{minimize} \quad a + b^T x + \sum_{j=1}^{L} c_j f_j(\cdot) \qquad \Longrightarrow \qquad \begin{array}{ll} \text{minimize} & a + b^T x + c^T y \\ \text{subject to} & f_j(\cdot) \leq y_j, \quad j = 1, 2, \ldots, L. \end{array} \tag{4.20}$$

This first task generates new nonlinearities of the form $f_j(\cdot) \leq t_j$ (or $f_j(\cdot) \geq t_j$, if f_j is concave). These expressions will not always be atomic, and may therefore require further transformation. For example, suppose that f, g are convex functions, x, y, z are problem variables, and A, b are parameters. Then the constraint

$$f(Ax + b) \leq y \tag{4.21}$$

is not atomic because the function argument is a not simple variable. Similarly, the constraints

$$f(y) \leq z \qquad g(x) \leq y \tag{4.22}$$

are not atomic because both refer to the same variable y. In each instance, the remedy involves creating additional variables and adding equality constraints to produce equivalent results:

$$w_1 = Az + b \qquad f(w_1) \leq y \tag{4.23}$$

$$w_2 = y \qquad f(y) \leq z \qquad g(x) \leq w_2. \tag{4.24}$$

These transformations do not depend upon adherence to the DCP ruleset for equivalence.

A more challenging case arises when we are confronted with a nonlinear composition such as $f(g(x)) \leq y$. In a DCP, such compositions can only occur under the conditions established by the rules in §3.1.4. In this case, f must be *nondecreasing* over the input range $g(\mathbf{dom}\, g)$, allowing us to establish the following equivalence.

Lemma 6. *Suppose that $f : \mathbb{R} \to (\mathbb{R} \cup +\infty)$ and $g : \mathbb{R}^n \to (\mathbb{R} \cup +\infty)$ are convex functions, and f*

is nondecreasing over $g(\mathbf{dom}\, g)$; that is, they satisfy composition rule ***C1*** *in §3.1.4. Then*

$$f(g(x)) \leq y \quad \Longleftrightarrow \quad \exists w_1, w_2 \quad f(w_1) \leq y,\ g(x) \leq w_2,\ w_1 = w_2 \tag{4.25}$$

Proof. First, assume the left-hand side is true; then the right-hand side is satisfied with $w_1 = w_2 = g(x)$. Now assume that the right-hand side is true; then

$$f(w_1) \leq y < +\infty \quad \Longrightarrow \quad w_1 \in \mathbf{dom}\, f. \tag{4.26}$$

Furthermore, Lemma 5 ensures that f is nondecreasing over a half-line that includes $g(\mathbf{dom}\, g)$ and extends to $+\infty$. Since $g(x) \leq w_2$, this half-line must include w as well. Therefore,

$$g(x) \leq w_2 \quad \Longrightarrow \quad f(g(x)) \leq f(w_2) = f(w_1) \leq y \quad \Longrightarrow \quad f(g(x)) \leq y. \qquad \square \tag{4.27}$$

In other words, the monotonicity of f, guaranteed in advance by the DCP ruleset, allows the expression $f(g(x)) \leq y$ to be decomposed into an equivalent set of canonicalized constraints as given in (4.25). Equivalencies for other compositions can also be established:

$$f \text{ convex, nonincreasing; } g \text{ concave: } f(g(x)) \leq y \iff f(w_1) \leq y,\, g(x) \geq w_2,\, w_1 = w_2; \tag{4.28}$$

$$f \text{ concave, nondecreasing; } g \text{ concave: } f(g(x)) \geq y \iff f(w_1) \geq y,\, g(x) \geq w_2,\, w_1 = w_2; \tag{4.29}$$

$$f \text{ concave, nonincreasing; } g \text{ convex: } f(g(x)) \geq y \iff f(w_1) \geq y,\, g(x) \leq w_2,\, w_1 = w_2. \tag{4.30}$$

We emphasize again that the combinations of convexity, concavity, and monotonicity stated here form sufficient conditions under which these decompositions are equivalent. All *disciplined* convex programs satisfy those conditions, but not all general convex programs do.

Linearization is a naturally recursive process, because the arguments to functions in nonlinear expressions may be themselves require linearization. It is most efficient for implementation purposes to proceed in a "bottom-up" fashion, by canonicalizing function arguments *first*, and *then* handling the top-level expressions in nonlinear inequalities and objective functions. Here is a full statement of this "bottom-up" linearization process:

Linearization. Given a scalar expression that conforms to the DCP ruleset, and can therefore be expressed in the form

$$a + b^T x + \sum_{j=1}^{L} c_j f_j(\mathit{arg}_{j,1}, \mathit{arg}_{j,2}, \ldots, \mathit{arg}_{j,m_j}), \tag{4.31}$$

linearization proceeds as follows: for each $j = 1, 2, \ldots, L$,

1. Apply this linearization process to each of the arguments $\mathit{arg}_{j,1}, \mathit{arg}_{j,2}, \ldots, \mathit{arg}_{j,m_j}$, so that each argument $\mathit{arg}_{j,k}$ is now affine.

2. Perform the following for each $j = 1, 2, \ldots, L$ for which f_j is convex or concave:
   (a) Replace the function call $f_j(\cdot)$ in (4.31) by a new temporary variable v_j.
   (b) Create m_j new equality constraints $u_{j,k} = \mathit{arg}_{j,k}$, $k = 1, 2, \ldots, m_j$.
   (c) Create a new constraint containing f_j:
      - if f_j is convex, a new epigraph nonlinearity $f_j(u_1, u_2, \ldots, u_{m_j}) \leq v_j$;
      - if f_j is concave, a new hypograph nonlinearity $f_j(u_1, u_2, \ldots, u_{m_j}) \geq v_j$;
3. If the original expression (4.31) was either
   - the left-hand side of a less-than ($\leq$) inequality, or
   - the right-hand side of a greater-than ($\geq$) inequality,

   add a new slack variable $z \geq 0$ to it, converting the constraint to an equality ($=$).

To illustrate this process, let us apply it to the nonlinear inequality

$$\exp(y) \leq \log(a\sqrt{x} + b) \tag{4.32}$$

found in the example problem (3.60); x, y are variables, and a, b are parameters. This expression obeys the DCP ruleset only if $a \geq 0$, so we assume this is the case. The first function reference we encounter is $\exp(y)$, which linearizes rather simply:

$$\exp(y) \quad \Longrightarrow \quad t_2, \quad \exp(t_1) \leq t_2, \quad t_1 = y, \tag{4.33}$$

producing the intermediate result

$$t_1 = y \qquad \begin{array}{c} t_2 \leq \log(a\sqrt{x} + b) \\ \exp(t_1) \leq t_2. \end{array} \tag{4.34}$$

For the next function $\log(a\sqrt{x} + b)$, the argument $a\sqrt{x} + b$ must be linearized first:

$$\begin{aligned} a\sqrt{x} + b \quad &\Longrightarrow \quad at_4 + b, \quad \sqrt{t_3} \geq t_4, \quad t_3 = x & (4.35) \\ \log(at_4 + b) \quad &\Longrightarrow \quad t_6, \quad \log(t_5) \geq t_6, \quad t_5 = at_4 + b. & (4.36) \end{aligned}$$

Thus the nonlinear inequality has been reduced to $t_2 \leq t_6$. Adding a slack variable to convert it to an equality, and performing some rearrangement, we arrive at the final result:

$$\begin{array}{cc} t_2 - t_6 + t_7 = 0 & \exp(t_1) \leq t_2 \\ -y + t_1 = 0 & \sqrt{t_3} \geq t_4 \\ -x + t_3 = 0 & \log(t_5) \geq t_6 \\ at_4 - t_5 = -b & t_7 \geq 0. \end{array} \tag{4.37}$$

In this case, the original constraint was decomposed into four equality constraints, three atomic nonlinear inequalities, and one slack variable.

At first, this conversion process may seem inefficient. However, linearization typically results in very sparse problem structure; by exploiting that structure, much of this apparent added cost can be recovered. Still, it could be argued that a preferable approach is the one employed in traditional NLP modeling systems, which is effectively to define a new function

$$f(x, y) \triangleq \exp(y) - \log(a\sqrt{x} + b) \tag{4.38}$$

and use automatic differentiation to compute its derivatives. There are several reasons why we prefer the expanded approach. First of all, functions in disciplined convex programming are not required to be differentiable, so an approach that relies solely on automatic differentiation would not be feasible. Secondly, decoupling the nonlinearities simplifies the task of finding initial points for many numerical algorithms in cases where the functions involved have limited domains. For example, for our example (4.32), care would have to be taken to ensure that $x \geq 0$ and $x > -b/a$; in the linearized version, it is easier to ensure in (4.37) that $t_3 \geq 0$ and $t_5 > 0$. Thirdly, even in cases where automatic differentiation would be possible, we suspect that the added cost of solving the larger linear system will not be significantly greater than the cost of the automatic differentiation itself, as long as sparsity can be exploited effectively. After all, the computations involved in automatic differentiation can be expressed in terms of matrix operations; see, for example, [34, 77].

4.2.2 Set decomposition

Disciplined convex programming allows set membership constraints that involve arbitrary intersections, Cartesian products, and affine combinations of convex sets. The canonical form requires that set membership constraints be atomic, so composite expressions must be separated.

The process is straightforward, and several examples should suffice to illustrate it. Suppose that S_1 and S_2 are convex sets, a, b are parameters, and x, y are variables. Then Cartesian products are separated as follows:

$$(ax + by, y) \in S_1 \times S_2 \quad \Longleftrightarrow \quad ax + by \in S_1 \quad y \in S_2, \tag{4.39}$$

Intersections are separated in a similar fashion:

$$x \in S_1 \cap S_2 \quad \Longleftrightarrow \quad x \in S_1 \quad x \in S_2. \tag{4.40}$$

For affine combinations of sets, we can draw directly from the mathematical definition:

$$ax + by \in 2S_1 + 3S_2 \qquad \Longleftrightarrow \qquad ax + by = 2w_1 + 3w_2 \quad w_1 \in S_1 \quad w_2 \in S_2, \tag{4.41}$$

where w_1, w_2 are new variables. Once the set membership constraints have been separated, steps must be taken to ensure that they are atomic, including the creation of new variables and addition of new equality constraints, if necessary. For example,

$$Ax + b \in S_1 \quad \Longrightarrow \quad w_1 = Ax + b,\ w_1 \in S_1 \tag{4.42}$$

$$(x, y) \in S_1,\ (x, z) \in S_2 \quad \Longrightarrow \quad w_2 = x,\ (x, y) \in S_1,\ (w_2, z) \in S_2. \tag{4.43}$$

Combining these steps yields a complete description of the set decomposition process:

Set decomposition. Given an expression of the form

$$(\mathit{lexp}_1, \mathit{lexp}_2, \ldots, \mathit{lexp}_m) \in S, \tag{4.44}$$

where $\mathit{lexp}_1, \mathit{lexp}_2, \ldots, \mathit{lexp}_m$ are affine numeric expressions and S is a convex set expression, decomposition proceeds as follows:

1. If S is a single set from the atom library, then replace (4.44) with

$$\mathit{lexp}_1 = u_1,\ \mathit{lexp}_2 = u_2,\ \ldots,\ \mathit{lexp}_m = u_m \qquad (u_1, u_2, \ldots, u_m) \in S. \tag{4.45}$$

2. If $S = \mathit{rexp}_1 \times \mathit{rexp}_2 \times \cdots \times \mathit{rexp}_m$, then replace (4.44) with

$$\mathit{lexp}_k \in \mathit{rexp}_k \qquad k = 1, 2, \ldots, m \tag{4.46}$$

and perform a full set atomization on each of these constraints.

3. If $S = \mathit{rexp}_1 \cap \mathit{rexp}_2 \cap \cdots \cap \mathit{rexp}_L$, then replace (4.44) with

$$(\mathit{lexp}_1, \mathit{lexp}_2, \ldots, \mathit{lexp}_m) \in \mathit{rexp}_j \qquad j = 1, 2, \ldots, m \tag{4.47}$$

and perform a full set atomization on each of these constraints.

4. If S is an affine combination of sets, then we can assume that $m = 1$, and S can be expressed in the form

$$S = a + \sum_{i=1}^{m_0} b_i S_{0,i} + \sum_{j=1}^{L} c_j f_j(S_{j,1}, S_{j,2}, \ldots, S_{j,m_j}). \tag{4.48}$$

Replace (4.44) with

$$\begin{aligned} &u_{j,i} \in S_{j,i} \qquad i = 0, 1, \ldots, L,\ j = 1, 2, \ldots, m_i \\ &\mathit{lexp}_1 = a + \sum_{i=1}^{m_0} b_i u_{0,i} + \sum_{j=1}^{L} c_j f(u_{j,1}, u_{j,2}, \ldots, u_{j,m_j}) \end{aligned} \tag{4.49}$$

and perform a full set decomposition on each of the expressions $u_{j,i} \in S_{j,i}$.

To illustrate this process, suppose that S_1, S_2, S_3 are sets from the atom library and x, y are variables, and consider the following set membership constraint:

$$(x, 2x+y) \in (S_1 \times (2S_1 + 3S_2)) \cap S_3. \tag{4.50}$$

The outermost operation is an intersection, which is split apart to yield

$$(x, 2x+y) \in S_1 \times (2S_1 + 3S_2) \qquad (x, 2x+y) \in S_3. \tag{4.51}$$

The first set membership constraint is a direct product, which splits apart to yield

$$x \in S_1 \qquad 2x+y \in 2S_2 + 3S_2 \qquad (x, 2x+y) \in S_3. \tag{4.52}$$

The first and third constraints involve a single set, so each must be made atomic:

$$t_1 = t_2 = x \qquad t_3 = 2x+y \qquad t_1 \in S_1 \qquad (t_2, t_3) \in S_3. \tag{4.53}$$

Finally, the second expression involves an affine set combination:

$$2x+y = 2t_4 + 3t_5 \qquad t_4 \in S_1 \qquad t_5 \in S_2. \tag{4.54}$$

So the final result, after some rearrangement, is

$$\begin{array}{cc} x - t_1 = 0 & t_1 \in S_1 \\ x - t_2 = 0 & (t_2, t_3) \in S_3 \\ 2x + y - t_3 = 0 & t_4 \in S_1 \\ 2x + y - 2t_4 - 3t_5 = 0 & t_5 \in S_2 \end{array} \tag{4.55}$$

In this case, the original constraint was decomposed into four equality constraints and four atomic set membership nonlinearities. As with linearization, set decomposition typically produces sparse results that can be exploited to mitigate the cost of solving a larger problem.

4.2.3 Graph expansion

Graph expansion is the process of replacing the reference to an atom with its graph implementation. As discussed in §1.1.7, it encapsulates the transformations performed manually in §1.1 and Chapter 2 to convert problems to solvable form. To explain, let us consider the partitioned form (4.6),

$$\begin{array}{ll} \text{minimize} & \sum_{k=1}^{L} c_{(k)}^T x_{(k)} + d \\ \text{subject to} & \sum_{k=1}^{L} A_{(k)} x_{(k)} = b \\ & x_{(k)} \in S_k, \ k = 1, 2, \dots, L \end{array} \tag{4.56}$$

and suppose that we are given a graph implementation for the final nonlinearity S_L. Let us canonicalize that implementation, and express it as follows:

$$\bar{x} \in S_L \quad \Longleftrightarrow \quad \begin{array}{l} \bar{A}z = \bar{b} \\ z \in \bar{S} \triangleq \bar{S}_1 \times \bar{S}_2 \times \cdots \times \bar{S}_{\bar{L}}. \end{array} \qquad z \triangleq \bar{P} \begin{bmatrix} \bar{x} \\ y \end{bmatrix}. \tag{4.57}$$

The variables $y \in \mathbb{R}^{n_y}$ are additional variables used in the construction of the graph implementation. The matrix $\bar{P}$ is a permutation matrix that reflects any rearrangement of $\bar{x}$ and y that may be present in the implementation. Replacing the set S_L with this definition yields

$$\begin{array}{rl} \mathcal{P}_{\text{DCP}}\text{: minimize} & \sum_{k=1}^{L} c_{(k)}^T x_{(k)} + d \\ \text{subject to} & \sum_{k=1}^{L} A_{(k)} x_{(k)} = b \\ & \bar{A}\bar{P} \begin{bmatrix} x_{(L)} \\ y \end{bmatrix} = \bar{b} \\ & x_{(k)} \in S_k, \;\; k = 1, 2, \dots, L-1 \\ & \bar{P} \begin{bmatrix} x_{(L)} \\ y \end{bmatrix} \in \bar{S} \triangleq \bar{S}_1 \times \bar{S}_2 \times \cdots \times \bar{S}_{\bar{L}}. \end{array} \tag{4.58}$$

The graph expansion adds new variables (and possibly rearranges some), adds new equality constraints, and substitutes the nonlinearity $S_{(k)}$ for a new set $\bar{S}$, which may itself be the Cartesian product of several nonlinearities. No further canonicalization steps are necessary.

In §3.5.2, we claimed that any common numerical algorithm for convex programming can exploit graph implementations. It should now be clear why this is the case: because expansion occurs before the algorithm is called. That is not to say, however, that graph implementations should be expanded without regard for the choice of that numerical algorithm. Atoms may have *both* simple and graph implementations, and the best choice depends upon the specific algorithm. For example, a function could have both a subgradient calculator and a graph implementation; the former would be preferred for a cutting-plane algorithm, and the latter for other methods. So multiple implementations of the same atom do not just allow a wider variety of algorithms to be employed, but also allows them to be represented in the most efficient manner in each case.

For a concrete illustration of graph expansion, let us revisit the problem from Chapter 2, which yields the following canonical form:

$$\begin{array}{rl} \text{minimize} & \|Ax - b\| \\ \text{subject to} & Cx = d \\ & l \le x \le u \end{array} \quad \Longrightarrow \quad \begin{array}{rlr} \text{minimize} & y & \\ \text{subject to} & Cx = d & f(w) \le y \\ & x - z_1 = l & z_1 \ge 0 \\ & x + z_2 = u & z_2 \ge 0, \\ & Ax - w = b & \end{array} \tag{4.59}$$

where y, z_1, and z_2 are additional variables and $f(w) \triangleq \|w\|$. This is the final canonical form if a cutting-plane algorithm is being used, and the norm function $f(w)$ has a subgradient implementation. For most other cases, however, a graph implementation will be employed.

Let us consider the ℓ_2 and ℓ_∞ cases here. For the ℓ_2 case, we recognize the equivalency

$$y > 0,\ \|w\|_2 \leq y \quad \Longleftrightarrow \quad \phi(w,y) \leq 0 \quad \Longrightarrow \quad \bar{y} = 0,\ \phi(w,y) \leq \bar{y}, \tag{4.60}$$

where ϕ is a twice-differentiable function

$$\phi : (\mathbb{R}^m \times \mathbb{R}) \to (\mathbb{R} \cup +\infty), \quad \phi(w,y) \triangleq \begin{cases} w^T w/y - y & y > 0 \\ +\infty & \text{otherwise,} \end{cases} \tag{4.61}$$

The variable $\bar{y}$ was introduced for canonicalization purposes. Graph expansion yields the DCP,

$$\begin{array}{ll} \text{minimize} & y \\ \text{subject to} & Cx = d \qquad\qquad z_1 \geq 0 \\ & x - z_1 = l \qquad\quad z_2 \geq 0 \\ & x + z_2 = u \qquad\quad \phi(w,y) \leq \bar{y} \\ & Ax - w = b \\ & \bar{y} = 0. \end{array} \tag{4.62}$$

For the ℓ_∞ case, the epigraph of f provides the following equivalency:

$$\|w\|_\infty \leq y \quad \Longleftrightarrow \quad \begin{array}{l} w = w_+ - w_- \\ \vec{1}y = w_+ + w_- \end{array} \quad w_+, w_- \geq 0, \tag{4.63}$$

which is already in canonical form. Performing graph expansion yields

$$\begin{array}{lll} \text{minimize} & y & \\ \text{subject to} & Cx = d & z_1 \geq 0 \\ & x - z_1 = l & z_2 \geq 0 \\ & x + z_2 = u & w_+ \geq 0 \\ & Ax - w = b & w_- \geq 0 \\ & w - w_+ + w_- = 0 & \\ & y\vec{1} - w_+ - w_- = 0. & \end{array} \tag{4.64}$$

4.2.4 Dual recovery

Because the original problem variables remain in the canonical form, it is straightforward to recover solutions to the original problem from solutions to the canonical form. However, the canonicalization process inevitably transforms the dual problem as well. It is not immediately clear how to similarly obtain *dual information*—say, certificates of infeasibility or ϵ-optimality—for the original problem given such information for the canonical form. We call this task *dual recovery*. Thankfully, dual recovery proves to be reasonably simple, given the canonicalization process described in this chapter.

