USING LOQO TO SOLVE SECOND-ORDER CONE PROGRAMMING PROBLEMS

ROBERT J. VANDERBEI AND HANDE YURTTAN

Statistics and Operations Research
Princeton University
SOR-98-9

ABSTRACT. Many nonlinear optimization problems can be cast as second-order cone programming problems. In this paper, we discuss a broad spectrum of such applications. For each application, we consider various formulations, some convex some not, and study which ones are amenable to solution using a general-purpose interior-point solver LOQO. We also compare with other commonly available nonlinear programming solvers and special-purpose codes for second-order cone programming.

1. INTRODUCTION.

Second-order cone programming (SOCP) refers to optimization problems having the form

$$\begin{array}{ll} \text{minimize} & f^T x \\ \text{subject to} & \|A_i x + b_i\| \le c_i^T x + d_i, \quad i = 1, \ldots, m, \end{array}$$

where $x \in \mathbb{R}^n$ is the optimization variable, and all other quantities, $f \in \mathbb{R}^n$, $A_i \in \mathbb{R}^{k_i \times n}$, $b_i \in \mathbb{R}^{k_i}$, $c_i \in \mathbb{R}^n$, and $d_i \in \mathbb{R}$, are data. The norm is the Euclidean norm: $\|u\| = \left(u^T u\right)^{1/2}$. Constraints of the form

$$\|Ax + b\| \le c^T x + d$$

are called *second-order cone constraints* because the affinely defined variables $u = Ax + b \in \mathbb{R}^k$ and $t = c^T x + d \in \mathbb{R}$ are constrained to belong to the *second-order cone* defined by

$$\{(u, t) \in \mathbb{R}^{k+1} : \|u\| \le t\}.$$

1991 *Mathematics Subject Classification.* Primary 90C30, Secondary 49M37, 65K05.

Key words and phrases. Nonlinear programming, interior-point methods, convex optimization, second-order cone programming.

Research of the first author supported by NSF grants CCR-9403789 and DMS-9870317 and by ONR grant N00014-98-1-0036.

A recent paper of Lobo, et al., [15], describes an impressive collection of applications that can be formulated as SOCP problems. In this paper, we describe how to solve these problems and others using LOQO, which is software that implements an interior-point method for general (nonconvex) nonlinear programming problems ([26]). The *nonlinear programming* problems handled by LOQO can be expressed as follows:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & h_i(x) \le 0, \quad i = 1, \ldots, m, \end{array} \tag{1}$$

where f and each of the h_i's are real-valued functions on $\mathbb{R}^n$.

According to the analysis given in [26], LOQO should converge to a globally optimal solution when the function f and all of the h_i's are convex and twice continuously differentiable. In the case of SOCP, f is linear and each

$$h_i(x) = \|A_i x + b_i\| - c_i^T x - d_i$$

is convex but is not differentiable on the affine set $A_i x + b_i = 0$.

The nondifferentiability of $h_i(x)$ could potentially pose a problem for any solver that relies on the computation of derivatives of the constraint functions. But, in practice, one would only expect trouble

(1) if an intermediate solution happens to be at a place of nondifferentiability or
(2) if the optimal solution is at a point of nondifferentiability.

With interior-point methods (and perhaps others), the first case is easy to address by simply randomizing the initial solution. Then the probability of encountering an intermediate solution at which a derivative fails to exist is zero. The second case is more serious. When we consider specific application areas later in this paper, one of our primary considerations will be this nondifferentiability at optimality issue. In the next section, we shall give an example that illustrates exactly what fails in the case of interior-point algorithms.

2. Interior-Point Methods for Nonlinear Programming.

In this section, we first describe the interior-point method implemented in LOQO and then we address the issue of nondifferentiability.

2.1. The Interior Point Algorithm. The general interior-point paradigm implemented in LOQO is described as follows. First, add slack variables w_i to each of the constraints in (1), reformulating the problem as

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & h(x) + w = 0, \\ & \qquad w \ge 0, \end{array} \tag{2}$$

where $h(x)$ and w represent the vectors with elements $h_i(x)$ and w_i, respectively. We then eliminate the inequality constraints in (2) by placing them in a barrier term, resulting in the classical Fiacco-

McCormick [6] logarithmic barrier problem

$$\begin{aligned} &\text{minimize} \quad f(x) - \mu \sum_{i=1}^{m} \log(w_i) \\ &\text{subject to} \ \ h(x) + w = 0. \end{aligned} \tag{3}$$

The Lagrangian for this problem is

$$L_\mu(x, w, y) = f(x) - \mu \sum_{i=1}^{m} \log(w_i) + y^T (h(x) + w),$$

and the first-order KKT conditions for a minimum are

$$\begin{aligned} \nabla_x L &= \nabla f(x) + \nabla h(x)^T y = 0, \\ \nabla_w L &= -\mu W^{-1} e + y \quad\ = 0, \\ \nabla_y L &= h(x) + w \quad\ \ = 0, \end{aligned} \tag{4}$$

where W is the diagonal matrix with elements w_i, e is the vector of all ones, and $\nabla h(x)$ is the Jacobian matrix of the vector $h(x)$. We now modify (4) by multiplying the second equation by W, producing the standard primal-dual system

$$\begin{aligned} \nabla f(x) + \nabla h(x)^T y &= 0, \\ -\mu e + WYe &= 0, \\ h(x) + w &= 0, \end{aligned} \tag{5}$$

where Y denotes the diagonal matrix with elements y_i. Note that the second equation implies that y is nonnegative, which is consistent with the fact that y is the vector of Lagrange multipliers associated with what were originally inequality constraints.

The basis of the numerical algorithm for finding a solution to the primal-dual system (5) is Newton's method, which is well known to be very efficient for linear and convex quadratic programming. In order to simplify notation and at the same time highlight connections with linear and quadratic programming, we introduce the following definitions:

$$H(x, y) = \nabla^2 f(x) + \sum_{i=1}^{m} y_i \nabla^2 h_i(x)$$

and

$$A(x) = \nabla h(x).$$

The Newton system for (5) is then

$$\begin{bmatrix} H(x,y) & 0 & A(x)^T \\ 0 & Y & W \\ A(x) & I & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta w \\ \Delta y \end{bmatrix} = \begin{bmatrix} -\nabla f(x) - A(x)^T y \\ \mu e - WYe \\ -h(x) - w \end{bmatrix}. \tag{6}$$

The system (6) is not symmetric, but is easily symmetrized by multiplying the second equation by W^{-1}, yielding

$$\begin{bmatrix} H(x,y) & 0 & A^T(x) \\ 0 & W^{-1}Y & I \\ A(x) & I & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta w \\ \Delta y \end{bmatrix} = \begin{bmatrix} -\sigma \\ \gamma \\ -\rho \end{bmatrix}, \tag{7}$$

where

$$\sigma = \nabla f(x) + A^T(x)y, \tag{8}$$

$$\gamma = \mu W^{-1}e - y, \tag{9}$$

$$\rho = w + h(x). \tag{10}$$

Note that ρ measures *primal infeasibility*. By analogy with linear programming, we refer to σ as *dual infeasibility*. Also, note that σ, γ, and ρ depend on x, y, and w even though we don't show this dependence explicitly.

It is (7), or a small modification of it, that LOQO solves at each iteration to find the search directions Δx, Δw, Δy. Since the second equation can be used to eliminate Δw without producing any off-diagonal fill-in in the remaining system, one normally does this elimination first. Hence, Δw is given by

$$\Delta w = WY^{-1}(\gamma - \Delta y)$$

and the resulting *reduced KKT system* is given by

$$\begin{bmatrix} H(x,y) & A^T(x) \\ A(x) & -WY^{-1} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = -\begin{bmatrix} \sigma \\ \rho + WY^{-1}\gamma \end{bmatrix}. \tag{11}$$

The algorithm then proceeds iteratively from an initial point $x^{(0)}$, $w^{(0)}$, $y^{(0)}$ through a sequence of points determined from the search directions described above:

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \alpha^{(k)}\Delta x^{(k)}, \\ w^{(k+1)} &= w^{(k)} + \alpha^{(k)}\Delta w^{(k)}, \\ y^{(k+1)} &= y^{(k)} + \alpha^{(k)}\Delta y^{(k)}. \end{aligned} \tag{12}$$

For linear programming different steplengths, $\alpha_{\mathrm{p}}^{(k)}$ and $\alpha_{\mathrm{d}}^{(k)}$, are used with the primal and dual directions, whereas for nonlinear programming a common steplength is employed. A merit function is used to choose the steplengths. See [26] for further details.

2.2. Nondifferentiability at Optimality. As mentioned in the introduction, nondifferentiability of the square root function at optimality can pose difficulties for interior-point optimizers. Here's the simplest example illustrating the difficulty:

$$\begin{aligned} &\text{minimize} && ax_1 + x_2 \\ &\text{subject to} && |x_1| \le x_2. \end{aligned} \tag{13}$$

The parameter a is a given real number satisfying $-1 < a < 1$. The optimal solution is easily seen to be $(0, 0)$. Part of the stopping rule for the interior-point algorithm described above is the attainment

of dual feasibility; that is, $\sigma = \nabla f(x) + A^T(x)y = 0$. For the problem under consideration, the condition for dual feasibility is

$$\begin{bmatrix} a \\ 1 \end{bmatrix} + \begin{bmatrix} \operatorname{sgn}(x_1) \\ -1 \end{bmatrix} y = 0. \tag{14}$$

We have written $\operatorname{sgn}(x_1)$ for the derivative of $|x_1|$, but what about $x_1 = 0$? Here, any value between -1 and 1 is a valid subgradient. But a specific choice must be selected *a priori*, since solvers don't work with set-valued functions. For lack of a better choice, suppose that we pick 0 as a specific subgradient of the absolute value function at the origin. That is, we adopt the common convention that $\operatorname{sgn}(0) = 0$. From the second equation in (14), we see that $y = 1$. Therefore, the first equation reduces to $\operatorname{sgn}(x_1) = a$. But, at optimality, $x_1 = 0$ and so in order to get dual feasibility we must have that

$$\operatorname{sgn}(0) = a.$$

Of course, if solvers worked with set-valued functions, then $\operatorname{sgn}(0)$ would be the interval $[-1, 1]$ and this condition would be $a \in [-1, 1]$ and there would be no problem. But for now solvers work only with singly valued functions and the particular value we picked for $\operatorname{sgn}(0)$, namely 0, might not be equal to a in which case an interior-point method can't possibly produce a dual feasible solution.

Note that there are two important properties at work in this example:

(1) the constraint function failed to be differentiable at the optimal point, and
(2) that constraint's inequality was tight at optimality, thereby forcing the corresponding dual variable to be nonzero.