To see what we mean, consider a simple DCP with a single nonlinear inequality constraint:

$$\begin{array}{rl} \mathcal{P}_1\text{: minimize} & q^T x \\ \text{subject to} & a + b^T x + \sum_{j=1}^{L} c_j f_j(x) \leq 0. \end{array} \tag{4.65}$$

The problem variable is $x \in \mathbb{R}^n$, the parameters are $a, r \in \mathbb{R}$, $b, q \in \mathbb{R}^n$, $c \in \mathbb{R}^L$, and the functions $f_j(x)$, $j = 0, 2, \dots, L$, are all convex. The sign rules dictate that $c \geq 0$. Linearization produces the canonical form

$$\begin{array}{rl} \mathcal{P}_2\text{: minimize} & q^T x \\ \text{subject to} & a + b^T x + c^T y + z = 0 \\ & \left.\begin{array}{l} \bar{x}_{(j)} = x \\ f_j(x_{(j)}) \leq y_j \end{array}\right\} j = 1, 2, \dots, L \\ & z \geq 0. \end{array} \tag{4.66}$$

Let us compare the dual functions for the original problem $\mathcal{P}_1$,

$$g_1(\lambda) \triangleq \inf_x q^T x + \lambda \left(a + b^T x + \sum_{j=1}^{L} c_j f_j(x) \right), \tag{4.67}$$

and for the canonical form $\mathcal{P}_2$,

$$\begin{aligned} g_2(\nu, \bar{\nu}, \bar{\lambda}, s) \triangleq \inf_{x, \bar{x}_{(1)}, \dots, \bar{x}_{(L)}, y, z} & q^T x + \nu(a + b^T x + c^T y + z) - sz \\ & + \sum_{j=1}^{L} \nu_{(j)}^T (x_{(j)} - x) + \bar{\lambda}_j (f_j(x_{(j)}) - y_j). \end{aligned} \tag{4.68}$$

Lemma 7. $g_2(\nu, \bar{\nu}, \bar{\lambda}, s) \leq g_1(\nu)$.

Proof. Rearranging g_2 reveals

$$\begin{aligned} g_2(\nu, \bar{\nu}, \bar{\lambda}, s) = \inf_x q^T x + \nu(a + b^T x) + \sum_{j=1}^{L} \inf_{\bar{x}_{(j)}} \bar{\nu}_{(j)}^T (\bar{x}_{(j)} - x) + \bar{\lambda}_j f_j(\bar{x}_{(j)}) \\ + \sum_{j=1}^{L} \inf_{y_j} (\nu c_j - \bar{\lambda}_j) y_j + \inf_z (\nu - s) z. \end{aligned} \tag{4.69}$$

This rearranged version makes plain that $g_2(\nu, \bar{\nu}, \bar{\lambda}, s) = -\infty$ unless

$$\nu = s, \qquad \bar{\lambda}_j = \nu c_j, \ j = 1, 2, \dots, L. \tag{4.70}$$

Clearly $g_2(\nu, \bar{\nu}, \bar{\lambda}, s) \leq g_1(\nu)$ when those conditions are not met. When they are met, g_2 reduces to

$$g_2(\nu, \bar{\nu}, \bar{\lambda}, s) = \inf_x q^T x + \nu(a + b^T x) + \sum_{j=1}^{L} \inf_{\bar{x}_{(j)}} \bar{\nu}_{(j)}^T (\bar{x}_{(j)} - x) + \nu c_j f_j(\bar{x}_{(j)}). \tag{4.71}$$

An upper bound on the infima over $\bar{x}_{(j)}$ is obtained if we select $\bar{x}_{(j)} = x$, so

$$g_2(\nu, \bar{\nu}, \bar{\lambda}, s) \leq \inf_x f_0(x) + \nu \left(a + b^T x + \sum_{j=1}^{L} c_j f_j(x) \right) = g_1(\nu). \qquad \Box \tag{4.72}$$

Several conclusions can be drawn from this result. The first is that

$$g_2(\nu, \bar{\nu}, \bar{\lambda}, s) > -\infty \quad \Longrightarrow \quad g_1(\nu) > -\infty, \tag{4.73}$$

which means that any dual feasible point $(\nu, \bar{\nu}, \bar{\lambda}, s)$ for the canonicalized problem $\mathcal{P}_2$ produces a dual feasible point $\lambda \triangleq \nu$ for the original problem $\mathcal{P}_1$. Secondly, given a primal feasible point x and that same dual feasible point,

$$q^T x - g_2(\nu, \bar{\nu}, \bar{\lambda}, s) \geq q^T x - g_1(\nu), \tag{4.74}$$

which means that g_2 is *conservative* in the duality gap sense: that if $((x, y, z), (\nu, \bar{\nu}, \bar{\lambda}, s))$ is a certificate of ϵ-optimality for $\mathcal{P}_2$, then so is $(x, \lambda \triangleq \nu)$ for $\mathcal{P}_1$. Finally, by selecting $q = 0$, we see that this same result proves true for certificates of infeasibility as well. In summary, recovering the Lagrange multiplier for an inequality constraint simply requires selecting the appropriate Lagrange multiplier for the corresponding *equality* constraint in the canonicalized problem.

It is tedious but straightforward to show that this result holds for multiple inequality constraints. It is also true that the nonlinearities created during the canonicalization process may be manipulated further without affecting the applicability of the above result—as long as those manipulations preserve equivalency. In other words, the Lagrange information for the nonlinearities themselves is effectively irrelevant to dual recovery, and therefore need not be computed or preserved by the numerical algorithm.

4.3 Numerical solution

A variety of methods that can be employed to solve convex programs, including (but not limited to) barrier, primal-dual, and cutting-plane methods. Each can be adapted to solve DCPs in canonical form (4.2). In this section, we provide a brief overview of these three aforementioned methods. Our purpose is *not* to describe them in detail; such was not a focus of our research. But we do wish to provide just enough background to introduce the *key computations* for each method—that is, the computations that require the vast majority of time and memory to perform. Examining these computations will allow us to answer two important questions:

- Under what conditions can a given algorithm solve a DCP? That is, what are the *computational requirements* that each method imposes on the DCPs it solves? Each method must be able to perform certain calculations involving the functions or sets present in the DCP (values, derivatives, cutting planes, *etc.*), and those calculations must be present in the atom library.

- How can the *problem structure* present in DCPs be exploited to achieve the best performance from these algorithms? Because DCPs are mechanically expanded, they tend to be larger but more structured (*e.g.*, sparse) than in traditional nonlinear programming methodologies. Exploiting this structure is essential to retaining attractive performance, and may even enable additional performance gains to be achieved.

By examining these methods in this manner, we hope to demonstrate convincingly that disciplined convex programs can indeed be solved effectively, while avoiding detail that is tangential to the focus of this dissertation.

4.3.1 Barrier methods

Suppose that the set S in $\mathcal{P}_{\text{DCP}}$ (4.2) has a non-empty interior. A barrier method represents this set by a convex, twice differentiable *barrier function* $\phi : \mathbb{R}^n \to (\mathbb{R} \cup +\infty)$, so named because it is finite on $\operatorname{Int} S$ and $+\infty$ outside of it. Using this barrier, a family of equality-constrained smooth convex minimization problems can be constructed, parameterized by a scalar $\mu > 0$:

$$\begin{array}{ll} \text{minimize} & c^T x + d + \mu\phi(x) \\ \text{subject to} & Ax = b. \end{array} \tag{4.75}$$

If $\mathcal{P}_{\text{DCP}}$ is strictly feasible and bounded below, then (4.75) has a unique solution $x^*(\mu) \in \operatorname{Int} S$ for each value of $\mu > 0$; and as $\mu \to 0$, $x^*(\mu)$ approaches the solution to the original problem: that is,

$$c^T x^*(\mu) + d \leq p^* + \vartheta\mu, \tag{4.76}$$

where p^* is the optimal value of $\mathcal{P}_{\text{DCP}}$, and $\vartheta > 0$ is a known positive constant called the *parameter* of the barrier. A barrier method tracks this trajectory of minimizers $x^*(\mu)$ while reducing μ in a controlled fashion. For example:

A barrier method [29]. Given $x \in \operatorname{Int} S$, $\mu > 0$, $\kappa > 1$, and $\epsilon > 0$:

1. Using x as the initial point, solve (4.75) using Newton's method, yielding $x^*(\mu)$.
2. Update $x \leftarrow x^*(\mu)$.
3. If $\vartheta\mu \leq \epsilon$, quit.

4. Otherwise, set $\mu \leftarrow \mu/\kappa$ and repeat all steps.

We refer the reader to [124] for an exhaustive study of barrier methods. The requirement that $\mathcal{P}_{\text{DCP}}$ be strictly feasible and bounded below is a significant practical limitation; fortunately, developments in *self-dual embedding* techniques rectify this problem at negligible additional cost [178].

Barrier methods can, in theory, solve any DCP; the challenge is determining an appropriate barrier. Each of the five types of nonlinearities discussed in §4.1 admits a barrier $\phi_k : \mathbb{R}^{n_k} \rightarrow (\mathbb{R} \cup +\infty)$ under certain conditions:

- $S_k = \mathbb{R}^{n_k}$:

$$\phi_k(x_{(k)}) \triangleq 0 \qquad \vartheta_k = 0. \tag{4.77}$$

- $S_k = \mathbb{R}^{n_i}_+$:

$$\phi_k(x_{(k)}) \triangleq \begin{cases} -\sum_{i=1}^{n_k} \log x_{(k),i} & \min_i x_{(k),i} > 0 \\ +\infty & \text{otherwise} \end{cases} \qquad \vartheta_k \triangleq n_k. \tag{4.78}$$

- $S_k = \operatorname{epi} f$: if f is convex and twice differentiable, and those derivatives are supplied in the function's implementation, then

$$\phi_k([\begin{smallmatrix} u \\ v \end{smallmatrix}]) \triangleq \begin{cases} -\log(v - f(u)) & f(u) < v \\ +\infty & f(u) \geq v \end{cases} \qquad \vartheta_k \triangleq 1. \tag{4.79}$$

- $S_k = \operatorname{hypo} g$: if g is concave and twice differentiable, and those derivatives are be supplied in the function's implementation, then

$$\phi_k([\begin{smallmatrix} u \\ v \end{smallmatrix}]) \triangleq \begin{cases} -\log(g(u) - v) & g(u) > v \\ +\infty & g(u) \leq v \end{cases} \qquad \vartheta_k \triangleq 1. \tag{4.80}$$

- S_k is a set atom: if the set S_k is convex and has a non-empty interior, then a barrier function ϕ_k with a known parameter ϑ_k may be supplied in the set's implementation.

For a set $S \triangleq S_1 \times S_2 \times \cdots \times S_L$ whose components S_k meet the above criteria, a barrier function $\phi : \mathbb{R}^n \rightarrow (\mathbb{R} \cup +\infty)$ and accompanying parameter ϑ can be constructed as follows:

$$\phi(x) \triangleq \phi_1(x_{(1)}) + \phi_2(x_{(2)}) + \cdots + \phi_L(x_{(L)}), \quad \vartheta \triangleq \vartheta_1 + \vartheta_2 + \cdots + \vartheta_L. \tag{4.81}$$

We say that a DCP in canonical form for which such a barrier exists is *barrier-solvable.*

The more important question, of course, is: under what conditions is a *general* DCP barrier-solvable? The short answer is that a DCP is barrier-solvable if it can be canonicalized into barrier-solvable form; but an *a priori* determination is sought here. This can be accomplished by examining the function and set atoms *before* canonicalization to see if they meet the right criteria. If all of the functions are twice differentiable and all of the sets have predefined barrier functions, then we may affirmatively conclude that the problem is barrier-solvable. For other functions and sets, we must examine their graph implementations; and because graph implementations are DCPs themselves, a precise definition of barrier-solvability is actually recursive:

> *Barrier-solvable DCPs.* A DCP is barrier-solvable if all of the function and set atoms that it employs are *barrier-compatible*:
>
> - A function atom is barrier-compatible if:
>   - it is affine;
>   - it is convex or concave and twice differentiable, and its implementation makes those derivatives available; or
>   - it has a graph implementation that is itself a barrier-solvable DCP.
> - A set atom is barrier-compatible if:
>   - its implementation supplies a barrier function and its accompanying parameter, implying that the set is convex and has a non-empty interior; or
>   - it has a graph implementation that is itself a barrier-solvable DCP.

It is important to note that these rules are not statements of theoretical existence; after all, we know that barrier methods are universal [124]. Instead, they serve as implementation specifications for the atom library. No atom need be excluded from use in a barrier method if its author is willing to fully implement it. Specifically, nondifferentiable functions and sets with empty interiors can be supported by giving them barrier-solvable graph implementations.

The choice of barrier function for a given set is not unique. For example, the function

$$\bar{\phi}_k(x_{(k)}) : \mathbb{R}^n \to (\mathbb{R} \cup +\infty), \quad \bar{\phi}_k(x_{(k)}) \triangleq \begin{cases} \sum_{i=1}^{n_k} x_{(k),i}^{-1} & \min_i x_{(k),i} > 0 \\ +\infty & \text{otherwise} \end{cases} \tag{4.82}$$

is another valid barrier function for the nonnegative orthant $S_k \in \mathbb{R}^n_+$. But this barrier will exhibit different performance in practice, and the original (4.78) is almost always preferred. It is the responsibility of the authors of atoms to insure that any barrier functions they supply for set atoms provide good practical performance. In fact, it is entirely possible that the barriers constructed for smooth functions via (4.79) or (4.80) are not the best choice, so replacements can be proposed using graph implementations. One desirable property of barrier functions is *self-concordance* [124], which can be used to obtain provable convergence and performance bounds for many barrier methods. The universality of barrier methods is not compromised if self-concordance is enforced.

The key calculation in the barrier method described above is the determination of a Newton search direction for (4.75), which involves the construction and solution of the linear system

$$\begin{bmatrix} \nabla^2\phi(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ y \end{bmatrix} = -\begin{bmatrix} r_d \\ r_p \end{bmatrix}, \qquad \begin{array}{l} r_d \triangleq \mu^{-1}c + \nabla\phi(x) \\ r_p \triangleq Ax - b. \end{array} \tag{4.83}$$

A linear system with this structure is called an *augmented system* and occurs often in interior-point methods for convex programming. It is symmetric but not positive definite, so direct solution requires an LDL^T factorization, where L is lower triangular and D is block diagonal with blocks of order 1 or 2. The Cartesian structure of S produces a *block-diagonal* structure in the Hessian matrix $\nabla^2\phi(x)$, as revealed by examining the equivalent linear system for the partitioned form (4.6):

$$\begin{bmatrix} \nabla^2\phi_1(x_{(1)}) & & & & A_{(1)}^T \\ & \nabla^2\phi_2(x_{(2)}) & & & A_{(2)}^T \\ & & \ddots & & \vdots \\ & & & \nabla^2\phi_L(x_{(L)}) & A_{(L)}^T \\ A_{(1)} & A_{(2)} & \dots & A_{(L)} & \end{bmatrix} \begin{bmatrix} \Delta x_{(1)} \\ \Delta x_{(2)} \\ \vdots \\ \Delta x_{(L)} \\ \bar{y} \end{bmatrix} = - \begin{bmatrix} r_{d,1} \\ r_{d,2} \\ \dots \\ r_{d,L} \\ r_p \end{bmatrix} \tag{4.84}$$
$$r_{d,k} \triangleq \mu^{-1}c_{(k)} + \nabla\phi_k(x_{(k)}), \quad k = 1,2,\dots,L$$

This block diagonal structure, as well as the sparsity of A, can be exploited to improve the performance of the LDL^T factorization.

Another kind of structure often occurs in barrier methods, and can be exploited to further improve performance. The Hessian $\nabla^2\phi(x)$ often obtains the form

$$\nabla^2\phi(x) = P + RQ^{-1}R^T \tag{4.85}$$

where P and Q have far fewer nonzeros than $\nabla^2\phi(x)$, and R has far fewer columns than rows. For example, the barrier for the epigraph of the Euclidean norm,

$$\phi_{\ell_2} : \mathbb{R}^n \times \mathbb{R} \to (\mathbb{R} \cup +\infty), \quad \phi_{\ell_2}(u,v) \triangleq \begin{cases} -\log(v^2 - u^Tu) & \|u\|_2 < v \\ +\infty & \|u\|_2 \geq v, \end{cases} \tag{4.86}$$

has a Hessian with diagonal-plus-rank-one structure:

$$\nabla^2\phi_{\ell_2}(u,v) \triangleq \frac{2}{t^2 - y^Ty}\begin{bmatrix} I & 0 \\ 0 & -1 \end{bmatrix} - \frac{2}{(t^2-y^Ty)^2}\begin{bmatrix} y \\ -t \end{bmatrix}\begin{bmatrix} y & -t \end{bmatrix}. \tag{4.87}$$

It is not difficult to show that this kind of structure can be used to create a new linear system

$$\begin{bmatrix} P & A^T & R \\ A & 0 & 0 \\ R^T & 0 & -Q \end{bmatrix} \begin{bmatrix} \Delta x \\ \bar{y} \\ z \end{bmatrix} = - \begin{bmatrix} t^{-1}c + \nabla\phi(x) \\ Ax - b \\ 0 \end{bmatrix}, \tag{4.88}$$

which is equivalent to (4.83), and is still symmetric indefinite, but potentially significantly more sparse. For example, in the case of ϕ_{ℓ_2}, the unmodified $(n+1)\times(n+1)$ Hessian $\nabla^2\phi_{\ell_2}$ is fully dense, but its expanded version has only $3n+4$ nonzeros. This structure can easily be exploited on a block-by-block basis if Cartesian/block-diagonal structure is present as well. Disciplined convex programming provides a convenient platform to support this type of performance optimization, because a decomposition of the form (4.85) can be included within the atom library for any barrier that might benefit from it.

In many contexts—including the solution of LPs in standard form—the $(1,1)$ block of an augmented system can be assumed to be positive definite. In such cases, it is common to convert the system to so-called *normal equations* form, which for (4.83) would yield

$$(A^T \nabla^2 \phi(x)^{-1} A) y = r_p - A \nabla^2 \phi(x)^{-1} r_d, \qquad \Delta x = -\nabla^2 \phi(x)^{-1}(r_d + A^T y). \tag{4.89}$$

This form is usually preferred for performance reasons. However, for DCPs, the Hessian $\nabla^2\phi(x)$ cannot be assumed to be positive definite, in part because the diagonal blocks corresponding to free variables will be identically zero. (Nothing we have specified here requires that other blocks be positive definite, but that condition can be enforced without unduly limiting generality.) The "problem" of free variables is not unique to disciplined convex programming, and a variety of other techniques have been proposed to "handle" them—though some are better suited for interior-point methods than others [113]. Still, despite this common preference for normal equations, supporting symmetric indefinite solvers in disciplined convex programming is likely to yield the greatest gains in terms of exploiting structure.

4.3.2 Primal-dual methods

Primal-dual methods solve smooth convex programs by directly searching for solutions to the problem's Karush-Kuhn-Tucker (KKT) conditions. Let us examine how one might be applied to our example problem (4.7). The KKT conditions for (4.7) are

$$\begin{array}{ll} c_x + A_x^T y = 0 & A_x x + A_z z + A_u u + A_v v = b \\ c_z + A_z^T y = s & ZS = \mu I \\ c_u + A_u^T y = -G^T \lambda & F\Lambda = \mu I \\ c_v + A_v^T y = \lambda & \\ \multicolumn{2}{c}{z, s, \lambda \geq 0 \quad f_k(u_{(k)}) \leq v_k \quad k = 1, 2, \dots, N} \end{array} \tag{4.90}$$

where F, Λ, Z, S, and G are defined as follows:

$$\begin{gathered} F = \mathbf{diag}(v_1 - f_1(u_{(1)}), v_2 - f_2(u_{(2)}), \dots, v_N - f_N(u_{(N)})) \\ \Lambda = \mathbf{diag}(\lambda) \quad Z \triangleq \mathbf{diag}(z) \quad S \triangleq \mathbf{diag}(S) \\ G^T \triangleq \begin{bmatrix} \nabla f_1(u_{(1)}) & & & \\ & \nabla f_2(u_{(2)}) & & \\ & & \ddots & \\ & & & \nabla f_N(u_{(N)}) \end{bmatrix} \implies G^T\lambda = \begin{bmatrix} \lambda_1 \nabla f_1(u_{(1)}) \\ \lambda_2 \nabla f_2(u_{(2)}) \\ \vdots \\ \lambda_N \nabla f_N(u_{(N)}) \end{bmatrix}. \end{gathered} \tag{4.91}$$

The exact KKT conditions result when $\mu = 0$; but for performance reasons, primal-dual methods begin with $\mu > 0$ and drive it to zero as the algorithm proceeds. For example:

> *A primal-dual method [29].* Given $\kappa > 1$, $\epsilon > 0$, and initial primal (x, z, u, v) and dual (y, s, λ) points satisfying
>
> $$z, s, \lambda > 0 \quad f_k(u_{(k)}) < v_{(k)} \quad k = 1, 2, \dots, N, \tag{4.92}$$
>
> 1. Set $\mu \leftarrow \hat{\eta}(x, z, u, v, y, s, \lambda)/(\kappa(n_z + N))$.
> 2. Take a single controlled Newton step toward the solution of (4.90).
> 3. If the equality constraints are satisfied and $\eta(x, z, u, v, y, s, \lambda) \leq \epsilon$, quit.
> 4. Otherwise, repeat all steps.

A number of details are omitted here, such as how to determine the step size for the Newton step. But as stated, this simple algorithm captures the key calculations. For a more complete introduction to primal-dual methods, see [29]. The above algorithm assumes that the problem is strictly feasible and bounded below, but more advanced primal-dual algorithms are not limited in this manner [5].

Because primal-dual methods apply only to smooth problems, DCPs in canonical form must satisfy two conditions: all epigraph and hypograph nonlinearities must involve twice differentiable functions; and no set nonlinearities may be used. A DCP is *primal-dual-solvable* if it can be canonicalized to a form that satisfies these conditions. A precise recursive definition is given here:

> *Primal-dual solvable* DCPs. A DCP is *primal-dual-solvable* if all of the functions and set atoms that it employs are *primal-dual-compatible.*
>
> - A function atom is primal-dual compatible if:
>   - it is affine;
>   - it is convex or concave and twice differentiable, and its implementation makes those derivatives available; or
>   - it has a graph implementation that is itself a primal-dual-solvable DCP.
> - A set atom is primal-dual compatible if
>   - it has a graph implementation that is itself a primal-dual-solvable DCP.

In short, a DCP is primal-dual solvable only if every atom, after full graph expansion, has been described using linear equations, linear inequalities, and smooth nonlinear inequalities.

The key computation in a primal-dual method is the determination of the Newton search direction for the KKT conditions, which involves the construction and solution of a symmetric indefinite linear system. The structure of the DCP canonical form is directly reflected in that linear system,

$$\begin{bmatrix} & & & & & & A_x^T \\ & & & & & -I & A_z^T \\ & & H & & G^T & & A_u^T \\ & & & & -I & & A_v^T \\ & & G & -I & -\Lambda^{-1}F & & \\ & -I & & & & -Z^{-1}S & \\ A_x & A_z & A_u & A_v & & & \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta z \\ \Delta u \\ \Delta v \\ \Delta\lambda \\ \Delta s \\ \Delta y \end{bmatrix} = - \begin{bmatrix} c_x + A_x^T y \\ c_z + A_z^T y - s \\ c_u + A_u^T y + G^T \lambda \\ c_v + A_v^T y - \lambda \\ \mu(\Lambda^{-1} - F)\vec{1} \\ \mu(Z^{-1} - S)\vec{1} \\ A_x x + A_z z + A_u u + A_v v - b \end{bmatrix}, \tag{4.93}$$

where F, Λ, Z, S, and G are defined in (4.91) above, and H is given by

$$H \triangleq \begin{bmatrix} \lambda_1 \nabla^2 f_1(u_{(1)}) & & & \\ & \lambda_2 \nabla^2 f_2(u_{(2)}) & & \\ & & \ddots & \\ & & & \lambda_N \nabla^2 f_N(u_{(N)}) \end{bmatrix}. \tag{4.94}$$

This system is obviously quite sparse, particularly if one accounts for the block structure of G and H, caused by the independence of the nonlinearities. It is important to note that while the diagonal blocks $-\Lambda^{-1}F$ and $-Z^{-1}S$ are nonsingular, there is no similar guarantee about H; nor is it practical to require it. For example, the function $f(x) \triangleq -\log \sum_{k=1}^n e^{x_k}$, which occurs quite often in geometric programming, has a Hessian

$$\nabla^2 f(x) = \mathbf{diag}(q) - qq^T, \quad q \triangleq \frac{1}{\sum_{k=1}^n e^{x_k}} \begin{bmatrix} e^{x_1} & e^{x_2} & \dots & e^{x_n} \end{bmatrix}^T, \tag{4.95}$$

which is not strictly convex—for example, $\nabla^2 f(x)\vec{1} = 0$. Therefore, a combination of block elimination and sparse symmetric indefinite factorization is likely the best approach to solving (4.93).

4.3.3 Cutting-plane methods

Cutting plane methods take a completely different approach than the previous two options, and are notable for their native ability to support nondifferentiable functions. A cutting plane method represents the set $S \subseteq \mathbb{R}^n$ with an *oracle*: a computational construct which, when supplied a point $x \in \mathbb{R}^n$, returns one of two results:

- if $x \notin S$, a cutting plane separating x and S: that is, a pair $(q, r) \in \mathbb{R}^n \times \mathbb{R}$ satisfying

$$q^T x \leq r \qquad q^T \bar{x} \geq r \quad \forall \bar{x} \in S. \tag{4.96}$$

- if $x \in S$, some sort of indication of this fact—say, by returning $(q, r) = (0, 0)$.

A cutting plane method to solve $\mathcal{P}_{\mathrm{DCP}}$ (4.2) feeds a sequence of *query points* $x_{(i)} \in \mathbb{R}^n$, $i = 1, 2, \dots, M$, all satisfying $Ax^{(i)} = b$, to the oracle Φ. Those that fall within S are used to refine an upper bound on the optimal value p^* of $\mathcal{P}_{\mathrm{DCP}}$:

$$p_u \triangleq \min \left\{ c^T x_{(j)} + d \;\mid\; x_{(j)} \in S,\; j = 1, 2, \dots, M \right\} \geq p^*. \tag{4.97}$$

Those that fall outside of S return cutting planes $(q_{(i)}, r_i)$, which are added to an LP relaxation of $\mathcal{P}_{\mathrm{DCP}}$ used to generate lower bounds on p^*:

$$p_l \triangleq \inf \left\{ c^T \bar{x} + d \;\mid\; A\bar{x} = b,\; q_{(j)}^T \bar{x} \geq r_j,\; j = 1, 2, \dots, M \right\} \leq p^*. \tag{4.98}$$

If the sequence of query points are chosen carefully, the upper and lower bounds will converge to the optimal value as $M \to \infty$. The following is a highly simplified version of such a method:

A hypothetical cutting-plane method. Given an oracle Φ for the set S and $\epsilon > 0$, set $(p_l, p_u) \leftarrow (-\infty, +\infty)$ and proceed:

1. Generate a new query point x satisfying $Ax = b$.
2. Consult the oracle concerning x, yielding a pair $\Phi(x) \triangleq (q, r)$.
3. If $x \notin S$ $(q \neq 0)$, add the inequality $q^T x \geq r$ to the LP approximation and update the lower bound p_l (4.98).
4. If $x \in S$ $(q = 0)$, update the upper bound p_u (4.97).
5. *Stop* if $p_u - p_l \leq \epsilon$.
6. Otherwise, repeat from step 1.

The critical step in this method is the generation of a query point. There are several ways to accomplish this in a manner that guarantees convergence. One such choice, employed by the *analytic center cutting-plane method* (ACCPM) [135], is to select the *analytic center* x^* of the set

$$\left\{ \bar{x} \in \mathbb{R}^n \;\mid\; c^T \bar{x} + d \leq p_u,\; A\bar{x} = b,\; q_{(j)}^T \bar{x} \leq r_j,\; j = 1, 2, \dots, M \right\}, \tag{4.99}$$

which is the optimal point of the equality-constrained smooth convex minimization,

$$\begin{array}{ll} \text{minimize} & -\log(p_u - c^T x - d) - \sum_{j=1}^M \log(q_{(j)}^T x - r_j) \\ \text{subject to} & Ax = b. \end{array} \tag{4.100}$$

If this point exists (and in practice, steps are taken to ensure that it does), it is unique. It is not difficult to deduce from (4.100) that $c^T x^* + d < p_u$; so if $x^* \in S$, the analytic center provides a strict reduction in the upper bound p_u. But in fact, it can also be shown that the quantity

$$\bar{p}_l \triangleq (M+1)(c^T x^* + d) - M p_u \tag{4.101}$$

is a lower bound on (4.98), and therefore for the original problem (4.2) as well—whether or not $x^* \notin S$. This lower bound can be used instead of the exact bound (4.98), and convergence is still assured. For more information about ACCPM, consult [135].

As with barrier methods, cutting-plane methods can, in theory, solve any DCP; the challenge is to construct an oracle Φ. Each of the five types of nonlinearities discussed in §4.1 admits an oracle $\Phi_k : \mathbb{R}^{n_k} \to \mathbb{R}^{n_k} \times \mathbb{R}$ under certain limited circumstances:

- $S_k = \mathbb{R}^{n_k}$: since $x_{(k)} \in S_k$ always,

$$\Phi_k(x_{(k)}) \equiv (0, 0). \tag{4.102}$$

- $S_k = \mathbb{R}^{n_k}_+$:

$$\Phi_k(x_{(k)}) \triangleq \begin{cases} (e_j, 0) & x_{(k)} \not\geq 0 \\ (0, 0) & x_{(k)} \geq 0 \end{cases} \quad j \triangleq \arg\min_j e_j^T x_{(k)}, \tag{4.103}$$

  where e_j is the jth column of the $n_k \times n_k$ identity vector. In other words, this oracle simply generates inequalities of the form $x_{(k,j)} \geq 0$.

- $S_k = \operatorname{epi} f$: if f must be convex and continuous, and a gradient or subgradient calculation has been provided in the implementation, then

$$\Phi_k(u, v) = \begin{cases} (-w, 1) & f(u) > v \\ (0, 0) & f(u) \leq v \end{cases} \quad w \in \partial f(u), \tag{4.104}$$

  where w is a subgradient of f at u.

- $S_k = \operatorname{hypo} g$: if g is concave and continuous, and a gradient or supergradient calculation has been provided in the implementation, then

$$\Phi_k(u, v) = \begin{cases} (w, -1) & g(u) < v \\ (0, 0) & g(u) \geq v \end{cases} \quad w \in -\partial(-g(u)). \tag{4.105}$$

  where w is a supergradient of g at u.

- S_k is a set atom: if S_k has a non-empty interior, then an oracle may be supplied in the set's implementation section.

For a set $S \triangleq S_1 \times S_2 \times \cdots \times S_L$ whose components S_k meet the above criteria, an oracle $\Phi : \mathbb{R}^n \to \mathbb{R}^n \times \mathbb{R}$ can be constructed as follows: if $x \notin S$, then

$$\Phi(x) \triangleq \left(\begin{bmatrix} 0_{n_1 \times 1} \\ q \\ 0_{n_2 \times 1} \end{bmatrix}, r \right) \qquad \begin{array}{c} \bar{k} = \inf \left\{ \, k \;\middle|\; x_{(k)} \notin S_k \, \right\}, \quad (q, r) \triangleq \Phi_{\bar{k}}(x_{(\bar{k})}) \\ n_1 \triangleq \sum_{l=1}^{\bar{k}-1} n_l, \quad n_2 \triangleq \sum_{l=\bar{k}+1}^{L} n_l, \quad . \end{array} \tag{4.106}$$

In short, this construction recognizes that a cutting plane for any one set S_k serves as a cutting plane for the entire set S. So it selects the first of the sets for which $x_{(k)} \notin S_k$, and constructs the cutting plane using that set's oracle alone.

We say that a DCP in canonical form for which such an oracle exists is *cutting-plane-solvable.* The answer to the question of which general DCPs are cutting-plane-solvable is similar to those given for other methods:

> *Cutting-plane-solvable DCPs.* A DCP is cutting-plane-solvable if all of the functions and sets that it employs are *cutting-plane-compatible*:
>
> - A function atom is cutting-plane-compatible if:
>   - it is affine;
>   - it is convex and continuous, and its implementation provides a gradient or subgradient calculation;
>   - it is concave and continuous, and its implementation provides a gradient or supergradient calculation;
>   - it has a graph implementation that is itself a cutting-plane-solvable DCP.
> - A set atom is cutting-plane-compatible if:
>   - it has a non-empty interior, and its implementation provides an oracle;
>   - it has a graph implementation that is itself a cutting-plane-solvable DCP.

The key computation in ACCPM is the determination of the analytic center (4.100) using an equality-constrained Newton algorithm. Each Newton step involves the construction and solution of a linear system of the form

$$\begin{bmatrix} \tilde{p}^2 cc^T + W & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ y \end{bmatrix} = - \begin{bmatrix} \tilde{p}c + Q\tilde{R}\vec{1} \\ Ax - b \end{bmatrix}, \tag{4.107}$$

where we have defined

$$\begin{gathered} Q \triangleq \begin{bmatrix} q_{(1)} & q_{(2)} & \dots & q_{(M)} \end{bmatrix} \quad r \triangleq \begin{bmatrix} r_1 & r_2 & \dots & r_M \end{bmatrix} \\ \tilde{p} \triangleq (p_u - c^T x - d)^{-1} \quad \tilde{R} \triangleq \mathbf{diag}(Q^T x - r)^{-1} \quad W \triangleq QRQ^T. \end{gathered} \tag{4.108}$$

To expose the problem structure, let us first eliminate the rank-one addition p^2cc^T from the $(1,1)$ block, producing the slightly larger system

$$\begin{bmatrix} W & c & A^T \\ c^T & -\tilde{p}^{-2} & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ z \\ y \end{bmatrix} = - \begin{bmatrix} Q\tilde{R}\vec{1} \\ \tilde{p}^{-1} \\ Ax - b \end{bmatrix}. \tag{4.109}$$

Now as (4.106) shows, the vectors $q_{(k)}$ are non-zero only for the rows corresponding to a single nonlinearity. This means that the matrix $W \triangleq Q\tilde{R}Q^T$ will be *block diagonal.* So if we separate the cutting planes into bins corresponding to the nonlinearity S_k from which they came,

$$\bar{S}_k \triangleq \left\{ x \in \mathbb{R}^{n_k} \mid q_{(k,j)}^T x \geq r_{k,j},\ j = 1, 2, \dots, N_j \right\}, \tag{4.110}$$

and we define

$$\begin{gathered} Q_{(k)} \triangleq \begin{bmatrix} q_{(k,1)} & q_{(k,2)} & \dots & q_{(k,M_k)} \end{bmatrix} \quad r_{(k)} \triangleq \begin{bmatrix} r_{k,1} & r_{k,2} & \dots & r_{k,M_k} \end{bmatrix} \\ \tilde{R}_{(k)} \triangleq \mathbf{diag}(Q_{(k)}^T x_{(k)} - r_{(k)})^{-1} \quad W_{(k)} \triangleq Q_{(k)}^T \tilde{R}_{(k)} Q_{(k)} \end{gathered} \tag{4.111}$$

for each $k = 1, 2, \dots, L$, then the full structure of (4.109) is revealed:

$$\begin{bmatrix} W_{(1)} & & & & c_{(1)} & A_{(1)}^T \\ & W_{(2)} & & & c_{(2)} & A_{(2)}^T \\ & & \ddots & & \vdots & \vdots \\ & & & W_{(L)} & c_{(L)} & A_{(L)}^T \\ c_{(1)}^T & c_{(2)}^T & \dots & c_{(L)}^T & -\tilde{p}^{-2} & \\ A_{(1)} & A_{(2)} & \dots & A_{(L)} & & \end{bmatrix} \begin{bmatrix} \Delta x_{(1)} \\ \Delta x_{(2)} \\ \vdots \\ \Delta x_{(L)} \\ z \\ y \end{bmatrix} = - \begin{bmatrix} Q_{(1)} \tilde{R}_{(1)} \vec{1} \\ Q_{(2)} \tilde{R}_{(2)} \vec{1} \\ \vdots \\ Q_{(L)} \tilde{R}_{(L)} \vec{1} \\ \tilde{p}^{-1} \\ Ax - b \end{bmatrix}. \tag{4.112}$$

The arrow structure is similar to that seen for barrier methods (4.88), although the blocks $W_{(k)}$ in this case may not be sparse. In practice, steps are generally taken to ensure that they are positive definite, so a normal equation reduction similar to (4.89) can be considered. This structure also suggests an opportunity to optimize the data handling capabilities of the underlying cutting-plane method, because the cutting planes for each nonlinearity S_k can be generated and collected separately—perhaps even by separate processing units.

4.4 Conclusions

The conventions and structure imposed by the disciplined convex programming methodology allow the solution process to be automated, including the conversion to solvable form and the recovery of dual information. The canonical form is suitable for solution by any common method for solving traditional convex optimization problems. And yet, the specific structure obtained by DCPs—in particular, the Cartesian structure of the nonlinearities, and the likely sparsity of the equality constraints—can be exploited by code specifically designed for DCPs to improve performance.

In the author's opinion, barrier methods (especially using self-dual embeddings) and disciplined convex programming are ideally suited to one another, in that disciplined convex programming allows the full potential of barrier methods to be realized in practice. In addition, the arrow structure of (4.84) and (4.112) suggest that an interesting topic for future work would be the design of distributed methods or *decomposition methods* to solve DCPs. The Cartesian structure of the nonlinearities provides a natural division of responsibility for such methods.

Chapter 5

Conclusion

We have introduced a new methodology for convex optimization called *disciplined convex programming.* This methodology requires that certain conventions be followed when constructing convex programs. The conventions are easy to learn, do not limit generality, and formalize the practices of those who already use convex optimization. In return for compliance, the methodology offers a number of benefits, including:

- *Reliable verification.* The task of determining if an arbitrary NLP is convex is intractable; the task of determining if it is a valid DCP is fast and straightforward. The verification process can even result in a determination of *conditional* compliance, in which the problem is deemed valid only if certain additional assertions on the parameters are satisfied.

- *Conversion to canonical form.* Once a problem has been verified as a DCP, conversion to canonical form (4.2) can take place quickly and automatically. The canonicalization transformations can be confidently applied without concern for equivalency, because the verification process confirms equivalence it in advance.

- *Full support for nondifferentiable functions.* Graph implementations provide support for nondifferentiable functions without the loss of performance and reliability typically associated with them. A graph implementation effectively encapsulates an equivalency-preserving transformation that removes the offending instances of nondifferentiability from the problem. As a result, any solver designed for general convex programming is enhanced by this support.

- *Support for a variety of numerical methods.* Primal-dual, barrier, and cutting-plane methods can all be employed to solve DCPs. Others are probably feasible as well. The structure of the DCP canonical form lends itself to the study of decomposition methods.