2.3. Smoothing by Perturbation. We tried using LOQO to solve problem (13) and, as expected, it failed. However, when we replace $|x_1|$ with $(\epsilon^2 + x_1^2)^{1/2}$, it solves easily even with small values for ϵ^2 such as 10^{-8}. In fact, for this problem, the exact solution to the perturbed problem can be computed by hand: $x_1 = -a\epsilon/\sqrt{1-a^2}$, $x_2 = \epsilon/\sqrt{1-a^2}$, and $y = 1$. This perturbation trick turns out to work quite well in general as we shall see in the applications section.

3. Alternate Formulations.

The perturbation technique of the previous section is an example of replacing a bad problem formulation with a better one that is almost, but not quite, equivalent to the original. It turns out that there are techniques in which one can replace a bad problem (i.e., not convex, not smooth, etc.) formulation with an equivalent but better one. In this section, we consider a few alternatives for second-order cone constraints.

3.1. The Second-Order Cone. Consider the second-order cone constraint

$$h(x) := \|Ax + b\| - c^T x - d \leq 0.$$

The convexity and smoothness properties of h derive from the corresponding properties of

$$\phi(u, t) = \|u\| - t.$$

The gradient and Hessian of ϕ are easy to compute:

$$\nabla\phi = \|u\|^{-1}\begin{bmatrix} u \\ \|u\| \end{bmatrix}, \qquad \nabla^2\phi = \|u\|^{-3}\begin{bmatrix} u^T u I - uu^T & 0 \\ 0 & 0 \end{bmatrix}.$$

The fact that the upper left block of the Hessian is positive semidefinite follows immediately from the definition of semidefiniteness together with the Cauchy-Schwarz inequality. The only place of nonsmoothness of ϕ is at $u = 0$.

3.2. Smoothing by Squaring. Our aim to replace $\|u\| - t \le 0$ with equivalent inequalities. The simplest possibility is to use

$$\begin{aligned} \|u\|^2 - t^2 &\le 0 \\ t &\ge 0. \end{aligned}$$

The corresponding function that gets composed into the nonlinear program then is

$$\gamma(u,t) = \|u\|^2 - t^2.$$

This function is smooth everywhere. But it is not convex as its Hessian clearly shows:

$$\nabla\gamma = 2\begin{bmatrix} u \\ -t \end{bmatrix}, \qquad \nabla^2\gamma = 2\begin{bmatrix} I & 0 \\ 0 & -1 \end{bmatrix}.$$

Even though the feasible region is convex, its representation as the intersection of nonconvex inequalities can lead to slow convergence to dual feasibility, as described in [26]. Therefore, one would expect this reformulation not to work well.

3.3. Convexification by Exponentiation. The function γ given above almost did the trick. If we can find a smooth convex function that maps the negative halfline into the negative halfline, perhaps composing it with γ will produce the function we need. With this motivation, let

$$\psi(u,t) = e^{(\|u\|^2 - t^2)/2} - 1.$$

Like γ, this function is smooth everywhere. It also turns out to be convex:

$$\begin{aligned} \nabla\psi &= e^{(\|u\|^2-t^2)/2}\begin{bmatrix} u \\ -t \end{bmatrix}, \\ \nabla^2\psi &= e^{(\|u\|^2-t^2)/2}\begin{bmatrix} I + uu^T & -tu \\ -tu^T & 1+t^2 \end{bmatrix} \\ &= e^{(\|u\|^2-t^2)/2}\left(I + \begin{bmatrix} u \\ -t \end{bmatrix}\begin{bmatrix} u^T & -t \end{bmatrix}\right). \end{aligned}$$

The second expression for the Hessian clearly shows that $\nabla^2\psi$ is positive definite.

Introducing the exponential function to get both smoothness and convexity is fine mathematically, but on the practical side there are significant dangers if the scale of the variables varies too widely. Indeed, $\exp(\|u\|^2)$ is huge even when $\|u\|$ is on the order of 10. In the following section, we shall confirm that this approach rarely works in practice.

3.4. Convexification by Ratios. We ran into the trouble of nonconvexity during smoothing by squaring. One way to fix this problem is to use

$$\begin{aligned} \frac{\|u\|^2}{t} - t &\leq 0 \\ t &\geq 0. \end{aligned}$$

Letting

$$\eta(u,t) = \frac{\|u\|^2}{t} - t,$$

we can verify the convexity of this new formulation:

$$\begin{aligned} \nabla\eta &= \begin{bmatrix} 2ut^{-1} \\ -\|u\|^2t^{-2} - 1 \end{bmatrix}, \\ \nabla^2\eta &= 2t^{-1}\begin{bmatrix} I & -ut^{-1} \\ -u^Tt^{-1} & \|u\|^2t^{-2} \end{bmatrix} \\ &= 2t^{-1}\begin{bmatrix} I \\ -u^Tt^{-1} \end{bmatrix}\begin{bmatrix} I & -ut^{-1} \end{bmatrix}. \end{aligned}$$

The second expression for the Hessian clearly shows that it is positive definite. The only place of nonsmoothness of η is at $t = 0$.

4. APPLICATIONS.

In this section, we consider several applications and, for each application, one or more ways to formulate the problem. All of the models to be mentioned have been encoded in a mathematical programming language called AMPL. This language is described in detail in [7]. The AMPL models mentioned in this section are available over the internet by visiting [24].

Our focus in this section is primarily on robustness, that is, which formulations work and which ones don't. In situations where more than one formulation can be used successfully, the question then becomes one of efficiency. Efficiency issues are addressed in the next section.