- *Significant problem structure.* The sparse and block structure of the DCP canonical form can be exploited to improve the performance of numerical solvers. To be fair, some of that structure is due to the canonicalization process, and *must* be exploited if DCPs are to be solved with performance comparable to other approaches. But DCPs also exhibit additional structure typically not revealed in these other approaches.

- *Collaboration.* The atom library provides a natural means for advanced users to create libraries of functions, sets, and models that can be shared with modelers within an organization or with the entire user community. Graph implementations allow a much wider variety of functions and sets to be defined than is possible in a typical value/gradient/Hessian approach.

5.1 Motivation, revisited

In §1.1 and Chapter 2, we presented norm minimization problems as a means to illustrate the practical complexities of convex programming. In §1.1.7, we suggested that disciplined convex programming could be used to simplify the task of solving these problems, and presented example models in the `cvx` modeling language. As stated there, the examples relied on features of the modeling language that were not yet implemented. Specifically, those models employed vector and matrix variables and parameters, while the current version of `cvx` supports only scalars (see §5.2.1).

Despite this limitation, the current version of `cvx` can still solve these problems and thereby demonstrate many of the promised benefits of disciplined convex programming. To show this, we considered a family of norm minimization problems of the form

$$\begin{array}{ll} \text{minimize} & \|Ax-b\| \\ \text{subject to} & -\vec{1} \leq x \leq \vec{1}. \end{array} \tag{5.1}$$

similar to those studied in Chapter 2. We solved the problem for the ℓ_2, ℓ_1, ℓ_∞, $\ell_{3.5}$, and largest-L norms. The dimensions of the parameters $A \in \mathbb{R}^{8\times 5}$ and $b \in \mathbb{R}^8$ were fixed, and their values were repeatedly chosen at random until a combination was found that produced "interesting" results—that is, results that exercised some, but not all, of the bound constraints $-\vec{1} \leq x \leq \vec{1}$ in each case. The final selections for A and b are provided in (A.2) of Appendix A.

The resulting `cvx` model, including the numerical data, is given in Figure 5.1. In that figure, the ℓ_2 norm is used; to select the other norms, the first line

```
minimize norm_2(
```

must simply be replaced with

```
minimize norm_1(
minimize norm_inf(
minimize norm_p( 3.5,
minimize norm_largest( 3,
```

for the ℓ_1, ℓ_∞, $\ell_{3.5}$, and largest-3 cases, respectively. This is the *only* change required of the model itself. The norms themselves have been defined in the external file `norms.cvx`, which is referenced

```
minimize norm_2(
    a11 * x1 + a12 * x2 + a13 * x3 + a14 * x4 + a15 * x5 - b1,
    a21 * x1 + a22 * x2 + a23 * x3 + a24 * x4 + a25 * x5 - b2,
    a31 * x1 + a32 * x2 + a33 * x3 + a34 * x4 + a35 * x5 - b3,
    a41 * x1 + a42 * x2 + a43 * x3 + a44 * x4 + a45 * x5 - b4,
    a51 * x1 + a52 * x2 + a53 * x3 + a54 * x4 + a55 * x5 - b5,
    a61 * x1 + a62 * x2 + a63 * x3 + a64 * x4 + a65 * x5 - b6,
    a71 * x1 + a72 * x2 + a73 * x3 + a74 * x4 + a75 * x5 - b7,
    a81 * x1 + a82 * x2 + a83 * x3 + a84 * x4 + a85 * x5 - b8
    );
subject to
    -1 <= x1 <= 1; -1 <= x2 <= 1; -1 <= x3 <= 1;
    -1 <= x4 <= 1; -1 <= x5 <= 1;
variables
    x1, x2, x3, x4, x5;
parameters
    a11, a12, a13, a14, a15, a21, a22, a23, a24, a25,
    a31, a32, a33, a34, a35, a41, a42, a43, a44, a45,
    a51, a52, a53, a54, a55, a61, a62, a63, a64, a65,
    a71, a72, a73, a74, a75, a81, a82, a83, a84, a85,
    b1, b2, b3, b4, b5, b6, b7, b8;
include "norms.cvx";
```

Figure 5.1: A norm minimization test problem for the cvx modeling framework.

```
function norm_largest( L, x1, x2, x3, x4, x5, x6, x7, x8 )
    convex( x1, x2, x3, x4, x5, x6, x7, x8 ) >= 0 if L in #[1,8] {
    epigraph := {
        variables
            xp1 >= 0, xm1 >= 0, v1 >= 0, xp2 >= 0, xm2 >= 0, v2 >= 0,
            xp3 >= 0, xm3 >= 0, v3 >= 0, xp4 >= 0, xm4 >= 0, v4 >= 0,
            xp5 >= 0, xm5 >= 0, v5 >= 0, xp6 >= 0, xm6 >= 0, v6 >= 0,
            xp7 >= 0, xm7 >= 0, v7 >= 0, xp8 >= 0, xm8 >= 0, v8 >= 0, q;
        norm_largest = v1 + v2 + v3 + v4 + v5 + v6 + v7 + v8 + L q;
        x1 = xp1 - xm1; xp1 + xm1 = q + v1;
        x2 = xp2 - xm2; xp2 + xm2 = q + v2;
        x3 = xp3 - xm3; xp3 + xm3 = q + v3;
        x4 = xp4 - xm4; xp4 + xm4 = q + v4;
        x5 = xp5 - xm5; xp5 + xm5 = q + v5;
        x6 = xp6 - xm6; xp6 + xm6 = q + v6;
        x7 = xp7 - xm7; xp7 + xm7 = q + v7;
        x8 = xp8 - xm8; xp8 + xm8 = q + v8;
    }
}
```

Figure 5.2: The entry in the file norms.cvx for the largest-L norm.

using the `include` command. Figure 5.2 provides an excerpt of this file, specifically the definition of the largest-L norm.

In each case, the problem was converted to canonical form and sent to a barrier method designed and coded by the author. Each required between 29-34 iterations to solve. The solver exploits both types of structure discussed in §4.3.1; however, because these problems are small, any performance improvements were not apparent. For the ℓ_2, ℓ_1, ℓ_∞, and largest-3 cases, the results were verified against solutions computed using the SeDuMi software package [148]; the problems were manually converted to the canonical form for that solver. For the $\ell_{3.5}$ case, we could make no such direct comparison, but we were able to confirm the correctness of the implementation of the `norm_p` function by running it for $p = 2.0$ and comparing the results to the `norm_2` case.

Clearly, the limitation to scalar variables causes both the model in Figure 5.1 and the function definition in Figure 5.2 to be more complicated than they would otherwise be. Nevertheless, the current `cvx` software demonstrates the feasibility of key principles of the disciplined convex programming approach, including the feasibility of graph implementations, ruleset verification, canonicalization, numerical solution, and exploitation of structure.

Shortly after the publication of this dissertation, we will make this software available, including the above example, for study and testing at the Web site `http://www.stanford.edu/~boyd/cvx/`. Appendix A includes a full sample of the `cvx` output, specifically for the $\ell_{3.5}$ case above. It is far too long to include here in the main text, as it includes output from each individual stage of the software, including the ruleset verifier, the canonicalizer, and the numerical solver.

5.2 Future work

Several areas of future work are suggested by this research. We provide a few ideas below, separated into three categories: improving the `cvx` framework (§5.2.1), extending disciplined convex programming (§5.2.2), and advanced applications (§5.2.3). Some will be pursued quite soon.

5.2.1 Improving the `cvx` framework

Vector and matrix support

As mentioned, the current version of the `cvx` modeling framework supports only scalar variables and parameters. As a result, models that involve vectors or matrices must be specified with separate variables and parameters declared for each element, as was done for the examples in §5.1. The scalar limitation also limits the effectiveness of the atom library, because it requires that different functions be declared for different input sizes. The current `cvx` implementation allows a single function declaration to handle argument lists of arbitrary length, but only for simple implementations—for graph implementations, a separate declaration is required for each input size. For example, a function `norm_1` implementing the ℓ_1 norm requires a separate declaration for each vector length.

The next revision of the `cvx` framework, which will be commenced soon after the publication of this dissertation, will provide support for vectors, matrices, and even three-dimensional array objects (to simplify the specification of linear matrix inequalities, for example).

Solvers

The current version of `cvx` employs a barrier method not unlike what is described in §4.3.1. A two-phase method has been adopted: the first phase searches for a strictly feasible point to use as the initial point for the second phase. It is generally reliable and performs well. However, this algorithm does not handle certain known scenarios, including problems that are feasible but not *strictly* feasible. If our goal is to simplify the use of convex programming for applications-oriented users, this limitation needs to be overcome, so an improved solver is needed. Our initial choice is to adapt the self-dual embedding work of Zhang [179] to solve DCPs. In addition, there are several solvers developed by others that we would like to connect to `cvx`, including MOSEK [118], LOQO [162], SNOPT [68], and ACCPM [135].

The particular structure of the DCP canonical form (4.6) provides challenges that traditional solvers must commonly address (*e.g.*, free variables), as well as unique opportunities for performance optimization. In fact, it may prove that general purpose solvers need some modification to fully exploit the structure of the DCP canonical form.

Code generation

`cvx` is a monolithic program that combines model parsing and verification, canonicalization, and numerical solution into a single program. But one of the potential benefits of disciplined convex programming is the promise of *code generation*. Given a disciplined convex programming model in the `cvx` language, a code generation system would execute the parsing, verification, and canonicalization steps as now; but would then create code for a *standalone* program or subroutine to complete the remaining tasks. This code would accept as input the numeric values of the model's parameters, and would provide as output the problem solution. This code could be included in an application without the need for the full `cvx` machinery.

One of the potential opportunities in code generation is the ability to deduce and exploit some of a problem's structure *statically*—that is, *before* the code generation process. For example, often the sparsity pattern of the Newton systems can be determined without knowledge of the actual numerical values of the parameters; in such cases, a symbolic factorization can be computed in advance and embedded into the code, providing considerable savings.

Barrier construction

In §4.3.1, we stated that the choice of barrier function for a given set is not unique, and that difference choices exhibit different theoretical and practical performance. It is the responsibility of the authors of atoms to insure that any supplied barrier functions achieve good practical performance. Thus of particular interest would be any method for the automated construction, verification, or analysis of barrier functions. Obviously, the work of Nesterov and Nemirovsky in [124] is central here.

5.2.2 Extending disciplined convex programming

Quadratic forms

The DCP ruleset does not permit the specification of polynomials, even simple quadratic forms such as `x^2 + 2 x y + y^2`. In §3.4, we claimed that this is less likely to be a hindrance for disciplined convex programming than for traditional nonlinear programming, because of the former's direct support for "quadratic-related" constructs such as Euclidean norms. We also showed how simple it is to circumvent the restriction.

Nevertheless, it seems reasonable for us to explore the relaxation of the DCP ruleset to allow quadratic polynomial expressions, particularly once vector and matrix support is added. For example, the prototypical product-free form (3.3) could be modified as follows:

$$a + b^T x + x^T Q x + \sum_{j=1}^{L} c_j f_j(arg_{j,1}, arg_{j,2}, \ldots, arg_{j,m_j}) \tag{5.2}$$

The sign rules would then be extended to require that Q be positive semidefinite, negative semidefinite, or identically zero for convex, concave, and affine expressions, respectively.

A suggestion has been made by Zhang (*e.g.*, [178]) to allow more general *sum of squares* (SOS) polynomials. A polynomial $p(x)$ of $x \in \mathbb{R}^n$ is an SOS if it can be expressed in the form $p(x) = \sum_{i=1}^m q_i^2(x)$, where each $q_i(x)$ is a polynomial. SOS polynomials have proven to be an interesting field of study that is closely related to semidefinite programming [136]. However, the addition of any new rule to the DCP ruleset must be weighed against the intent of disciplined convex programming; so the ruleset must remain simple to learn, apply, and verify. It is not clear that SOS polynomials meet this criteria; simpler quadratic forms such as (5.2), on the other hand, may.

Perspective transformations

A *perspective transformation* of a function $f : \mathbb{R}^n \to (\mathbb{R} \cup +\infty)$ yields a new function

$$\tilde{f} : (\mathbb{R}^n \times \mathbb{R}) \to (\mathbb{R} \cup +\infty), \quad \tilde{f}(x, y) \triangleq \begin{cases} y f(y^{-1} x) & y > 0 \\ +\infty & y \leq 0. \end{cases} \tag{5.3}$$

If f is convex, concave, or affine, then is $\tilde{f}$ is also convex, concave, or affine, respectively. Because the perspective transformation can be applied without exception, it is a good candidate for inclusion in the DCP ruleset.

The perspective transformation can be quite useful in disciplined convex programming. For example, consider the smooth quadratic function $f(x) \triangleq x^T x$, and its epigraph

$$\operatorname{epi} f \triangleq \left\{ (x, y) \;\middle|\; \|x\|_2^2 = x^T x \leq y \right\}. \tag{5.4}$$

This function allows us to bound the squared ℓ_2 norm of a vector; but often we wish to bound the non-squared norm. Performing a perspective transformation on f yields the result

$$\tilde{f}(x, y) = y f(y^{-1} x) = y^{-1}(x^T x) \tag{5.5}$$

if $y > 0$, and

$$\tilde{f}(x, y) \leq y \quad \Longleftrightarrow \quad y^{-1}(x^T x) \leq y \quad \Longleftrightarrow \quad \|x\|_2^2 \leq y^2. \tag{5.6}$$

Thus the smooth function $\tilde{f}$ can be used to represent the epigraph of the ℓ_2 norm as a smooth nonlinear inequality. (The $y = 0$ case is a technical detail that in practice may be ignored.)

Without the perspective transformation, the function $\tilde{f}$ must be added separately to the atom library; with it, the existing function f suffices. Thus the perspective transformation increases the expressiveness of the atom library. It is an advanced operation that model-oriented users may find difficult to use, but advanced users could employ it in the creation of new functions with graph implementations, which would benefit all users. On the other hand, adding the perspective transformation to the DCP would require a modification to the canonical form, and would complicate the canonicalization process. So the full consequences of such an addition remain to be explored.

Quasiconvexity

A function $f : \mathbb{R}^n \to (\mathbb{R} \cup +\infty)$ is *quasiconvex* if

$$f(\alpha x + (1 - \alpha) y) \leq \max\{f(x), f(y)\} \quad \forall x, y \in \mathbb{R}^n, \tag{5.7}$$

or, equivalently, if its *sublevel sets*

$$S_{f,\alpha} \triangleq \{\, x \mid f(x) \leq \alpha \,\} \tag{5.8}$$

are convex. Similarly, a function g is *quasiconcave* if $-g$ is quasiconvex; and a function h is *quasilinear* if it is both quasiconvex and quasiconcave. Not surprisingly, all convex functions are quasiconvex, but these classes are larger: for example, the concave function $f(x) = \log x$ is quasiconvex—in fact, it is quasilinear, as are all scalar monotonic functions.

A set of rules, consistent with the design goals of disciplined convex programming, can be constructed to govern the use of quasiconvex, quasiconcave, and quasilinear functions in convex programs. For example, constraints of the form

$$\textit{quasiconvex} \leq \textit{constant} \qquad \textit{quasiconcave} \geq \textit{constant} \qquad \textit{quasilinear} = \textit{constant} \tag{5.9}$$

```
function log( x ) concave quasiconvex nondecreasing {
    value    := x <= 0 ? -Inf : @log( x );
    gradient := 1 / x;
    Hessian  := 1 / ( x * x );
    epigraph := { x <= @exp( log ); }
    graph    := { x = @exp( log ); }
}
```

Figure 5.3: An example implementation of the logarithm function with support for quasiconvexity and quasilinearity.

constrain the variables in a convex manner—that is, the values of the variables that satisfy these constraints form a convex set. A number of arithmetic operations preserve quasiconvexity, including but not limited to the following:

- the maximum of quasiconvex expressions is quasiconvex;
- the product of a quasiconvex expression and a nonnegative constant is quasiconvex;
- the composition of a nondecreasing function and a quasiconvex function is quasiconvex; *etc.*

Interestingly, while maximums of quasiconvex functions are quasiconvex, *sums* of them are not (in general), and would therefore be forbidden by a "quasiconvex-capable" DCP ruleset.

Graph implementations provide an elegant means to represent quasiconvex, quasiconcave, and quasilinear functions. This is not surprising given that a sublevel set is just a slice of the epigraph, where the epigraph variable is held fixed. For example, suppose that a is a constant, and consider the constraints

$$\log(x) \geq a \qquad \log(x) \leq a \qquad \log(x) = a. \tag{5.10}$$

Of these, only the first is compliant with the current DCP ruleset, but all three constrain x in a convex manner. A simple transformation of the last two produces compliant results:

$$\log(x) \geq a \qquad x \leq \exp(a) \qquad x = \exp(a). \tag{5.11}$$

In `cvx`, we might represent these transformations in a graph implementation as shown in Figure 5.3. (The functions `@log` and `@exp` refer to the "native" C routines for the logarithm and exponential, respectively.) Note the use of the attribute `quasiconvex` to denote quasiconvexity. Similar keywords `quasiconcave` or `quasilinear` would also be available, but are unnecessary in this case, as those attributes can be deduced from those stated.

Quasiconvex objective functions require a bit more effort to support fully. The simplest case involves the composition of a monotonic function and a convex function. For example, consider the convex program

$$\begin{array}{ll} \text{minimize} & f(g(x)) \\ \text{subject to} & x \in C, \end{array} \tag{5.12}$$

where $f : \mathbb{R} \rightarrow (\mathbb{R} \cup +\infty)$ is nondecreasing but *not* convex, $g : \mathbb{R}^n \rightarrow (\mathbb{R} \cup +\infty)$ is convex, and the constraint $x \in C$ represents an arbitrary collection of valid DCP constraints. The composition rule stated above dictates that $f(g(x))$ is quasiconvex. In this case, it is not difficult to see that solutions to this problem can be obtained by solving a modified problem with f removed; *i.e.*,

$$\begin{array}{ll} \text{minimize} & g(x) \\ \text{subject to} & x \in C. \end{array} \tag{5.13}$$

The more general case is a bit more difficult: consider a problem

$$\begin{array}{ll} \text{minimize} & h(x) \\ \text{subject to} & x \in C \end{array} \tag{5.14}$$

where $h : \mathbb{R}^n \rightarrow (\mathbb{R} \cup +\infty)$ is quasiconvex. Using the common epigraph transformation to move the objective function into a constraint, *i.e.*,

$$\begin{array}{ll} \text{minimize} & y \\ \text{subject to} & x \in C \\ & h(x) \leq y, \end{array} \tag{5.15}$$

does not produce convex results, because $f(x) \leq y$ is not convex. However, given the quasiconvex extensions discussed above, and assuming that h has been given a graph implementation, (5.15) is convex *if y is fixed*, in which case (5.15) is a disciplined convex feasibility problem. We can therefore solve (5.14) using bisection, at each iteration computing a solution to (5.15) for a fixed value of y.

It seems, then, that the disciplined convex programming methodology would be genuinely improved by adding support for quasiconvexity.

Generalized convexity

In §1.2.2, we introduced the notion of cones—sets that are closed under nonnegative scaling. Given a closed, convex, solid ($\operatorname{Int} \mathcal{K} \neq \emptyset$) cone $\mathcal{K}$, we can define *generalized inequalities* $\succeq_{\mathcal{K}}$, $\preceq_{\mathcal{K}}$ as follows:

$$x \succeq_{\mathcal{K}} y \quad \Longleftrightarrow \quad y \preceq_{\mathcal{K}} x \quad \Longleftrightarrow \quad x - y \in \mathcal{K}. \tag{5.16}$$

Generalized inequalities retain key properties of standard inequalities, including transitivity and invariance under nonnegative scaling. The most common cones considered in practice are the nonnegative orthant

$$\mathbb{R}^n_+ = \{\, x \in \mathbb{R}^n \mid x_i \geq 0,\ i = 1, 2, \dots, n \,\} \tag{5.17}$$

and the cone of positive semidefinite matrices

$$\mathcal{S}^n_+ \triangleq \left\{\, X = X^T \in \mathbb{R}^{n \times n} \mid \lambda_{\min}(X) \geq 0 \,\right\}. \tag{5.18}$$

For more information about generalized inequalities, see [29].

The concept of convexity can be extended to generalized inequalities. Given a vector space S, a function $g : \mathbb{R}^m \to S$, and a closed, convex, solid cone $\mathcal{K} \subseteq S$, we say that g is *$\mathcal{K}$-convex* if

$$g(\alpha x + (1-\alpha)y) \preceq_{\mathcal{K}} \alpha g(x) + (1-\alpha)g(y) \quad \forall x, y \in \mathbb{R}^m,\ \alpha \in (0,1). \tag{5.19}$$

Equivalently, a function g is $\mathcal{K}$-convex if and only if its *$\mathcal{K}$-epigraph*

$$\operatorname{epi}_{\mathcal{K}} g \triangleq \{\, (x,y) \in \mathbb{R}^m \times S \mid g(x) \preceq_{\mathcal{K}} y \,\} = \{\, (x,y) \in \mathbb{R}^m \times S \mid y - g(x) \in \mathcal{K} \,\} \tag{5.20}$$

is a convex set. This generalized epigraph definition makes it clear that $\mathcal{K}$-convexity is entirely compatible with convex programming. Incorporating generalized inequalities into convex programming has been considered by others, including [29, 179]; in the latter case it is called *conically ordered convex programming*.

We could consider extending the DCP ruleset to support generalized inequalities. The most interesting aspect would probably be the extension of the composition rules. For example, a function $f : S \to (\mathbb{R} \cup +\infty)$ is *$\mathcal{K}$-nondecreasing* if it satisfies

$$f(x) \geq f(y) \quad \forall x \succeq_{\mathcal{K}} y; \tag{5.21}$$

This definition allows us to construct the following rule: if f is convex and $\mathcal{K}$-nondecreasing and g is $\mathcal{K}$-convex, then $(f \circ g)(x) = f(g(x))$ is convex; and under those conditions, the composition can be separated for canonicalization purposes as follows:

$$f(g(x)) \leq y \quad \Longleftrightarrow \quad f(z) \leq y,\ g(x) \preceq_{\mathcal{K}} z. \tag{5.22}$$

Certainly the most likely use of generalized convexity is the manipulation of linear matrix inequalities. For example, let $\mathcal{S}^n$ be the set of symmetric $n \times n$ matrices, and $\mathcal{S}^n_+$ as defined above, and consider the following functions:

$$f_n : \mathcal{S}_n \to \mathbb{R}, \quad f_n(Z) \triangleq \lambda_{\max}(Z) \qquad g_n : \mathbb{R}^{m\times n} \to \mathcal{S}_n, \quad g_n(X) \triangleq X^T X. \tag{5.23}$$

It is not difficult to verify that f_n is convex and $\mathcal{S}^n_+$-nondecreasing, and g_n is $\mathcal{S}^n_+$-convex. So the composition $f_n \circ g_n$

$$f_n \circ g_n : \mathbb{R}^{m\times n} \to \mathbb{R}, \quad (f_n \circ g_n)(X) = \lambda_{\max}(X^T X) = \sigma_{\max}(X)^2 \tag{5.24}$$

obeys the extended composition rule; so the constraint $f_n(g_n(X)) \leq y$ is a DCP-compliant way to bound the squared maximum singular value of a matrix. Separating this composition in the manner suggested by (5.22) yields the following equivalence:

$$\sigma_{\max}(X)^2 \leq y \quad \Longleftrightarrow \quad \exists Z \quad \lambda_{\max}(Z) \leq y, \quad X^T X \preceq Z. \tag{5.25}$$

Those familiar with linear matrix inequalities may know that this same bound can be achieved using a single linear matrix inequality,

$$\sigma_{\max}(X)^2 \leq y \quad \Longleftrightarrow \quad \begin{bmatrix} I & X \\ X^T & yI \end{bmatrix} \in \mathcal{S}_+^{m+n}. \tag{5.26}$$

It would be interesting to determine which of these conversions proves more efficient in practice.

Adding support for generalized inequalities to disciplined convex programming poses two challenges: finding an elegant way of integrating it into the modeling language, and extending the canonical form and numerical algorithms to account for them.

5.2.3 Advanced applications

Geometric programming

In §1.2.2 we briefly discussed geometric programs (GPs) as a standard form for convex programming, and stated that they were a unique case in that they are not convex in their "native" form, but require a change of variables to convert them into convex form.

Because of the increasing popularity of geometric programming, it seems sensible to provide "native" support for geometric programming under the disciplined convex programming methodology, and in the `cvx` modeling framework in particular. The cleanest way to do so would be to create a separate parser that accepts geometric programs specified in their native form, verifies their validity, and converts the problem into a DCP for `cvx` to solve. On the back end, the software would accept the results returned by `cvx` and perform a reverse change of variables to convert the results back to native form. The dual information must be converted as well.

For a more complete introduction to geometric programming, we refer the reader to [27]. In that article, the authors introduce a number of useful problem types that are not commonly thought of as GPs, but can easily be transformed into them. Their work is not dissimilar in spirit to disciplined convex programming; in fact, it suggests a sort of "disciplined geometric programming" methodology, consisting of a set of rules that govern the construction of problems that can be quickly and automatically verified and converted to "canonical" geometric form—and, therefore, to DCPs.

Automatic relaxations of nonconvex problems

Problems that are not convex often admit useful convex approximations or relaxations. For example, we can define a *mixed-integer disciplined convex program* (MIDCP) as a problem involving integer variables that reduces to a DCP when the integer constraints are relaxed. The traditional techniques to solve mixed-integer linear and nonlinear programs (MILPs and MINLPs) can be easily adapted to the disciplined convex programming case. A number of more novel convex relaxations of nonconvex programs have been studied recently as well. For example, certain problems involving nonconvex quadratic constraints can be relaxed into semidefinite programs (SDPs); *e.g.*, [173, 21]. These relaxations often produce surprisingly close approximations.

An interesting topic for future study would be to create a system that would accept certain classes of nonconvex problems in the `cvx` modeling language, and automatically generate and solve convex relaxations of them. The new software would then convert the problem into relaxed form as a DCP, which the standard DCP system would solve. The primal and dual results would then be mapped back to the original problem.

5.3 A brief defense—or, there's nothing like leather

A town fear'd a siege, and held consultation,
Which was the best method of fortification:
A grave skilful mason gave in his opinion,
That nothing but stone could secure the dominion.
A carpenter said, tho' that was well spoke,
Yet 'twas better by far to defend it with oak.
A currier[1] *(much wiser than both these together,)*
Said; try what you please there's nothing like leather.
— Daniel Fenning, *The Universal Spelling-Book*, 1767

We are all inclined to favor the tools with which we have the most expertise. But we would hope that, unlike the currier, we would recognize when ours are simply not suitable to the task at hand. Convex programming has the potential to be an incredibly powerful tool: convex programs are well behaved and can be solved reliably, efficiently, and accurately. Nevertheless, there are many applications for which no convex model can be found.

Still, there is good reason to be optimistic about the usefulness of convex programming in a new field of interest. Examining the gamut of known applications for convex programming, we see several scenarios. For some, a simple, idealized model was chosen as a starting point, then incrementally improved—while maintaining convexity, of course. For others, the models first presented themselves as nonconvex, but proved otherwise once a key transformation or change of variables was found. Still others are in fact approximations or relaxations of nonconvex problems, but useful nonetheless because the original, nonconvex model cannot be solved in practice.

It seems appropriate, then, to at least attempt similar searches for convexity in new mathematical programming applications, given the significant theoretical benefits that accrue if successful. The primary reason *not* to do so is that the expertise required proves too great. It is the author's hope that disciplined convex programming removes this impediment.

[1]One who prepares tanned animal hides for use.

Appendix A

cvx: An example

In this appendix we provide the full `cvx` output for the example problem from §5.1,

$$\begin{array}{ll} \text{minimize} & \|Ax-b\|_{3.5} \\ \text{subject to} & -1 \leq x \leq 1 \end{array} \tag{A.1}$$

for the following specific values of A and b, respectively:

$$\begin{bmatrix} +1.1991370639 & -1.2233383780 & -0.0866841750 & +0.7887334551 & -0.5210075139 \\ -2.5773355744 & +0.3032048978 & -0.0126202764 & -0.5770829494 & -0.1553593843 \\ -2.0862702167 & -0.7300973691 & -0.3458579112 & +0.5276343380 & -0.0984983465 \\ +0.3861318799 & -1.1435702969 & +0.9863109678 & +1.6716940874 & +0.9971702607 \\ -0.8610312724 & -1.4131928935 & +0.6432561643 & +0.8000794868 & +0.4344697335 \\ -1.2308076439 & -0.5918178669 & +2.9199441507 & +0.8837869741 & -0.0257208688 \\ +2.6415543653 & +0.5188876241 & -1.2485849488 & -0.2241850743 & -0.3799341199 \\ -0.9044035801 & -1.4928111869 & +0.1571147888 & +0.2969911629 & -0.2423955222 \end{bmatrix}, \quad \begin{bmatrix} -2.3302283037 \\ +1.5132101113 \\ +2.3283246440 \\ -2.0469091512 \\ +3.4031789976 \\ -0.9883449326 \\ +0.3454270903 \\ +0.7081227198 \end{bmatrix}. \tag{A.2}$$

The original model file is given in Figure 5.1, except that the first line

```
minimize norm_2
```

was replaced by

```
minimize norm_p( 3.5,
```

in order to select the $\ell_{3.5}$ norm. The output listing that is provided in this appendix was generated by executing a single command,

```
cvx minnorm.cvx minnorm.dat
```

where `minnorm.cvx` and `minnorm.dat` contain the model and the numerical data, respectively. The only modifications that have been made to it are to edit it to fit within the margins of the page, and to break it into the various computational phases to be introduced separately:

- initial parsing (§A.1),
- convexity verification (§A.2),

- canonicalization (§A.3),
- numerical instance generation (§A.4), and
- numerical solution (§A.5).

The purpose of this appendix is primarily to demonstrate the existence and successful operation of the `cvx` software, and to simply give an example of its operation. While we do provide some brief commentary with each section of the listing, we have not attempted to provide a complete and detailed explanation of it.

A.1 Initial parsing

In this phase, the file `minnorm.cvx` is read in and the problem is verified against the grammar of the `cvx` modeling language. At this point, the `cvx` program has *not* yet verified that the problem obeys the DCP ruleset; that takes place in the second phase.

The included file `norms.cvx`, containing the definitions of the various norms, has been read in; and only those functions actually required for this problem have been kept. The function `rootperspec` has actually been implemented in `C++`, and is not shown here, but implements the concave function

$$f_p : \mathbb{R} \times \mathbb{R} \to (\mathbb{R} \cup -\infty), \quad f_p(s,q) \triangleq \begin{cases} q(s/q)^{1/p} & q > 0,\ s \geq 0 \\ -\infty & \text{otherwise,} \end{cases} \tag{A.3}$$

which is used in the graph implementation of `norm_p`, the ℓ_p norm. Note also the use of the dollar signs `$1`, `$2`, *etc.* in the function declarations; these correspond to the first, second, *etc.* arguments of the function, respectively. They are an artifact of the processing steps; the actual function definitions in the file `norms.cvx` use the actual argument names instead.

```
CVX: A parser and solver for convex programs

Reading problem file: minnorm.cvx
Files successfully read.

Original specification:
------------------------------------------------------------

parameters
   a11, a12, a13, a14, a15, a21, a22, a23, a24, a25, a31, a32, a33, a34,
   a35, a41, a42, a43, a44, a45, a51, a52, a53, a54, a55, a61, a62, a63,
   a64, a65, a71, a72, a73, a74, a75, a81, a82, a83, a84, a85, b1, b2, b3,
   b4, b5, b6, b7, b8;
variables
   x1, x2, x3, x4, x5;
```

```
minimize
   norm_p( 3, a11 x1 + a12 x2 + a13 x3 + a14 x4 + a15 x5 - b1,
              a21 x1 + a22 x2 + a23 x3 + a24 x4 + a25 x5 - b2,
              a31 x1 + a32 x2 + a33 x3 + a34 x4 + a35 x5 - b3,
              a41 x1 + a42 x2 + a43 x3 + a44 x4 + a45 x5 - b4,
              a51 x1 + a52 x2 + a53 x3 + a54 x4 + a55 x5 - b5,
              a61 x1 + a62 x2 + a63 x3 + a64 x4 + a65 x5 - b6,
              a71 x1 + a72 x2 + a73 x3 + a74 x4 + a75 x5 - b7,
              a81 x1 + a82 x2 + a83 x3 + a84 x4 + a85 x5 - b8 );

constraints
   -1 <= x1 <= 1;
   -1 <= x2 <= 1;
   -1 <= x3 <= 1;
   -1 <= x4 <= 1;
   -1 <= x5 <= 1;

function norm_p( p, x1, x2, x3, x4, x5, x6, x7, x8 )
   convex( $1, $2, $3, $4, $5, $6, $7, $8 ) if $0 > 1
   {
      epigraph := {
         variables xp1, xp2, xp3, xp4, xp5, xp6, xp7, xp8, xm1, xm2,
            xm3, xm4, xm5, xm6, xm7, xm8, s1, s2, s3, s4, s5, s6, s7, s8;
         xp1 >= 0;
         xp2 >= 0;
         xp3 >= 0;
         xp4 >= 0;
         xp5 >= 0;
         xp6 >= 0;
         xp7 >= 0;
         xp8 >= 0;
         xm1 >= 0;
         xm2 >= 0;
         xm3 >= 0;
         xm4 >= 0;
         xm5 >= 0;
         xm6 >= 0;
         xm7 >= 0;
         xm8 >= 0;
         x1 = xp1 - xm1;
         x2 = xp2 - xm2;
         x3 = xp3 - xm3;
         x4 = xp4 - xm4;
         x5 = xp5 - xm5;
         x6 = xp6 - xm6;
         x7 = xp7 - xm7;
         x8 = xp8 - xm8;
         xp1 + xm1 <= rootperspec( p, s1, norm_p );
         xp2 + xm2 <= rootperspec( p, s2, norm_p );
         xp3 + xm3 <= rootperspec( p, s3, norm_p );
         xp4 + xm4 <= rootperspec( p, s4, norm_p );
         xp5 + xm5 <= rootperspec( p, s5, norm_p );
         xp6 + xm6 <= rootperspec( p, s6, norm_p );
         xp7 + xm7 <= rootperspec( p, s7, norm_p );
         xp8 + xm8 <= rootperspec( p, s8, norm_p );
         s1 + s2 + s3 + s4 + s5 + s6 + s7 + s8 = norm_p;
      }
   }
function rootperspec( p, s, q ) concave( $1, $2 ) nondecreasing( $1 )
                                nonincreasing( $2 );

------------------------------------------------------------
```