4.1. Antenna array weight design. The problem here is to determine how to combine the outputs from each antenna in an array of n antennae so that the resulting aggregate output signal has certain desired characteristics. For concreteness, we assume that the antennae are evenly spaced along the y-axis (in a 2-dimensional world). We also assume that the input signal has a certain fixed wavelength λ but that it can arrive from any direction θ. For certain values of θ, the so-called *side-lobe* S, we wish to attenuate the signal as much as possible whereas for other values, those in the *main-lobe* M, we

wish to preserve some given desired output profile. The optimization problem then is

$$\begin{array}{lll} \text{minimize} & s & \\ \text{subject to} & |G(\theta)|^2 \le s, & \theta \in S \\ & |G(\theta)|^2 \le u(\theta)^2, & \theta \in M \\ & G(\theta) = G_0(\theta), & \theta \in P \\ & G(\theta) = \displaystyle\sum_{k=1}^{n} w_k \exp\left(-i\frac{2\pi}{\lambda} y_k \sin\theta\right), & \theta \in S \cup M. \end{array}$$

Here, $i = \sqrt{-1}$, y_k is the position of the k-th antenna on the y-axis, the $u(\theta)$'s are given upper bounds for the $|G(\theta)|$'s when $\theta \in M$, and the $G_0(\theta)$'s are given desired values for the $G(\theta)$'s when θ belongs to a small finite set P of angles in M. The variables in the model are the scalar s, the weights w_k, $k = 1, \ldots, n$, and the array outputs $G(\theta)$, $\theta \in S \cup M$. It is important to note that the w_k's and the $G(\theta)$'s are complex numbers. Generally speaking, the sets S and M are intervals of the real line and so the model involves an infinite set of constraints. To make a model with a finite number of constraints, one approximates the intervals with finite samples of evenly spaced angles.

The model just given is clearly a convex nonlinear programming problem. The nonlinear functions that describe the constraints are quadratic and convex. LOQO should work well on such problems. A particular instance, *antenna.mod*, was created in which the discretization used 0.5° increments, $n = 24$, $\lambda = 2$, the y_k's were spaced 1.12 units apart, the side lobe S was the interval from -3.5° to 0°, the main lobe M was the interval from -20° to 60° minus the side lobe, and P consisted of four angles: 10°, 15°, 20°, and 26°. The upper bound profile $u(\theta)$ was obtained from H. Lebret and matches the data used in the experiments described in [13]. LOQO has trouble reaching 8 digits of agreement between the primal and dual solution, but it does find a primal/dual pair showing 7 figures of agreement. It takes 55 iterations to obtain this solution.

The model, as presented, is not a second-order cone problem. In [15], Lobo, et al., suggest an equivalent formulation in which s is replaced with $t = \sqrt{s}$ in the model. After the replacement, one uses the monotonicity of the square function (on the nonnegative halfline) to replace the objective function t^2 with t. Also, the constraint

$$|G(\theta)|^2 \le t^2 \tag{15}$$

is replaced with

$$|G(\theta)| \le t \tag{16}$$

which is a second-order cone constraint. These modifications are encoded in an AMPL model called *antenna_socp.mod*. LOQO finds a feasible primal/dual pair of solutions having 6 digits of agreement in their objectives after 44 iterations and then runs into numerical difficulties and can't make further progress.

We also tried two other experiments. *Antenna_nonconvex.mod* is identical to the AMPL model *antenna_socp.mod* except that the second-order cone constraint (16) is replaced with (15). LOQO was

unable to solve this nonconvex variant of the model. Finally, we tried replacing (16) with

$$\exp\left(|G(\theta)|^2 - t^2\right) \leq 1$$

but, again, LOQO could not solve it. We don't understand why LOQO had trouble with this last formulation other than that maybe the exponential function introduced numbers that were large and the algorithm got into numerical trouble. This last variant is encoded in an AMPL file called *antenna_exp.mod*.

To sum up, the first (convex) version of this problem works the best with the SOCP variant a close second.

4.2. Grasping force optimization. The problem is to find the smallest amount of normal force required to grasp an object with a set of m robot fingers. The fingers exert contact forces at given points $p_1, \ldots, p_m \in \mathbb{R}^3$ on the object, which is assumed to be rigid (but fragile). Let $v_i \in \mathbb{R}^3$ denote the inwardly pointing unit vector normal to the surface at the i-th contact point and let F_i denote the force applied there. The normal component of force F_i is given by $v_i v_i^T F_i$. The problem is to minimize the maximum normal component of force subject to force and torque balance constraints and the constraint that the tangential component of each contact force can be no larger in magnitude than a constant μ (the coefficient of friction) times the normal component:

$$\begin{aligned}
&\text{minimize} && t \\
&\text{subject to} && v_i^T F_i \leq t, \quad i = 1, \ldots, m, \\
& && \sum_{i=1}^{m} F_i = -F_{\text{ext}} \\
& && \sum_{i=1}^{m} p_i \times F_i = -T_{\text{ext}} \\
& && \|(I - v_i v_i^T) F_i\| \leq \mu v_i^T F_i, \quad i = 1, \ldots, m.
\end{aligned}$$

In the constraints, F_{ext} denotes an externally applied force acting at the center of mass and T_{ext} denotes an externally applied torque acting about the origin.

The specific problem we considered involves supporting a parabolic cone whose lower surface is described by the equation

$$p(3) = p(1)^2 + p(2)^2$$

(we denote the components of vector p by $(p(1), p(2), p(3))$). There are five robot fingers equally spaced around an off-centered circle of radius 1 in the horizontal coordinate plane and extending upward until contacting the surface of the cone.

We created an AMPL model called *grasp.mod* for the problem instance just described. LOQO is unable to solve this problem. The reason is that one of the five forces is zero at optimality and therefore the nondifferentiability of the Euclidean norm $\|\cdot\|$ becomes a problem.

Grasp_eps.mod is identical to *grasp.mod* except that the Euclidean norm is perturbed slightly to make it differentiable everywhere:

$$\|u\|_\epsilon = \left(\epsilon^2 + \sum_{j=1}^{3} u(j)^2\right)^{1/2}.$$

With $\epsilon^2 = 10^{-8}$, LOQO solves the problem in 38 iterations. From the solution to this model we note that indeed one of the five contact points applies no force at optimality.

In *grasp_nonconvex.mod*, the friction constraints are rewritten as

$$\|(I - v_i v_i^T)F_i\|^2 \le (\mu v_i^T F_i)^2, \quad i = 1, \ldots, m.$$

In addition, the following nonnegativity constraints are added:

$$v_i^T F_i \ge 0, \quad i = 1, \ldots, m.$$

As discussed in the Section 3.2, the resulting constraint functions are no longer convex. LOQO is unable to solve this variant. It finds a primal feasible solution that is close (3 digits of agreement) to the optimal solution found by *grasp_eps.mod* but is unable to get a corresponding dual feasible solution.

In *grasp_exp.mod*, the friction constraints from *grasp_nonconvex.mod* are rewritten as

$$\exp\left(\|(I - v_i v_i^T F_i\|^2 - (\mu v_i^T F_i)^2\right) \le 1, \quad i = 1, \ldots, m.$$

LOQO is also unable to solve this problem to the level of precision stipulated by its default stopping rule. But it does find a primal feasible solution and a dual feasible solution for which there are 5 digits of agreement between the objective functions. For many real-word applications, this level of precision is good enough. It took, however, 270 iterations to get this solution!

Related work on "grasping" models can be found in [5, 4].

To sum up, for the grasp models, ϵ-perturbation of the original second-order cone constraints seems to work best.

4.3. FIR filter design. In this section, we consider the minimax dB linear phase lowpass finite impulse response (FIR) filter design problem as described in [15]:

$$\begin{aligned}
\text{minimize} \quad & t \\
\text{subject to} \quad & 1/t \le 2\sum_{k=0}^{n/2-1} h_k \cos((k-(n-1)/2)\omega) \le t, \quad 0 \le \omega \le \omega_p, \\
& -\beta \le 2\sum_{k=0}^{n/2-1} h_k \cos((k-(n-1)/2)\omega) \le \beta, \quad \omega_s \le \omega \le \pi, \\
& t \ge 1.
\end{aligned}$$

Here, n is an even integer, β is a small positive number, and $0 < \omega_p < \omega_s < \pi$. The variables are the coefficients h_k, $k = 0, \ldots, n/2-1$. The problem, as stated, has an infinite number of constraints. Consequently, we discretize the frequency parameter ω to obtain an approximation having only a finite

number of constraints. In the specific instance we considered, we chose $n = 20$, $\beta = 0.01$, $\omega_p = 90^\circ$, $\omega_s = 120^\circ$, and we discretized the set of ω's using 1° increments. This problem does not involve second-order cone constraints. In fact, the only nonlinearity is the $1/t$ appearing in the first set of constraints. It is, however, a nonlinear programming problem. In fact, constraints of the form

$$1/t \le u$$

are passed to LOQO with the function $h(u,t) = 1/t - u$ which is convex and twice continuously differentiable on $\{(u,t) : t > 0\}$. Therefore, one would expect LOQO to solve this problem easily and in fact it does finding a solution with 8 digits of agreement between primal and dual objective values in only 29 iterations. This model can be found in *fir_convex.mod*.

To convert the filter design problem to one with second-order cone constraints, Lobo, et al., [15] suggest introducing one new scalar variable, u, replacing the first set of constraints with

$$u \le 2 \sum_{k=0}^{n/2-1} h_k \cos((k - (n-1)/2)\omega) \le t, \quad 0 \le \omega \le \omega_p,$$

and adding one new (second-order cone) constraint:

$$\sqrt{4 + (u-t)^2} \le u + t,$$

which is equivalent to the requirement that $1/t \le u$. The resulting model, *fir_socp.mod*, also solves easily with LOQO requiring only 30 iterations to achieve the same convergence criteria. One reason that it is easy to solve is that the argument of the square root is bounded below by 4 and therefore nonsmoothness issues don't arise. Most importantly, however, the Hessian in *fir_socp.mod* is easier to compute than in *fir_convex.mod*, since we replaced a group of nonlinear constraints with one nonlinear constraint and a group of linear constraints. The ratio reformulation, *fir_ratio.mod* performs just as well, reaching the optimal solution in 34 iterations.

Furthermore information on this subject can be found in [20, 3]. In particular, the interior-point approach to the convex optimization version of the problem is discussed in [28].

To sum up, all versions of this problem are easily solved by LOQO, but the SOCP version is perhaps the best.