A.2 Ruleset verification

In this phase, the problem is verified for compliance with the DCP ruleset. The output resembles the indented proofs given in §3.3. The graph implementation of the function `norm_p` is verified in the same manner as the main problem.

```
Diagnosing convexity:
No errors found.

Convexity analysis:
------------------------------------------------------------

parameters
   a11, a12, a13, a14, a15, a21, a22, a23, a24, a25, a31, a32, a33, a34,
   a35, a41, a42, a43, a44, a45, a51, a52, a53, a54, a55, a61, a62, a63,
   a64, a65, a71, a72, a73, a74, a75, a81, a82, a83, a84, a85, b1, b2, b3,
   b4, b5, b6, b7, b8;
variables
   x1, x2, x3, x4, x5;

minimize
   norm_p( 3, a11 x1 + a12 x2 + a13 x3 + a14 x4 + a15 x5 - b1,
              a21 x1 + a22 x2 + a23 x3 + a24 x4 + a25 x5 - b2,
              a31 x1 + a32 x2 + a33 x3 + a34 x4 + a35 x5 - b3,
              a41 x1 + a42 x2 + a43 x3 + a44 x4 + a45 x5 - b4,
              a51 x1 + a52 x2 + a53 x3 + a54 x4 + a55 x5 - b5,
              a61 x1 + a62 x2 + a63 x3 + a64 x4 + a65 x5 - b6,
              a71 x1 + a72 x2 + a73 x3 + a74 x4 + a75 x5 - b7,
              a81 x1 + a82 x2 + a83 x3 + a84 x4 + a85 x5 - b8 ) :: convex;
      3 :: constant;
      a11 x1 + a12 x2 + a13 x3 + a14 x4 + a15 x5 - b1 :: affine;
         a11 x1 :: affine;
      a11 :: constant;
      x1 :: affine;
   a12 x2 :: affine;
      a12 :: constant;
      x2 :: affine;
   a13 x3 :: affine;
      a13 :: constant;
      x3 :: affine;
   a14 x4 :: affine;
      a14 :: constant;
      x4 :: affine;
   a15 x5 :: affine;
      a15 :: constant;
      x5 :: affine;
   -b1 :: constant;
      b1 :: constant;
a21 x1 + a22 x2 + a23 x3 + a24 x4 + a25 x5 - b2 :: affine;
   a21 x1 :: affine;
      a21 :: constant;
      x1 :: affine;
   a22 x2 :: affine;
      a22 :: constant;
      x2 :: affine;
   a23 x3 :: affine;
      a23 :: constant;
      x3 :: affine;
   a24 x4 :: affine;
      a24 :: constant;
      x4 :: affine;
   a25 x5 :: affine;
      a25 :: constant;
      x5 :: affine;
   -b2 :: constant;
      b2 :: constant;
a31 x1 + a32 x2 + a33 x3 + a34 x4 + a35 x5 - b3 :: affine;
   a31 x1 :: affine;
      a31 :: constant;
      x1 :: affine;
   a32 x2 :: affine;
      a32 :: constant;
      x2 :: affine;
   a33 x3 :: affine;
      a33 :: constant;
      x3 :: affine;
   a34 x4 :: affine;
      a34 :: constant;
      x4 :: affine;
   a35 x5 :: affine;
```

```
      a35 :: constant;
      x5 :: affine;
   -b3 :: constant;
      b3 :: constant;
a41 x1 + a42 x2 + a43 x3 + a44 x4 + a45 x5 - b4 :: affine;
   a41 x1 :: affine;
      a41 :: constant;
      x1 :: affine;
   a42 x2 :: affine;
      a42 :: constant;
      x2 :: affine;
   a43 x3 :: affine;
      a43 :: constant;
      x3 :: affine;
   a44 x4 :: affine;
      a44 :: constant;
      x4 :: affine;
   a45 x5 :: affine;
      a45 :: constant;
      x5 :: affine;
   -b4 :: constant;
      b4 :: constant;
a51 x1 + a52 x2 + a53 x3 + a54 x4 + a55 x5 - b5 :: affine;
   a51 x1 :: affine;
      a51 :: constant;
      x1 :: affine;
   a52 x2 :: affine;
      a52 :: constant;
      x2 :: affine;
   a53 x3 :: affine;
      a53 :: constant;
      x3 :: affine;
   a54 x4 :: affine;
      a54 :: constant;
      x4 :: affine;
   a55 x5 :: affine;
      a55 :: constant;
      x5 :: affine;
   -b5 :: constant;
      b5 :: constant;
a61 x1 + a62 x2 + a63 x3 + a64 x4 + a65 x5 - b6 :: affine;
   a61 x1 :: affine;
      a61 :: constant;
      x1 :: affine;
   a62 x2 :: affine;
      a62 :: constant;
      x2 :: affine;
   a63 x3 :: affine;
```

```
            a63 :: constant;
            x3 :: affine;
         a64 x4 :: affine;
            a64 :: constant;
            x4 :: affine;
         a65 x5 :: affine;
            a65 :: constant;
            x5 :: affine;
         -b6 :: constant;
            b6 :: constant;
      a71 x1 + a72 x2 + a73 x3 + a74 x4 + a75 x5 - b7 :: affine;
         a71 x1 :: affine;
            a71 :: constant;
            x1 :: affine;
         a72 x2 :: affine;
            a72 :: constant;
            x2 :: affine;
         a73 x3 :: affine;
            a73 :: constant;
            x3 :: affine;
         a74 x4 :: affine;
            a74 :: constant;
            x4 :: affine;
         a75 x5 :: affine;
            a75 :: constant;
            x5 :: affine;
         -b7 :: constant;
            b7 :: constant;
      a81 x1 + a82 x2 + a83 x3 + a84 x4 + a85 x5 - b8 :: affine;
         a81 x1 :: affine;
            a81 :: constant;
            x1 :: affine;
         a82 x2 :: affine;
            a82 :: constant;
            x2 :: affine;
         a83 x3 :: affine;
            a83 :: constant;
            x3 :: affine;
         a84 x4 :: affine;
            a84 :: constant;
            x4 :: affine;
         a85 x5 :: affine;
            a85 :: constant;
            x5 :: affine;
         -b8 :: constant;
            b8 :: constant;

constraints
```

```
   -1 <= x1 <= 1 :: bool convex;
      -1 <= x1 :: bool convex;
         -1 :: constant;
            1 :: constant;
         x1 :: affine;
      x1 <= 1 :: bool convex;
         x1 :: affine;
         1 :: constant;
   -1 <= x2 <= 1 :: bool convex;
      -1 <= x2 :: bool convex;
         -1 :: constant;
            1 :: constant;
         x2 :: affine;
      x2 <= 1 :: bool convex;
         x2 :: affine;
         1 :: constant;
   -1 <= x3 <= 1 :: bool convex;
      -1 <= x3 :: bool convex;
         -1 :: constant;
            1 :: constant;
         x3 :: affine;
      x3 <= 1 :: bool convex;
         x3 :: affine;
         1 :: constant;
   -1 <= x4 <= 1 :: bool convex;
      -1 <= x4 :: bool convex;
         -1 :: constant;
            1 :: constant;
         x4 :: affine;
      x4 <= 1 :: bool convex;
         x4 :: affine;
         1 :: constant;
   -1 <= x5 <= 1 :: bool convex;
      -1 <= x5 :: bool convex;
         -1 :: constant;
            1 :: constant;
         x5 :: affine;
      x5 <= 1 :: bool convex;
         x5 :: affine;
         1 :: constant;

function norm_p( p, x1, x2, x3, x4, x5, x6, x7, x8 )
   convex( $1, $2, $3, $4, $5, $6, $7, $8 ) if $0 > 1
   {
      epigraph := {
         variables xp1, xp2, xp3, xp4, xp5, xp6, xp7, xp8, xm1, xm2,
            xm3, xm4, xm5, xm6, xm7, xm8, s1, s2, s3, s4, s5, s6, s7, s8;
         xp1 >= 0 :: bool convex;
            xp1 :: affine;
            0 :: constant;
         xp2 >= 0 :: bool convex;
            xp2 :: affine;
            0 :: constant;
         xp3 >= 0 :: bool convex;
            xp3 :: affine;
            0 :: constant;
         xp4 >= 0 :: bool convex;
            xp4 :: affine;
            0 :: constant;
         xp5 >= 0 :: bool convex;
            xp5 :: affine;
            0 :: constant;
         xp6 >= 0 :: bool convex;
            xp6 :: affine;
            0 :: constant;
         xp7 >= 0 :: bool convex;
            xp7 :: affine;
            0 :: constant;
         xp8 >= 0 :: bool convex;
            xp8 :: affine;
            0 :: constant;
         xm1 >= 0 :: bool convex;
            xm1 :: affine;
            0 :: constant;
         xm2 >= 0 :: bool convex;
            xm2 :: affine;
            0 :: constant;
         xm3 >= 0 :: bool convex;
            xm3 :: affine;
            0 :: constant;
         xm4 >= 0 :: bool convex;
            xm4 :: affine;
            0 :: constant;
         xm5 >= 0 :: bool convex;
            xm5 :: affine;
            0 :: constant;
         xm6 >= 0 :: bool convex;
            xm6 :: affine;
            0 :: constant;
         xm7 >= 0 :: bool convex;
            xm7 :: affine;
            0 :: constant;
         xm8 >= 0 :: bool convex;
            xm8 :: affine;
            0 :: constant;
         x1 = xp1 - xm1 :: bool convex;
```

```
   x1 :: affine;
   xp1 - xm1 :: affine;
      xp1 :: affine;
      -xm1 :: affine;
         xm1 :: affine;
x2 = xp2 - xm2 :: bool convex;
   x2 :: affine;
   xp2 - xm2 :: affine;
      xp2 :: affine;
      -xm2 :: affine;
         xm2 :: affine;
x3 = xp3 - xm3 :: bool convex;
   x3 :: affine;
   xp3 - xm3 :: affine;
      xp3 :: affine;
      -xm3 :: affine;
         xm3 :: affine;
x4 = xp4 - xm4 :: bool convex;
   x4 :: affine;
   xp4 - xm4 :: affine;
      xp4 :: affine;
      -xm4 :: affine;
         xm4 :: affine;
x5 = xp5 - xm5 :: bool convex;
   x5 :: affine;
   xp5 - xm5 :: affine;
      xp5 :: affine;
      -xm5 :: affine;
         xm5 :: affine;
x6 = xp6 - xm6 :: bool convex;
   x6 :: affine;
   xp6 - xm6 :: affine;
      xp6 :: affine;
      -xm6 :: affine;
         xm6 :: affine;
x7 = xp7 - xm7 :: bool convex;
   x7 :: affine;
   xp7 - xm7 :: affine;
      xp7 :: affine;
      -xm7 :: affine;
         xm7 :: affine;
x8 = xp8 - xm8 :: bool convex;
   x8 :: affine;
   xp8 - xm8 :: affine;
      xp8 :: affine;
      -xm8 :: affine;
         xm8 :: affine;
xp1 + xm1 <= rootperspec( p, s1, norm_p ) :: bool convex;
```

```
   xp1 + xm1 :: affine;
      xp1 :: affine;
      xm1 :: affine;
   rootperspec( p, s1, norm_p ) :: concave( $1, $2 );
      p :: constant;
      s1 :: affine;
      norm_p :: affine;
xp2 + xm2 <= rootperspec( p, s2, norm_p ) :: bool convex;
   xp2 + xm2 :: affine;
      xp2 :: affine;
      xm2 :: affine;
   rootperspec( p, s2, norm_p ) :: concave( $1, $2 );
      p :: constant;
      s2 :: affine;
      norm_p :: affine;
xp3 + xm3 <= rootperspec( p, s3, norm_p ) :: bool convex;
   xp3 + xm3 :: affine;
      xp3 :: affine;
      xm3 :: affine;
   rootperspec( p, s3, norm_p ) :: concave( $1, $2 );
      p :: constant;
      s3 :: affine;
      norm_p :: affine;
xp4 + xm4 <= rootperspec( p, s4, norm_p ) :: bool convex;
   xp4 + xm4 :: affine;
      xp4 :: affine;
      xm4 :: affine;
   rootperspec( p, s4, norm_p ) :: concave( $1, $2 );
      p :: constant;
      s4 :: affine;
      norm_p :: affine;
xp5 + xm5 <= rootperspec( p, s5, norm_p ) :: bool convex;
   xp5 + xm5 :: affine;
      xp5 :: affine;
      xm5 :: affine;
   rootperspec( p, s5, norm_p ) :: concave( $1, $2 );
      p :: constant;
      s5 :: affine;
      norm_p :: affine;
xp6 + xm6 <= rootperspec( p, s6, norm_p ) :: bool convex;
   xp6 + xm6 :: affine;
      xp6 :: affine;
      xm6 :: affine;
   rootperspec( p, s6, norm_p ) :: concave( $1, $2 );
      p :: constant;
      s6 :: affine;
      norm_p :: affine;
xp7 + xm7 <= rootperspec( p, s7, norm_p ) :: bool convex;
```

```
            xp7 + xm7 :: affine;
               xp7 :: affine;
               xm7 :: affine;
            rootperspec( p, s7, norm_p ) :: concave( $1, $2 );
               p :: constant;
               s7 :: affine;
               norm_p :: affine;
         xp8 + xm8 <= rootperspec( p, s8, norm_p ) :: bool convex;
            xp8 + xm8 :: affine;
               xp8 :: affine;
               xm8 :: affine;
            rootperspec( p, s8, norm_p ) :: concave( $1, $2 );
               p :: constant;
               s8 :: affine;
               norm_p :: affine;
         s1 + s2 + s3 + s4 + s5 + s6 + s7 + s8 = norm_p :: bool convex;
            s1 + s2 + s3 + s4 + s5 + s6 + s7 + s8 :: affine;
               s1 :: affine;
               s2 :: affine;
               s3 :: affine;
               s4 :: affine;
               s5 :: affine;
               s6 :: affine;
               s7 :: affine;
               s8 :: affine;
            norm_p :: affine;
      }
   }
function rootperspec( p, s, q ) concave( $1, $2 ) nondecreasing( $1 )
         nonincreasing( $2 );

------------------------------------------------------------
```

A.3 Canonicalization

In this stage, the canonical form is generated. This phase's output is the most difficult to read, but a couple of comments should prove helpful. First of all, The objects `@1`, `@2`, `@3`, *etc.* are the additional variables that have been created during the canonicalization process. Secondly, the double slash `//` represents a comment in the `cvx` modeling language; any text following it is ignored. So in this listing, the comments are used to show how functions, sets, *etc.* looked *before* canonicalization. Immediately following such comments is the replacement. Sometimes the replacement itself requires further processing, in which case it is commented out as well.

For example, immediately after the `minimize` keyword below, the reference to the `norm_p` function from the objective function has been commented out. That is because it has been replaced with a temporary variable; *i.e.*,

$$\text{minimize} \quad \|Ax-b\|_{3.5} \qquad \Longrightarrow \qquad \begin{array}{ll} \text{minimize} & t_1 \\ & \|Ax-b\|_{3.5} \leq t_1 \end{array} \tag{A.4}$$

Of course, at this point, further canonicalization is then performed on the constraint $\|Ax-b\|_{3.5} \leq t_1$ as well, so it too is commented out. In fact, `norm_p` is replaced with its graph implementation, so the final result looks quite different from the original problem.

```
Generating canonical form:
No errors found.

Unrolled problem:
------------------------------------------------------------


parameters
   a11, a12, a13, a14, a15, a21, a22, a23, a24, a25, a31, a32, a33, a34,
   a35, a41, a42, a43, a44, a45, a51, a52, a53, a54, a55, a61, a62, a63,
   a64, a65, a71, a72, a73, a74, a75, a81, a82, a83, a84, a85, b1, b2, b3,
   b4, b5, b6, b7, b8;
variables
   x1, x2, x3, x4, x5, @1, @2, @3, @4, @5, @6, @7, @8, @9, @10, @11, @12,
   @13, @14, @15, @16, @17, @18, @19, @20, @21, @22, @23, @24, @25, @26,
   @27, @28, @29, @30, @31, @32, @33, @34, @35, @36, @37, @38, @39, @40,
   @41, @42, @43, @44, @45, @46, @47, @48, @49, @50, @51, @52, @53, @54,
   @55, @56, @57, @58;
define
   @d1 := 3;

minimize
// norm_p( 3, a11 x1 + a12 x2 + a13 x3 + a14 x4 + a15 x5 - b1,
//           a21 x1 + a22 x2 + a23 x3 + a24 x4 + a25 x5 - b2,
//           a31 x1 + a32 x2 + a33 x3 + a34 x4 + a35 x5 - b3,
//           a41 x1 + a42 x2 + a43 x3 + a44 x4 + a45 x5 - b4,
//           a51 x1 + a52 x2 + a53 x3 + a54 x4 + a55 x5 - b5,
//           a61 x1 + a62 x2 + a63 x3 + a64 x4 + a65 x5 - b6,
//           a71 x1 + a72 x2 + a73 x3 + a74 x4 + a75 x5 - b7,
//           a81 x1 + a82 x2 + a83 x3 + a84 x4 + a85 x5 - b8 );
      @1;
//       @1 >= norm_p( 3, a11 x1 + a12 x2 + a13 x3 + a14 x4 + a15 x5 - b1,
//                        a21 x1 + a22 x2 + a23 x3 + a24 x4 + a25 x5 - b2,
//                        a31 x1 + a32 x2 + a33 x3 + a34 x4 + a35 x5 - b3,
//                        a41 x1 + a42 x2 + a43 x3 + a44 x4 + a45 x5 - b4,
//                        a51 x1 + a52 x2 + a53 x3 + a54 x4 + a55 x5 - b5,
//                        a61 x1 + a62 x2 + a63 x3 + a64 x4 + a65 x5 - b6,
//                        a71 x1 + a72 x2 + a73 x3 + a74 x4 + a75 x5 - b7,
//                        a81 x1 + a82 x2 + a83 x3 + a84 x4 + a85 x5 - b8 );
//          ( @d1, @2, @3, @4, @5, @6, @7, @8, @9, @1 ) in epi norm_p;
//             @d1 := 3;
               @2 = a11 x1 + a12 x2 + a13 x3 + a14 x4 + a15 x5 - b1;
               @3 = a21 x1 + a22 x2 + a23 x3 + a24 x4 + a25 x5 - b2;
               @4 = a31 x1 + a32 x2 + a33 x3 + a34 x4 + a35 x5 - b3;
               @5 = a41 x1 + a42 x2 + a43 x3 + a44 x4 + a45 x5 - b4;
               @6 = a51 x1 + a52 x2 + a53 x3 + a54 x4 + a55 x5 - b5;
               @7 = a61 x1 + a62 x2 + a63 x3 + a64 x4 + a65 x5 - b6;
               @8 = a71 x1 + a72 x2 + a73 x3 + a74 x4 + a75 x5 - b7;
               @9 = a81 x1 + a82 x2 + a83 x3 + a84 x4 + a85 x5 - b8;
               @10 >= 0;
               @11 >= 0;
               @12 >= 0;
               @13 >= 0;
               @14 >= 0;
               @15 >= 0;
               @16 >= 0;
               @17 >= 0;
               @18 >= 0;
               @19 >= 0;
               @20 >= 0;
               @21 >= 0;
               @22 >= 0;
               @23 >= 0;
               @24 >= 0;
               @25 >= 0;
               @2 = @10 - @18;
               @3 = @11 - @19;
               @4 = @12 - @20;
               @5 = @13 - @21;
               @6 = @14 - @22;
               @7 = @15 - @23;
               @8 = @16 - @24;
               @9 = @17 - @25;
//             @10 + @18 <= rootperspec( @d1, @26, @1 );
                  ( @26, @1, @34 ) in hyp @rootperspec( @d1 );
                     @34 = @10 + @18;
//             @11 + @19 <= rootperspec( @d1, @27, @1 );
                  ( @27, @52, @35 ) in hyp @rootperspec( @d1 );
                     @52 = @1;
                     @35 = @11 + @19;
//             @12 + @20 <= rootperspec( @d1, @28, @1 );
                  ( @28, @53, @36 ) in hyp @rootperspec( @d1 );
                     @53 = @1;
                     @36 = @12 + @20;
//             @13 + @21 <= rootperspec( @d1, @29, @1 );
                  ( @29, @54, @37 ) in hyp @rootperspec( @d1 );
                     @54 = @1;
                     @37 = @13 + @21;
```

```
//             @14 + @22 <= rootperspec( @d1, @30, @1 );
                  ( @30, @55, @38 ) in hyp @rootperspec( @d1 );
                     @55 = @1;
                     @38 = @14 + @22;
//             @15 + @23 <= rootperspec( @d1, @31, @1 );
                  ( @31, @56, @39 ) in hyp @rootperspec( @d1 );
                     @56 = @1;
                     @39 = @15 + @23;
//             @16 + @24 <= rootperspec( @d1, @32, @1 );
                  ( @32, @57, @40 ) in hyp @rootperspec( @d1 );

constraints
               @57 = @1;
               @40 = @16 + @24;
//       @17 + @25 <= rootperspec( @d1, @33, @1 );
            ( @33, @58, @41 ) in hyp @rootperspec( @d1 );
               @58 = @1;
               @41 = @17 + @25;
         @26 + @27 + @28 + @29 + @30 + @31 + @32 + @33 = @1;
// -1 <= x1 <= 1;
//    -1 <= x1;
         - 1 + @42 = x1;
            @42 >= 0;
//    x1 <= 1;
         x1 + @43 = 1;
            @43 >= 0;
// -1 <= x2 <= 1;
//    -1 <= x2;
         - 1 + @44 = x2;
            @44 >= 0;
//    x2 <= 1;
         x2 + @45 = 1;
            @45 >= 0;
// -1 <= x3 <= 1;
//    -1 <= x3;
         - 1 + @46 = x3;
            @46 >= 0;
//    x3 <= 1;
         x3 + @47 = 1;
            @47 >= 0;
// -1 <= x4 <= 1;
//    -1 <= x4;
         - 1 + @48 = x4;
            @48 >= 0;
//    x4 <= 1;
         x4 + @49 = 1;
            @49 >= 0;
// -1 <= x5 <= 1;
//    -1 <= x5;
         - 1 + @50 = x5;
            @50 >= 0;
//    x5 <= 1;
         x5 + @51 = 1;
            @51 >= 0;

-------------------------------------------------------------
```

A.4 Numerical instance generation

In this phase, the numerical data is read in, and the actual instance of the problem to be solved is generated. We have taken the liberty to shave a couple of digits off the numerical values in the first 8 equality constraints in order to keep the text within the margins.

```
Reading data file: minnorm.dat
Generating numerical instance:
No errors found.

Numerical instance:
-------------------------------------------------------------

minimize
   @1;
subject to
 - 1.19914 x1 + 1.22334 x2 + 0.08669 x3 - 0.78873 x4 + 0.52101 x5 + @2 =  2.33023;
   2.57734 x1 - 0.30321 x2 + 0.01262 x3 + 0.57708 x4 + 0.15537 x5 + @3 = -1.51321;
   2.08627 x1 + 0.73010 x2 + 0.34586 x3 - 0.52763 x4 + 0.09850 x5 + @4 = -2.32832;
 - 0.38613 x1 + 1.14357 x2 - 0.98631 x3 - 1.67169 x4 - 0.99717 x5 + @5 =  2.04691;
   0.86103 x1 + 1.41319 x2 - 0.64326 x3 - 0.80008 x4 - 0.43447 x5 + @6 = -3.40318;
   1.23081 x1 + 0.59182 x2 - 2.91994 x3 - 0.88379 x4 + 0.02572 x5 + @7 =  0.98835;
 - 2.64155 x1 - 0.51889 x2 + 1.24858 x3 + 0.22419 x4 + 0.37993 x5 + @8 = -0.34543;
   0.90440 x1 + 1.49281 x2 - 0.15712 x3 - 0.29699 x4 + 0.24240 x5 + @9 = -0.70812;
   @2 + @18 - @10 = 0;
   @3 - @11 + @19 = 0;
   @20 - @12 + @4 = 0;
   - @13 + @21 + @5 = 0;
   - @14 + @6 + @22 = 0;
   - @15 + @7 + @23 = 0;
   - @16 + @8 + @24 = 0;
   - @17 + @9 + @25 = 0;
   - @18 - @10 + @34 = 0;
   - @1 + @52 = 0;
   - @11 - @19 + @35 = 0;
   - @1 + @53 = 0;
   - @20 - @12 + @36 = 0;
   - @1 + @54 = 0;
   - @13 - @21 + @37 = 0;
   - @1 + @55 = 0;
   - @14 - @22 + @38 = 0;
   @56 - @1 = 0;
   - @15 - @23 + @39 = 0;
   - @1 + @57 = 0;
   - @16 - @24 + @40 = 0;
   - @1 + @58 = 0;
   - @17 - @25 + @41 = 0;
   - @1 + @26 + @28 + @27 + @29 + @30 + @31 + @32 + @33 = 0;
   - x1 + @42 = 1;
   x1 + @43 = 1;
   - x2 + @44 = 1;
   x2 + @45 = 1;
   - x3 + @46 = 1;
   x3 + @47 = 1;
   - x4 + @48 = 1;
   x4 + @49 = 1;
   - x5 + @50 = 1;
   x5 + @51 = 1;
   ( @10, @11, @12, @13, @14, @15, @16, @17, @18, @19, @20, @21, @22, @23, @24,
     @25, @42, @43, @44, @45, @46, @47, @48, @49, @50, @51 ) in Rn+;
   ( @26, @1, @34 ) in hyp rootperspec( 3 );
   ( @27, @52, @35 ) in hyp rootperspec( 3 );
   ( @28, @53, @36 ) in hyp rootperspec( 3 );
   ( @29, @54, @37 ) in hyp rootperspec( 3 );
   ( @30, @55, @38 ) in hyp rootperspec( 3 );
   ( @31, @56, @39 ) in hyp rootperspec( 3 );
   ( @32, @57, @40 ) in hyp rootperspec( 3 );
   ( @33, @58, @41 ) in hyp rootperspec( 3 );
variables
   x1, x2, x3, x4, x5;
temporaries
   @10, @11, @12, @13, @14, @15, @16, @17, @18, @19, @20, @21, @22, @23, @24, @25,
   @42, @43, @44, @45, @46, @47, @48, @49, @50, @51, @26, @1, @34, @27, @52, @35,
   @28, @53, @36, @29, @54, @37, @30, @55, @38, @31, @56, @39, @32, @57, @40, @33,
   @58, @41, @2, @3, @4, @5, @6, @7, @8, @9;

------------------------------------------------------------
```

A.5 Numerical solution

Finally, the numerical solver is called. A two-phase barrier method is employed: the first phase solves a modified version of the problem to find a strictly feasible point, which is then used to initialize the second phase of the algorithm. The columns are as follows:

- the iteration number;

- `objective`: the current objective value;
- `barrier`: the value of the barrier function plus objective (see §4.3.1);
- `gap target`: the duality gap that would be achieved if the Newton method were allowed to go to completion at this stage of the algorithm.
- `primal`: the error in the equality constraints;
- `dual`: the error in the dual feasibility conditions;
- `decrement`: the current Newton decrement (see [124]);
- `L.S. step`: the size of the line search step taken.