4.4. Portfolio optimization. Consider a collection of investments (or investment categories) J. Let μ_j denote the expected return on a \$1 investment after one year's investment in $j \in J$ and let $\sigma_{jj'}$ denote the covariance between the returns for investments j and j' (these quantities are considered given—they are usually gleaned from historical data). The aim is to decide, for each investment j, what fraction x_j of one's assets to invest in j. The overall annual expected return, referred to as the *reward*, for the portfolio defined by a specific choice of the x_j's is given by $\sum_j \mu_j x_j$ and the variance of the return, referred to as the *risk*, is $\sum_{jj'} x_j \sigma_{jj'} x_{j'}$. The Markowitz model [17] for portfolio optimization involves optimizing a linear combination of risk and reward. Letting $\lambda > 0$

denote a parameter used for the linear combination, the model can be written as follows:

$$\begin{aligned} \text{minimize} \quad & -\sum_j \mu_j x_j + \lambda \sum_{jj'} x_j \sigma_{jj'} x_{j'} \\ \text{subject to} \quad & \sum_j x_j = 1 \\ & x_j \geq 0, \qquad \text{for all } j \in J. \end{aligned} \tag{17}$$

As λ varies over the positive reals, the optimal portfolios represent different levels of the tradeoff between risk and reward. The one parameter family of optimal portfolios so obtained is referred to as the *efficient frontier*. Typically one solves the problem for several different values of λ and picks the solution that seems most appealing.

A drawback to the above approach is that it is hard to predict how the portfolios will change as λ changes and therefore it is hard to home in on a desirable value. For this reason, one often formulates the problem as the minimization of the risk subject to a lower bound $r_{\min}$ on the reward,

$$\begin{aligned} \text{minimize} \quad & \sum_{jj'} x_j \sigma_{jj'} x_{j'} \\ \text{subject to} \quad & \sum_j \mu_j x_j \geq r_{\min} \\ & \textstyle\sum_j x_j = 1 \\ & x_j \geq 0, \qquad \text{for all } j \in J, \end{aligned} \tag{18}$$

or as the maximization of reward subject to a cap $s_{\max}$ on the risk,

$$\begin{aligned} \text{maximize} \quad & \sum_j \mu_j x_j \\ \text{subject to} \quad & \sum_{jj'} x_j \sigma_{jj'} x_{j'} \leq s_{\max} \\ & \textstyle\sum_j x_j = 1 \\ & x_j \geq 0, \qquad \text{for all } j \in J. \end{aligned} \tag{19}$$

Formulations (17) and (18) are convex quadratic programming problems (i.e., with linear constraints) whereas (19) is a quadratically-constrained linear-objective optimization problem. All of these models should be easy to solve with LOQO. In fact, using real data we created three AMPL models, *optriskreward.mod*, *optrisk.mod*, and *optreward.mod* for problems (17), (18), and (19), respectively. The three models were each solved by LOQO in 16, 19, and 18 iterations, respectively.

Each of the above three models could, in principle, be converted to second-order cone programming problems. But, for the two that are quadratic programming problems, such a reformulation seems like a step in the wrong direction. Hence, we only made a second-order cone programming variant of (19) by simply taking the square root of the quadratic constraints. The resulting AMPL model is called *optreward_socp.mod*. LOQO solves this problem in 18 iterations.

4.5. Truss design. Given a set of n linear elastic bars (also referred to as *elements*) connecting a set of m nodes and a limit v on the total volume of material available, an important problem in struc-

tural optimization is to determine the cross section x_e for each element that yields the stiffest overall structure (relative to a given applied load) (see [2]).

Externally applied loads act at some or all of the nodes. Let f denote the vector of applied loads. Note that it is a vector of forces, that is, an m-vector of 3-vectors (or 2-vectors for planar trusses). Associated with each element e is a rank-one positive semidefinite matrix $K_e = a_e a_e^T$ called the *element stiffness matrix*. The vector a_e is very sparse having only 6 nonzero elements for spatial trusses and 4 nonzeros for planar trusses. The stiffness matrix for the structure built with element cross sectional areas x_e is then given by

$$K(x) = \sum_e x_e K_e.$$

To maximize the stiffness of the structure under load f, we minimize the elastic energy $f^T K(x)^{-1} f$. Hence the optimization problem can be formulated as follows:

$$\begin{array}{lll} \text{minimize} & f^T K(x)^{-1} f & \\ \text{subject to} & \sum_e l_e x_e \le v & \\ & x_e \ge 0, & \text{for all } e. \end{array} \tag{20}$$

Here, l_e denotes the length of member e so that the product $l_e x_e$ represents the volume of the member.

In this formulation, the problem is not a second-order cone programming problem. But, even as a nonlinear programming problem, there is at least one significant disadvantage to the model as posed. Namely, the matrix $K(x)$ is exceedingly sparse but the matrix $K(x)^{-1}$ might be completely dense. Since these models tend to be very large, one is motivated to find equivalent formulations that don't involve the inverse of $K(x)$. It turns out that there are many such models (see, e.g., [2]) including ones having linear objectives and convex quadratic constraints and, in the case of one scenario of external loads as considered here, even ones that are simply linear programming problems.

In [15], an equivalent second-order cone programming problem is given. We present here the details of essentially the same equivalence. Since it is easier to work backwards, we start by stating a problem that is almost a second-order cone programming problem:

$$\begin{array}{lll} \text{minimize} & \sum_e t_e & \\ \text{subject to} & \sum_e y_e a_e = f & \\ & y_e^2 \le t_e x_e & \\ & t_e \ge 0, & \text{for all } e \\ & \sum_e l_e x_e \le v & \\ & x_e \ge 0, & \text{for all } e. \end{array} \tag{21}$$

Here, the y_e's and the t_e's are new scalar variables. We start by showing that (21) is equivalent to (20). To this end, note first that each variable t_e will make the inequality $y_e^2 \le t_e x_e$ tight at optimality.

Therefore, we can replace this inequality with an equality and eliminate the t_e's from the problem:

$$\begin{aligned} \text{minimize} \quad & \sum_e \frac{y_e^2}{x_e} \\ \text{subject to} \quad & \sum_e y_e a_e = f \\ & \textstyle\sum_e l_e x_e \le v \\ & x_e \ge 0, \qquad \text{for all } e. \end{aligned}$$

Now, we can optimize first over the y_e variables and then over the x_e's. Since the y_e's are not involved in any of the inequality constraints, this inner minimization can be carried out explicitly with the help of a vector λ of Lagrange multipliers. The Lagrangian for the inner problem is

$$L(y,\lambda) = \sum_e \frac{y_e^2}{x_e} + \lambda^T \left(\sum_e y_e a_e - f \right).$$

Setting all derivatives of L to zero, we get

$$\begin{aligned} \frac{2y_e}{x_e} + \lambda^T a_e &= 0 \quad \text{for all } e \\ \sum_e y_e a_e - f &= 0. \end{aligned}$$

Solving the first set of equations for the y_e's and then eliminating them from the second set, we find that

$$\lambda = -2K(x)^{-1} f$$

and then that

$$y_e = x_e a_e^T K(x)^{-1} f.$$

Therefore, after the inner optimization, the objective function becomes

$$\sum_e \frac{y_e^2}{x_e} = \sum_e x_e f^T K(x)^{-1} a_e a_e^T K(x)^{-1} f = f^T K(x)^{-1} f$$

and we see that (21) is in fact equivalent to (20).

To convert (21) into a second-order cone programming problem, one simply needs to replace each inequality

$$y_e^2 \le t_e x_e \tag{22}$$

with its second-order cone equivalent:

$$\left(4y_e^2 + (t_e - x_e)^2\right)^{1/2} \le t_e + x_e. \tag{23}$$

AMPL model *structure_socp.mod* encodes a specific instance known as the Michel truss [19]. See, e.g., [25] for details of this instance. LOQO solves this problem in 42 iterations. An $\epsilon^2 = 10^{-8}$

perturbation to the argument of the square root function gives a model *structure_socp_eps.mod* that solves in 36 iterations. Another convex formulation, *structure_ratio.mod* is solved just as quickly. In *structure_nonconvex.mod*, constraints (22) take the place of (23). With this model, LOQO finds a primal feasible solution but converges very slowly toward optimality and never seems to find a reasonable dual solution.

Problem (42) in [2] gives linear programming formulation of the problem. AMPL model *structure2.mod* implements this formulation. LOQO found an optimal solution to this model in 17 iterations.

Problem (41) in [2] gives a formulation in which the objective function is linear and the constraints are convex quadratic. AMPL model *structure3.mod* implements this formulation. LOQO found an optimal solution to this model in 61 iterations.

4.6. Equilibrium of a system of piecewise linear springs. The problem here is to find the shape of a hanging chain in which each of the N links is a spring that buckles under compression but exerts a restoring force proportional to its elongation when under tension. Each node has a weight hanging from it (the springs are assumed to have neglible mass). Specifically, the problem is to minimize the potential energy of the system subject to given boundary conditions. After some preliminary model reformulation (see [15]), the problem can be written as

$$\begin{aligned} \text{minimize} \quad & \textstyle\sum_j m_j g y_j + \frac{k}{2}\|t\|^2 \\ \text{subject to} \quad & \|p_j - p_{j-1}\| - l_0 \le t_j, \quad j = 1, \ldots, N, \\ & p_0 = a \\ & p_N = b \\ & t \ge 0. \end{aligned}$$

Here, g is the acceleration to due gravity, k is the stiffness constant for the springs, l_0 is the rest length of each spring, m_j is the mass of the j-th weight, $p_j = (x_j, y_j)$ denote the position vector of the j-th node, a is the position of one end point of the chain, b is the position of the other end point, $t = (t_1, \ldots, t_N)$, and finally t_j is an upper bound on the spring energy of the j-th spring. The unknowns are the p_j's and the t_j's.

In *springs.mod*, we chose $N = 100$, $g = 9.8$, $k = 100$, $m_j = 1$, $a = (0, 0)$, $b = (2, -1)$, and $l_0 = 2\|b - a\|/N$. Although the constraints are either linear or second-order cone constraints, the problem, as formulated, is not a second-order cone programming problem since the objective is quadratic. However, it is a general nonlinear programming problem and therefore can be attacked by LOQO. In fact, the objective is convex quadratic and the nondifferentiability of the second-order cone constraints only occurs at points where two adjacent nodes occupy exactly the same position in space, which should not happen in practice. Hence, one would expect LOQO to be able to solve this problem without trouble and indeed that is the case. LOQO finds an optimal solution in just 16 iterations.

To convert the problem to a second-order cone programming problem, Lobo, et al., suggest adding one new scalar variable, y, to replace $\|t\|^2$ in the objective function and then adding one extra second-order cone constraint

$$\sqrt{(2\|t\|)^2 + (1 - y)^2} \le 1 + y$$

(which is equivalent to the condition that y is an upper bound on $\|t\|^2$). We did not make such a model since with LOQO it is completely unnecessary to linearize the objective function.

We did, however, try two other experiments. In *springs_nonconvex.mod*, we replaced the second-order cone constraint in *springs.mod* with the nonconvex but smooth substitute

$$\|p_j - p_{j-1}\|^2 - (l_0 + t_j)^2 \le 0.$$

LOQO solves this nonconvex nonlinear optimization problem to the correct optimal solution in 15 iterations. Lastly, we replaced this nonconvex quadratic constraint with a convex exponential variant:

$$\exp\left(\|p_j - p_{j-1}\|^2 - (l_0 + t_j)^2\right) \le 1.$$

The resulting convex model, found in *springs_exp.mod*, was solved by LOQO in 19 iterations.