An asterisk `*` indicates that the given quantity has fallen to within its target tolerance; and a caret `^` indicates when the Newton decrement is sufficiently small, indicating convergence of the barrier problem, and prompting a reduction in the gap target. The second phase is complete when the gap target, the primal feasibility, and *either* the dual feasibility or the Newton decrement reach their target tolerances.

```
Solving problem:
PHASE I                      gap         feasibility                  L.S.
     objective   barrier    target     primal       dual    decrement   step
----------------------------------------------------------------------------
  1  +1.0e+000  +1.9e+001  9.8e+000   0.0e+000*  2.8e-001   2.0e+001   32.000
  2  +1.0e+000  -7.3e+001  9.8e+000   1.1e-014*  6.6e-001   2.9e+002   0.490
  3  -5.1e+000                        1.7e-014*
PHASE II                     gap         feasibility                  L.S.
     objective   barrier    target     primal       dual    decrement   step
----------------------------------------------------------------------------
  1  +1.6e+001  -9.2e+000  7.3e+001   4.0e-015*  3.0e-001   9.0e-001   1.000
  2  +1.6e+001  -9.6e+000  7.3e+001   1.7e-015*  5.7e-002   3.4e-002   1.000
  3  +1.6e+001  -9.7e+000  7.3e+001   1.6e-015*  2.7e-003   8.8e-005^  1.000
  4  +1.6e+001  +4.1e+003  7.3e-001   1.3e-015*  1.4e+000   3.8e+005   0.005
  5  +9.2e+000  +2.3e+003  7.3e-001   1.6e-015*  1.6e+000   9.5e+004   0.010
  6  +5.5e+000  +1.4e+003  7.3e-001   2.1e-015*  1.8e+000   1.7e+004   0.020
  7  +4.1e+000  +1.1e+003  7.3e-001   2.3e-015*  1.8e+000   3.4e+003   0.058
  8  +3.3e+000  +8.8e+002  7.3e-001   2.2e-015*  1.3e+000   2.0e+002   0.168
  9  +3.1e+000  +8.5e+002  7.3e-001   1.6e-015*  8.9e-001   2.4e+001   0.700
 10  +3.0e+000  +8.5e+002  7.3e-001   1.3e-015*  2.9e+000   4.7e+000   4.000
 11  +3.1e+000  +8.4e+002  7.3e-001   4.3e-015*  5.8e-001   3.6e+000   1.000
 12  +3.0e+000  +8.4e+002  7.3e-001   1.3e-015*  7.0e-001   8.1e-001   2.000
 13  +3.0e+000  +8.4e+002  7.3e-001   1.8e-015*  1.8e-001   2.0e-001   1.000
 14  +3.0e+000  +8.4e+002  7.3e-001   1.4e-015*  2.4e-002   2.7e-003^  1.000
 15  +3.0e+000  +7.7e+004  7.3e-003   7.3e-016*  2.0e+000   2.0e+005   0.007
 16  +3.0e+000  +7.6e+004  7.3e-003   1.1e-015*  2.0e+000   1.9e+004   0.028
 17  +2.9e+000  +7.5e+004  7.3e-003   1.6e-015*  1.6e+000   2.0e+002   0.168
 18  +2.9e+000  +7.5e+004  7.3e-003   1.5e-015*  1.0e+000   1.2e+001   0.700
 19  +2.9e+000  +7.5e+004  7.3e-003   6.9e-016*  1.1e+000   2.0e+000   2.000
 20  +2.9e+000  +7.5e+004  7.3e-003   1.1e-015*  1.5e-001   1.2e-001   1.000
 21  +2.9e+000  +7.5e+004  7.3e-003   1.5e-015*  1.6e-002   1.3e-003^  1.000
 22  +2.9e+000  +7.5e+006  7.3e-005   1.4e-015*  2.0e+000   1.9e+005   0.010
 23  +2.9e+000  +7.5e+006  7.3e-005   1.3e-015*  1.5e+000   1.7e+002   0.240
 24  +2.9e+000  +7.5e+006  7.3e-005   1.4e-015*  2.5e-001   1.4e+000   1.000
 25  +2.9e+000  +7.5e+006  7.3e-005   7.2e-016*  2.1e-001   4.1e-001   2.000
 26  +2.9e+000  +7.5e+006  7.3e-005   1.4e-015*  5.6e-002   4.1e-002   1.000
 27  +2.9e+000  +7.5e+006  7.3e-005   1.0e-015*  6.5e-003   4.0e-004^  1.000
 28  +2.9e+000  +7.5e+008  7.3e-007*  8.6e-016*  2.0e+000   1.9e+005   0.010
 29  +2.9e+000  +7.5e+008  7.3e-007*  1.5e-015*  1.6e+000   1.8e+002   0.240
 30  +2.9e+000  +7.5e+008  7.3e-007*  8.6e-016*  1.2e-001   7.4e-002   1.000
 31  +2.9e+000  +7.5e+008  7.3e-007*  3.1e-015*  8.2e-003   2.9e-004   1.000
 32  +2.9e+000  +7.5e+008  7.3e-007*  7.6e-015*  3.3e-005   4.2e-007   64.000
 33  +2.9e+000  +7.5e+008  7.3e-007*  6.5e-013*  2.1e-003   0.0e+000*

Gap reduced to below relative tolerance
Objective value: 2.94468467408
Variable values:
   x1 := -0.696770615848
   x2 := -0.999999954181
   x3 := -0.846465295022
   x4 := -0.999999952577
   x5 := 0.999999897447
Temporary values:
   @10 := 1.48167743052
   @11 := 0.411802980495
   @12 := 7.47069104167e-008
   @13 := 1.41203220248
   @14 := 3.22578402325e-009
   @15 := 1.91782222211e-008
   @16 := 5.24577794679e-009
   @17 := 0.742471406931
   @18 := 7.77392503635e-009
   @19 := 1.00628868731e-007
   @20 := 0.47795152988
   @21 := 8.5600873361e-009
   @22 := 2.30014866094
   @23 := 0.943385832085
   @24 := 1.803737348
   @25 := 3.09598523271e-008
   @42 := 0.303229384152
   @43 := 1.69677061585
   @44 := 4.58190352402e-008
   @45 := 1.99999995418
   @46 := 0.153534704978
   @47 := 1.84646529502
```

```
@48 := 4.74233096922e-008
@49 := 1.99999995258
@50 := 1.99999989745
@51 := 1.02552818158e-007
@26 := 0.375131406379
@1 := 2.94468467408
@34 := 1.4816774383
@27 := 0.00805363271806
@52 := 2.94468467408
@35 := 0.411803081124
@28 := 0.0125914164181
@53 := 2.94468467408
@36 := 0.477951604587
@29 := 0.324680521907
@54 := 2.94468467408
@37 := 1.41203221104
@30 := 1.40342798043
@55 := 2.94468467408
@38 := 2.30014866417
@31 := 0.0968256756975
@56 := 2.94468467408
@39 := 0.943385851263
@32 := 0.676771891577
@57 := 2.94468467408
@40 := 1.80373735324
@33 := 0.0472021489467
@58 := 2.94468467408
@41 := 0.742471437891
@2 := 1.48167742275
@3 := 0.411802879866
@4 := -0.477951455174
@5 := 1.41203219392
@6 := -2.30014865772
@7 := -0.943385812906
@8 := -1.80373734275
@9 := 0.742471375971
```

Bibliography

[1] M. Abdi, H. El Nahas, A. Jard, and E. Moulines. Semidefinite positive relaxation of the maximum-likelihood criterion applied to multiuser detection in a CDMA context. *IEEE Signal Processing Letters*, 9(6):165–167, June 2002.

[2] W. Achtziger, M. Bendsoe, A. Ben-Tal, and J. Zowe. Equivalent displacement based formulations for maximum strength truss topology design. *Impact of Computing in Science and Engineering*, 4(4):315–45, December 1992.

[3] F. Alizadeh. Interior point methods in semidefinite programming with applications to combinatorial optimization. *SIAM J. on Optimization*, 5(1):13–51, February 1995.

[4] B. Alkire and L. Vandenberghe. Convex optimization problems involving finite autocorrelation sequences. *Mathematical Programming*, Series A, 93:331–359, 2002.

[5] E. Andersen and Y. Ye. On a homogeneous algorithm for the monotone complementarity problem. *Mathematical Programming*, 84:375–400, 1999.

[6] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide.* SIAM, 1992.

[7] M. Avriel, R. Dembo, and U. Passy. Solution of generalized geometric programs. *International J. for Numerical Methods in Engineering*, 9:149–168, 1975.

[8] O. Bahn, J. Goffin, J. Vial, and O. Du Merle. Implementation and behavior of an interior point cutting plane algorithm for convex programming: An application to geometric programming. Working Paper, University of Geneva, Geneva, Switzerland, 1991.

[9] R. Banavar and A. Kalele. A mixed norm performance measure for the design of multirate filterbanks. *IEEE Trans. on Signal Processing*, 49(2):354–359, February 2001.

[10] A. Ben-Tal and M. Bendsoe. A new method for optimal truss topology design. *SIAM J. on Optimization*, 13(2), 1993.

[11] A. Ben-Tal, M. Kocvara, A. Nemirovski, and J. Zowe. Free material optimization via semidefinite programming: the multiload case with contact conditions. *SIAM Review*, 42(4):695–715, 2000.

[12] A. Ben-Tal and A. Nemirovski. Interior point polynomial time method for truss topology design. *SIAM J. on Optimization*, 4(3):596–612, August 1994.

[13] A. Ben-Tal and A. Nemirovski. Robust truss topology design via semidefinite programming. *SIAM J. on Optimization*, 7(4):991–1016, 1997.

[14] A. Ben-Tal and A. Nemirovski. Structural design via semidefinite programming. In *Handbook on Semidefinite Programming*, pages 443–467. Kluwer, Boston, 2000.

[15] M. Bendsoe, A. Ben-Tal, and J. Zowe. Optimization methods for truss geometry and topology design. *Structural Optimization*, 7:141–159, 1994.

[16] S. Benson. DSDP 4.5: A daul scaling algorithm for semidefinite programming. `http://www-unix.mcs.anl.gov/~benson/dsdp/`, March 2002.

[17] D. Bertsekas. *Nonlinear Programming.* Athena Scientific, Belmont, MA, 1995.

[18] D. Bertsekas, A. Nedic, and A. Ozdaglar. *Convex Analysis and Optimization.* Athena Scientific, Nashua, NH, 2004.

[19] D. Bertsimas and J. Nino-Mora. Optimization of multiclass queuing networks with changeover times via the achievable region approach: Part II, the multi-station case. *Mathematics of Operations Research*, 24(2), May 1999.

[20] C. Bischof, A. Carle, G. Corliss, A. Griewank, and P. Hovland. ADIFOR: Generating derivative codes from Fortran programs. *Scientific Programming*, pages 1–29, December 1991.

[21] P. Biswas and Y. Ye. Semidefinite programming for ad hoc wireless sensor network localization. Technical report, Stanford University, April 2004. `http://www.stanford.edu/~yyye/adhocn4.pdf`.

[22] B. Borchers. CDSP, a C library for semidefinite programming. *Optimization Methods and Software*, 11:613–623, 1999.

[23] J. Borwein and A. Lewis. Duality relationships for entropy-like minimization problems. *SIAM J. Control and Optimization*, 29(2):325–338, March 1991.

[24] S. Boyd and C. Barratt. *Linear Controller Design: Limits of Performance.* Prentice-Hall, 1991.

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

[26] S. Boyd, M. Hershenson, and T. Lee. Optimal analog circuit design via geometric programming, 1997. Preliminary Patent Filing, Stanford Docket S97-122.

[27] S. Boyd, S. Kim, L. Vandenberghe, and A. Hassibi. A tutorial on geometric programming. Technical report, Information Systems Laboratory, Stanford University, September 2004. `http://www.stanford.edu/~boyd/gp_tutorial.html`.

[28] S. Boyd and L. Vandenberghe. Semidefinite programming relaxations of non-convex problems in control and combinatorial optimization. In A. Paulraj, V. Roychowdhuri, , and C. Schaper, editors, *Communications, Computation, Control and Signal Processing: A Tribute to Thomas Kailath*, chapter 15, pages 279–288. Kluwer Academic Publishers, 1997.

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

[30] A. Brooke, D. Kendrick, A. Meeraus, and R. Raman. *GAMS: A User's Guide*. The Scientific Press, South San Francisco, 1998. `http://www.gams.com/docs/gams/GAMSUsersGuide.pdf`.

[31] R. Byrd, N. Gould, J. Nocedal, and R. Waltz. An active-set algorithm for nonlinear programming using linear programming and equality constrained subproblems. Technical Report OTC 2002/4, Optimization Technology Center, Northwestern University, October 2002.

[32] G. Calafiore and M. Indri. Robust calibration and control of robotic manipulators. In *American Control Conference*, pages 2003–2007, 2000.

[33] J. Chinneck. MProbe 5.0 (software package). `http://www.sce.carleton.ca/faculty/chinneck/mprobe.html`, December 2003.

[34] T. Coleman and A. Verma. Structure and efficient Jacobian calculation. In M. Berz, C. Bischof, G. Corliss, and A. Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 149–159. SIAM, Philadelphia, PA, 1996.

[35] A. Conn, N. Gould, D. Orban, and Ph. Toint. A primal-dual trust-region algorithm for non-convex nonlinear programming. *Mathematical Programming*, 87:215–249, 2000.

[36] A. Conn, N. Gould, and Ph. Toint. *LANCELOT: a Fortran Package for Large-Scale Nonlinear Optimization (Release A)*, volume 17 of *Springer Series in Computational Mathematics*. Springer Verlag, 1992.

[37] A. Conn, N. Gould, and Ph. Toint. *Trust-Region Methods*. Series on Optimization. SIAM/MPS, Philadelphia, 2000.

[38] C. Crusius. *A Parser/Solver for Convex Optimization Problems*. PhD thesis, Stanford University, 2002.

[39] M. Dahleh and I. Diaz-Bobillo. *Control of Uncertain Systems. A Linear Programming Approach*. Prentice Hall, 1995.

[40] G. B. Dantzig. *Linear Programming and Extensions.* Princeton University Press, 1963.

[41] T. Davidson, Z. Luo, and K. Wong. Design of orthogonal pulse shapes for communications via semidefinite programming. *IEEE Trans. on Communications*, 48(5):1433–1445, May 2000.

[42] J. Dawson, S. Boyd, M. Hershenson, and T. Lee. Optimal allocation of local feedback in multistage amplifiers via geometric programming. *IEEE J. of Circuits and Systems I*, 48(1):1–11, January 2001.

[43] C. de Souza, R. Palhares, and P. Peres. Robust H_∞ filter design for uncertain linear systems with multiple time-varying state delays. *IEEE Trans. on Signal Processing*, 49(3):569–575, March 2001.

[44] S. Dirkse and M. Ferris. The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems. *Optimization Methods and Software*, 5:123–156, 1995.

[45] A. Doherty, P. Parrilo, and F. Spedalieri. Distinguishing separable and entangled states. *Physical Review Letters*, 88(18), 2002.

[46] Y. Doids, V. Guruswami, and S. Khanna. The 2-catalog segmentation problem. In *Proceedings of SODA*, pages 378–380, 1999.

[47] C. Du, L. Xie, and Y. Soh. H_∞ filtering of 2-D discrete systems. *IEEE Trans. on Signal Processing*, 48(6):1760–1768, June 2000.

[48] H. Du, L. Xie, and Y. Soh. H_∞ reduced-order approximation of 2-D digital filters. *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 48(6):688–698, June 2001.

[49] R. Duffin. Linearizing geometric programs. *SIAM Review*, 12:211–227, 1970.

[50] G. Dullerud and F. Paganini. *A Course in Robust Control Theory*, volume 36 of *Texts in Applied Mathematics.* Springer-Verlag, 2000.

[51] B. Dumitrescu, I. Tabus, and P. Stoica. On the parameterization of positive real sequences and MA parameter estimation. *IEEE Trans. on Signal Processing*, 49(11):2630–2639, November 2001.

[52] J. Ecker. Geometric programming: methods, computations and applications. *SIAM Review*, 22(3):338–362, 1980.

[53] L. El Ghaoui, J.-L. Commeau, F. Delebecque, and R. Nikoukhah. LMITOOL 2.1 (software package). `http://robotics.eecs.berkeley.edu/~elghaoui/lmitool/lmitool.html`, March 1999.

[54] J.-P. A. Haeberly F. Alizadeh and M. Overton. Primal-dual interior-point methods for semidefinite programming: Convergence rates, stability and numerical results. *SIAM J. Optimization*, 8:46–76, 1998.

[55] U. Feige and M. Goemans. Approximating the value of two prover proof systems, with applications to MAX 2SAT and MAX DICUT. In *Proceedings of the 3rd Israel Symposium on Theory and Computing Systems*, pages 182–189, 1995.

[56] U. Feige and M. Langberg. Approximation algorithms for maximization problems arising in graph partitioning. *J. of Algorithms*, 41:174–211, 2001.

[57] U. Feige and M. Langberg. The rpr^2 rounding technique for semidefinite programs. In *ICALP*, Lecture Notes in Computer Science. Springer, Berlin, 2001.

[58] R. Fourer. Nonlinear programming frequently asked questions. `http://www-unix.mcs.anl.gov/otc/Guide/faq/nonlinear-programming-faq.html`, 2000.

[59] R. Fourer, D. Gay, and B. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, December 1999.

[60] R. Freund. Polynomial-time algorithms for linear programming based only on primal scaling and projected gradients of a potential function. *Mathematical Programming*, 51:203–222, 1991.

[61] E. Fridman and U. Shaked. A new H_∞ filter design for linear time delay systems. *IEEE Trans. on Signal Processing*, 49(11):2839–2843, July 2001.

[62] A. Frieze and M. Jerrum. Improved approximation algorithms for max k-cut and max bisection. *Algorithmica*, 18:67–81, 1997.

[63] Frontline Systems, Inc. Premium Solver Platform (software package). `http://www.solver.com`, September 2004.

[64] M. Fu, C. de Souza, and Z. Luo. Finite-horizon robust Kalman filter design. *IEEE Trans. on Signal Processing*, 49(9):2103–2112, September 2001.

[65] K. Fujisawa, M. Kojima, K. Nakata, and M. Yamashita. SDPA (Semi-Definite Programming Algorithm) user's manual—version 6.00. Technical report, Tokyo Insitute of Technology, July 2002.

[66] J. Geromel. Optimal linear filtering under parameter uncertainty. *IEEE Trans. on Signal Processing*, 47(1):168–175, January 1999.

[67] J. Geromel and M. De Oliveira. H_2/H_∞ robust filtering for convex bounded uncertain systems. *IEEE Trans. on Automatic Control*, 46(1):100–107, January 2001.

[68] P. Gill, W. Murray, and M. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM J. on Optimization*, 12:979–1006, 2002.

[69] P. Gill, W. Murray, M. Saunders, and M. Wright. User's guide for NPSOL 5.0: A FORTRAN package for nonlinear programming. Technical Report SOL 86-1, Systems Optimization Laboratory, Stanford University, July 1998. `http://www.sbsi-sol-optimize.com/manuals/NPSOL%205-0%20Manual.pdf`.

[70] P. Gill, W. Murray, and M. Wright. *Practical Optimization*. Academic Press, London, 1981.

[71] M. Goemans and D. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *J. of the ACM*, 42:1115–1145, 1995.

[72] D. Goldfarb and G. Iyengar. Robust portfolio selection problems. Technical report, Computational Optimization Research Center, Columbia University, March 2002. `http://www.corc.ieor.columbia.edu/reports/techreports/tr-2002-03.pdf`.

[73] D. Goldfarb and G. Iyengar. Robust quadratically constrained problems program. Technical Report TR-2002-04, Department of IEOR, Columbia University, New York, NY, 2002.

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

[75] C. Gonzaga. Path following methods for linear programming. *SIAM Review*, 34(2):167–227, 1992.

[76] H. Greenberg. Mathematical programming glossary, 1996–2004. `http://www.cudenver.edu/~hgreenbe/glossary`.

[77] A. Griewank and U. Naumann. Accumulating Jacobians as chained sparse matrix products. *Mathematical Programming*, 3(95):555–571, 2003.

[78] O. Güler and R. Hauser. Self-scaled barrier functions on symmetric cones and their classification. *Foundations of Computational Mathematics*, 2:121–143, 2002.

[79] D. Guo, L. Rasmussen, S. Sun, and T. Lim. A matrix-algebraic approach to linear parallel interference cancellation in CDMA. *IEEE Trans. on Communications*, 48(1):152–161, January 2000.

[80] L. Han, J. Trinkle, and Z. Li. Grasp analysis as linear matrix inequality problems. *IEEE Trans. on Robotics and Automation*, 16(6):663–674, December 2000.

[81] Q. Han, Y. Ye, and J. Zhang. An improved rounding method and semidefinite programming relaxation for graph partition. *Math. Programming*, 92:509–535, 2002.

[82] M. Hershenson, S. Boyd, and T. Lee. Optimal design of a CMOS op-amp via geometric programming. *IEEE Trans. on Computer-Aided Design*, January 2001.

[83] M. Hershenson, S. Mohan, S. Boyd, and T. Lee. Optimization of inductor circuits via geometric programming. In *Proceedings 36th IEEE/ACM Integrated Circuit Design Automation Conference*, 1999.

[84] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms I*, volume 305 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, New York, 1993.

[85] J.-B. Hiriart-Urruty and C. Lemaréchal. *Convex Analysis and Minimization Algorithms II: Advanced Theory and Bundle Methods*, volume 306 of *Grundlehren der mathematischen Wissenschaften*. Springer-Verlag, New York, 1993.

[86] P. Hovland, B. Norris, and C. Bischof. ADIC (software package), November 2003. `http://www-fp.mcs.anl.gov/adic/`.

[87] L. Huaizhong and M. Fu. A linear matrix inequality approach to robust H_∞ filtering. *IEEE Trans. on Signal Processing*, 45(9):2338–2350, September 1997.

[88] F. Jarre, M. Kocvara, and J. Zowe. Optimal truss design by interior point methods. *SIAM J. on Optimization*, 8(4):1084–1107, 1998.

[89] F. Jarre and M. Saunders. A practical interior-point method for convex programming. *SIAM J. on Optimization*, 5:149–171, 1995.

[90] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395, 1984.

[91] J. Keuchel, C. Schnörr, C. Schellewald, and D. Cremers. Binary partitioning, perceptual grouping, and restoration with semidefinite programming. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 25(11):1364–1379, November 2003.

[92] J. Kleinberg, C. Papadimitriou, and P. Raghavan. Segmentation problems. In *Proceedings of the 30th Symposium on Theory of Computation*, pages 473–482, 1998.

[93] M. Kocvara, J. Zowe, and A. Nemirovski. Cascading—an approach to robust material optimization. *Computers and Structures*, 76:431–442, 2000.

[94] M. Kojima, S. Mizuno, and A. Yoshise. An $O(\sqrt{n}L)$-iteration potential reduction algorithm for linear complementarity problems. *Mathematical Programming*, 50:331–342, 1991.

[95] K. Kortanek, X. Xu, and Y. Ye. An infeasible interior-point algorithm for solving primal and dual geometric progams. *Mathematical Programming*, 1:155–181, 1997.

[96] K. Kortanek, X. Xu, and Y. Ye. An infeasible interior-point algorithm for solving primal and dual geometric programs. *Mathematical Programming*, 76:155–182, 1997.

[97] J. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM J. of Optimization*, 11:796–817, 2001.

[98] J. Lasserre. Bounds on measures satisfying moment conditions. *Annals of Applied Probability*, 12:1114–1137, 2002.

[99] J. Lasserre. Semidefinite programming vs. LP relaxation for polynomial programming. *Mathematics of Operations Research*, 27(2):347–360, May 2002.

[100] H. Lebret and S. Boyd. Antenna array pattern synthesis via convex optimization. *IEEE Trans. on Signal Processing*, 45(3):526–532, March 1997.

[101] Lindo Systems, Inc. LINGO version 8.0 (software package). `http://www.lindo.com`, September 2004.

[102] Lindo Systems, Inc. What's Best! version 6.0 (software package). `http://www.lindo.com`, September 2004.

[103] M. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284:193–228, November 1998. Special issue on Signals and Image Processing.

[104] J. Löfberg. YALMIP version 2.1 (software package). `http://www.control.isy.liu.se/~johanl/yalmip.html`, September 2001.

[105] L. Lovasz. *An Algorithmic Theory of Numbers, Graphs and Convexity*, volume 50 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, Philadelphia, 1986.

[106] W. Lu. A unified approach for the design of 2-D digital filters via semidefinite programming. *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 49(6):814–826, June 2002.

[107] Z.-Q. Luo, J. Sturm, and S. Zhang. Duality and self-duality for conic convex programming. Technical report, Department of Electrical and Computer Engineering, McMaster University, 1996.

[108] S. Mahajan and H. Ramesh. Derandomizing semidefinite programming based approximation algorithms. *SIAM J. of Computing*, 28:1641–1663, 1999.

[109] M. Mahmoud and A. Boujarwah. Robust H_∞ filtering for a class of linear parameter-varying systems. *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 48(9):1131–1138, September 2001.

[110] The Mathworks, Inc. LMI control toolbox 1.0.8 (software package). `http://www.mathworks.com/products/lmi`, August 2002.

[111] The MathWorks, Inc. MATLAB (software package). `http://www.mathworks.com`, 2004.

[112] N. Megiddo. Pathways to the optimal set in linear programming. In N. Megiddo, editor, *Progress in Mathematical Programming : Interior Point and Related Methods*, pages 131–158. Springer Verlag, New York, 1989. Identical version in : *Proceedings of the 6th Mathematical Programming Symposium of Japan, Nagoya, Japan, 1-35, 1986.*

[113] C. Mészáros. On free variables in interior-point methods. Technical report, Imperial College, London, April 1997. `http://www.doc.ic.ac.uk/research/technicalreports/1997/DTR97-4.pdf`.

[114] M. Milanese and A. Vicino. Optimal estimation theory for dynamic systems with set membership uncertainty: An overview. *Automatica*, 27(6):997–1009, November 1991.

[115] G. Millerioux and J. Daafouz. Global chaos synchronization and robust filtering in noisy context. *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 48(10):1170–1176, October 2001.

[116] R. Monteiro and I. Adler. Interior path following primal-dual algorithms : Part I : Linear programming. *Mathematical Programming*, 44:27–41, 1989.

[117] J. Moré and D. Sorensen. NMTR (software package), March 2000. `http://www-unix.mcs.anl.gov/~more/nmtr`.

[118] MOSEK ApS. Mosek (software package). `http://www.mosek.com`, July 2001.

[119] B. Murtagh and M. Saunders. MINOS 5.5 user's guide. Technical report, Systems Optimizaiton Laboratory, Stanford University, July 1998. `http://www.sbsi-sol-optimize.com/manuals/Minos%205-5%20Manual.pdf`.

[120] S. Nash and A. Sofer. A barrier method for large-scale constrained optimization. *ORSA J. on Computing*, 5:40–53, 1993.

[121] I. Nenov, D. Fylstra, and L. Kolev. Convexity determination in the Microsoft Excel solver using automatic differentiation techniques. In *The 4th International Conference on Automatic Differentiation*, 2004.

[122] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*, volume 87 of *Applied Optimization*. Kluwer, Boston, 2004.

[123] Yu. Nesterov and A. Nemirovsky. A general approach to polynomial-time algorithms design for convex programming. Technical report, Central Economics and Mathematics Institute, USSR Acadamy of Science, Moscow, USSR, 1988.

[124] Yu. Nesterov and A. Nemirovsky. *Interior-Point Polynomial Algorithms in Convex Programming: Theory and Algorithms*, volume 13 of *Studies in Applied Mathematics*. SIAM Publications, Philadelphia, PA, 1993.

[125] Yu. Nesterov, O. Péton, and J.-Ph. Vial. Homogeneous analytic center cutting plane methods with approximate centers. In F. Potra, C. Roos, and T. Terlaky, editors, *Optimization Methods and Software*, pages 243–273, November 1999. Special Issue on Interior Point Methods.

[126] Yu. Nesterov and M. Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations Research*, 22:1–42, 1997.

[127] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, New York, 1999.

[128] D. Orban and R. Fourer. DrAmpl: a meta-solver for optimization. Technical report, École Polytechnique de Montréal, 2004.

[129] M. Overton and R. Womersley. On the sum of the largest eigenvalues of a symmetric matrix. *SIAM J. on Matrix Analysis and Applications*, 13(1):41–45, January 1992.

[130] R. Palhares, C. de Souza, and P. Dias Peres. Robust H_∞ filtering for uncertain discrete-time state-delayed systems. *IEEE Trans. on Signal Processing*, 48(8):1696–1703, August 2001.

[131] R. Palhares and P. Peres. LMI approach to the mixed H_2/H_∞ filtering design for discrete-time uncertain systems. *IEEE Trans. on Aerospace and Electronic Systems*, 37(1):292–296, January 2001.

[132] J. Park, H. Cho, and D. Park. Design of GBSB neural associative memories using semidefinite programming. *IEEE Trans. on Neural Networks*, 10(4):946–950, July 1999.

[133] P. Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical Programming*, Series B, 96(2):293–320, 2003.

[134] G. Pataki. Geometry of cone-optimization problems and semi-definite programs. Technical report, GSIA Carnegie Mellon University, Pittsburgh, PA, 1994.

[135] O. Péton and J.-P. Vial. A tutorial on ACCPM: User's guide for version 2.01. Technical Report 2000.5, HEC/Logilab, University of Geneva, March 2001. See also the `http://ecolu-info.unige.ch/~logilab/software/accpm/accpm.html`.

[136] S. Prajna, A. Papachristodoulou, and P. Parrilo. *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*, 2002. Available from `http://www.cds.caltech.edu/sostools` and `http://www.aut.ee.ethz.ch/parrilo/sostools`.

[137] B. Radig and S. Florczyk. *Evaluation of Convex Optimization Techniques for the Weighted Graph-Matching Problem in Computer Vision*, pages 361–368. Springer, December 2001.

[138] L. Rasmussen, T. Lim, and A. Johansson. A matrix-algebraic approach to successive interference cancellation in CDMA. *IEEE Trans. on Communications*, 48(1):145–151, January 2000.

[139] J. Renegar. A polynomial-time algorithm, based on Newton's method, for linear programming. *Mathematical Programming*, 40:59–93, 1988.

[140] M. Rijckaert and X. Martens. Analysis and optimization of the Williams-Otto process by geometric programming. *AIChE J.*, 20(4):742–750, July 1974.

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

[142] R. Rockafellar. *Convex Analysis.* Princeton Univ. Press, Princeton, New Jersey, second edition, 1970.

[143] C. Roos, T. Terlaky, and J.-Ph. Vial. *Interior Point Approach to Linear Optimization: Theory and Algorithms.* John Wiley & Sons, New York, NY, 1997.

[144] E. Rosenberg. *Globally Convergent Algorithms for Convex Programming with Applications to Geometric Programming.* PhD thesis, Department of Operations Research, Stanford University, 1979.

[145] U. Shaked, L. Xie, and Y. Soh. New approaches to robust minimum variance filter design. *IEEE Trans. on Signal Processing*, 49(11):2620–2629, November 2001.

[146] J. Stoer and C. Witzgall. *Convexity and Optimization in Finite Dimensions I.* Springer-Verlag, 1970.

[147] P. Stoica, T. McKelvey, and J. Mari. MA estimation in polynomial time. *IEEE Trans. on Signal Processing*, 48(7):1999–2012, July 2000.

[148] J. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11:625–653, 1999.

[149] J. A. K. Suykens, T. Van Gestel, J. De Brabanter, B. De Moor, and J. Vandewalle. *Least Squares Support Vector Machines.* World Scientific, Singapore, 2002.

[150] H. Tan and L. Rasmussen. The application of semidefinite programming for detection in CDMA. *IEEE J. on Selected Areas in Communications*, 19(8):1442–1449, August 2001.

[151] Z. Tan, Y. Soh, and L. Xie. Envelope-constrained H_∞ filter design: an LMI optimization approach. *IEEE Trans. on Signal Processing*, 48(10):2960–2963, October 2000.

[152] Z. Tan, Y. Soh, and L. Xie. Envelope-constrained H_∞ FIR filter design. *IEEE Trans. on Circuits and Systems II: Analog and Digital Signal Processing*, 47(1):79–82, January 2000.

[153] R. Tapia, Y. Zhang, and L. Velazquez. On convergence of minimization methods: Attraction, repulsion and selection. *J. of Optimization Theory and Applications*, 107:529–546, 2000.

[154] C. Tseng and B. Chen. H_∞ fuzzy estimation for a class of nonlinear discrete-time dynamic systems. *IEEE Trans. on Signal Processing*, 49(11):2605–2619, November 2001.

[155] T. Tsuchiya. A polynomial primal-dual path-following algorithm for second-order cone programming. Technical report, The Institute of Statistical Mathematics, Tokyo, Japan, October 1997.

[156] H. Tuan, P. Apkarian, and T. Nguyen. Robust and reduced-order filtering: new LMI-based characterizations and methods. *IEEE Trans. on Signal Processing*, 49(12):2975–2984, December 2001.

[157] H. Tuan, P. Apkarian, T. Nguyen, and T. Narikiyo. Robust mixed H_2/H_∞ filtering of 2-D systems. *IEEE Trans. on Signal Processing*, 50(7):1759–1771, July 2002.

[158] R. Tütüncü, K. Toh, and M. Todd. SDPT3—a MATLAB software package for semidefinite-quadratic-linear programming, version 3.0. Technical report, Carnegie Mellon University, August 2001.

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

[160] L. Vandenberghe, S. Boyd, and A. El Gamal. Optimal wire and transistor sizing for circuits with non-tree topology. In *Proceedings of the 1997 IEEE/ACM International Conference on Computer Aided Design*, pages 252–259, 1997.

[161] L. Vandenberghe, S. Boyd, and A. El Gamal. Optimizing dominant time constant in RC circuits. *IEEE Trans. on Computer-Aided Design*, 2(2):110–125, February 1998.

[162] R. Vanderbei. LOQO user's manual—version 4.05. Technical report, Operations Research and Financial Engineering, Princeton University, October 2000.

[163] R. Vanderbei and H. Benson. On formulating semidefinite programming problems as smooth convex nonlinear optimization problems. Technical Report ORFE-99-01, Operations Research and Financial Engineering, Princeton University, January 2000.

[164] F. Wang and V. Balakrishnan. Robust Kalman filters for linear time-varying systems with stochastic parametric uncertainties. *IEEE Trans. on Signal Processing*, 50(4):803–813, April 2002.

[165] S. Wang, L. Xie, and C. Zhang. H_2 optimal inverse of periodic FIR digital filters. *IEEE Trans. on Signal Processing*, 48(9):2696–2700, September 2000.

[166] J. Weickert and C. Schnörr. A theoretical framework for convex regularizers in PDE-based computation of image motion. *International J. of Computer Vision, Band 45*, 3:245–264, 2001.

[167] M. Wright. Some properties of the Hessian of the logarithmic barrier function. *Mathematical Programming*, 67:265–295, 1994.

[168] S. Wright. *Primal Dual Interior Point Methods.* SIAM Publications, Philadelphia, PA, 1999.

[169] S.-P. Wu and S. Boyd. SDPSOL: A parser/solver for semidefinite programs with matrix structure. In L. El Ghaoui and S.-I. Niculescu, editors, *Recent Advances in LMI Methods for Control*, chapter 4, pages 79–91. SIAM, Philadelphia, 2000.

[170] S.-P. Wu, S. Boyd, and L. Vandenberghe. FIR filter design via spectral factorization and convex optimization. In B. Datta, editor, *Applied and Computational Control, Signals, and Circuits*, volume 1, pages 215–245. Birkhauser, 1998.

[171] G. Xue and Y. Ye. An efficient algorithm for minimizing a sum of p-norms. *SIAM J. of Optimization*, 10(2):551–879, 2000.

[172] F. Yang and Y. Hung. Robust mixed H_2/H_∞ filtering with regional pole assignment for uncertain discrete-time systems. *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 49(8):1236–1241, August 2002.

[173] Y. Ye. Approximating quadratic programming with bound constraints. Technical report, Department of Management Science, The University of Iowa, March 1997. `http://www.stanford.edu/~yyye/bqp.ps`.

[174] Y. Ye. *Interior-Point Algorithms: Theory and Practice.* John Wiley & Sons, New York, NY, 1997.

[175] Y. Ye. A path to the Arrow-Debreu competitive market equilibrium. Technical report, Stanford University, February 2004. `http://www.stanford.edu/~yyye/arrow-debreu2.pdf`.

[176] Y. Ye, M. Todd, and S. Mizuno. An $O(\sqrt{n}L)$-iteration homogeneous and self-dual linear programming algorithm. *Mathematics of Operations Research*, 19(1):53–67, 1994.

[177] Y. Ye and J. Zhang. Approximation for dense-n/2-subgraph and the complement of min-bisection. *Manuscript*, 1999.

[178] S. Zhang. A new self-dual embedding method for convex programming. Technical report, Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, October 2001.

[179] S. Zhang. On conically ordered convex programs. Technical report, Department of Systems Engineering and Engineering Management, The Chinese University of Hong Kong, May 2003.

[180] H. Zhou, L. Xie, and C. Zhang. A direct approach to H_2 optimal deconvolution of periodic digital channels. *IEEE Trans. on Signal Processing*, 50(7):1685–1698, July 2002.

[181] M. Zibulevsky. Pattern recognition via support vector machine with computationally efficient nonlinear transform. Technical report, The University of New Mexico, Computer Science Department, 1998. `http://iew3.technion.ac.il/~mcib/nipspsvm.ps.gz`.

[182] J. Zowe, M. Kocvara, and M. Bendsoe. Free material optimization via mathematical programming. *Mathematical Programming*, 9:445–466, 1997.

[183] U. Zwick. Outward rotations: a tool for rounding solutions of semidefinite programming relaxations, with applications to max cut and other problems. In *Proceedings of the 31th Symposium on Theory of Computation*, pages 679–687, 1999.