4.7. Euclidean single facility location. Let $a_1, \ldots, a_m$ denote points in $\mathbb{R}^d$ representing the coordinates of m existing facilities. The problem is to locate a new facility so as to minimize a weighted sum of the distances between the new facility and each of the existing ones:

$$\text{minimize} \quad \sum_{i=1}^{m} w_i \|x - a_i\|.$$

A particular instance of this problem can be found in *esfl.mod*. This instance involves 1000 existing facilities on the plane. LOQO solves this problem in 10 iterations.

These problems can be formulated as second-order cone programming problems as follows:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m} w_i t_i \\ \text{subject to} \quad & \|x - a_i\| \le t_i, \quad i = 1, \ldots, m. \end{aligned}$$

We modified *esfl.mod* to make a version in this form. The AMPL model *esfl_socp.mod* contains this problem. LOQO solves this problem in 12 iterations. Each iteration, however, is slower than before so overall the first formulation seems better.

For facility location problems nondifferentiability can be an issue. The fact that LOQO solved the instance described above was really just luck. To illustrate what can go wrong, consider the case of three existing facilities. When $m = 3$, the problem is called *Fermat's problem* (see [27, 12]). In this case, the optimal solution can be described precisely. Namely, if the three points are such that the triangle formed by them has no angle of 120° or larger, then the optimal location of the new facility is at the unique point in the triangle that makes a 120° angle with each pair of vertices. If, on the other hand, one of the angles does exceed 120°, then the optimal solution is at that vertex. We made two AMPL models for Fermat's problem. In *fermat.mod*, the optimal solution is in the interior of the triangle whereas in *fermat2.mod* , the optimal solution is at a vertex.

LOQO solves *fermat.mod* in 8 iterations, whereas it is unable to solve *fermat2.mod*. However, an $\epsilon^2 = 10^{-8}$ perturbation to *fermat2.mod*, called *fermat2_eps.mod* solves in 16 iterations.

We also formulated these problems as SOCP's. LOQO solves *fermat_socp.mod* in 8 iterations, but cannot solve *fermat2_socp.mod*. The ϵ-perturbed model, *fermat_socp_eps.mod* is solved in 17 iterations.

For related work, see [1, 11, 16, 21, 29].

4.8. Euclidean multiple facility location. The Euclidean multifacility location problem generalizes the single facility model of the previous subsection. Here instead of adding one new facility, we consider adding n new ones. The objective is to minimize a weighted sum of the distances between each old and each new facility plus a weighted sum of the distances between each pair of new facilities:

$$\text{minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}\|x_j - a_i\| + \sum_{j=1}^{n}\sum_{j'=1}^{j-1} v_{jj'}\|x_j - x_{j'}\|.$$

A particular instance of this problem can be found in *emfl.mod*. This instance involves 200 existing facilities on the plane and 25 new ones. LOQO is unable to solve this problem in 200 iterations. Therefore, we made an $\epsilon^2 = 10^{-8}$ perturbation to the model and called the perturbed model *emfl_eps.mod*. This version solves in 17 iterations.

We also considered a second-order cone programming variant of the problem:

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{m}\sum_{j=1}^{n} w_{ij}s_{ij} + \sum_{j=1}^{n}\sum_{j'=1}^{j-1} v_{jj'}t_{jj'} \\ \text{subject to} \quad & \|x_j - a_i\| \le s_{ij}, \qquad 1 \le i \le m, \quad 1 \le j \le n, \\ & \|x_j - x_{j'}\| \le t_{jj'}, \qquad 1 \le j < j' \le n. \end{aligned}$$

LOQO fails to solve an instance of this problem *emfl_socp.mod* because of nondifferentiability problems. However, an $\epsilon^2 = 10^{-8}$ perturbation solves in 23 iterations, and the ratio reformulation *emfl_socp_ratio.mod* solves in 31 iterations. We also tried a nonconvex variant and an exponential variant using the methods of Sections 3.2 and 3.3 but neither of these variant proved solvable.

4.9. Steiner points. Given a set of n points $p_1, \ldots, p_n$ in $\mathbb{R}^2$, the Steiner tree problem is to find the shortest planar straight-line graph that spans the set of points. The solution is always a tree, called the *Steiner tree*. It includes all of the given points plus some extra vertices, called *Steiner points*. There are at most $n - 2$ Steiner points. This problem is NP-hard [18, 8]. However, for a given topology, computing the position of the Steiner points and the associated length of the tree is a tractible problem that can be solved as a second-order cone programming problem.

We define a *full Steiner topology* to be any tree graph that contains the original set of points plus a full complement of $n - 2$ Steiner points for which the degree of each of the original points is 1 and the degree of each of the Steiner points is 3. Given a full Steiner topology, let $p_{n+1}, \ldots, p_{2n-2}$ denote the Steiner points and let $\mathcal{A}$ denote the arcs in the graph. Then the *full Steiner topology problem* is defined as follows:

$$\text{minimize} \quad \sum_{(i,j)\in\mathcal{A}} \|p_i - p_j\|.$$

Here, the original points $p_1, \ldots, p_n$ are fixed whereas the Steiner points $p_{n+1}, \ldots, p_{2n-2}$ are optimization variables. This problem was first studied in [9, 10, 22].

The AMPL model *steiner.mod* encodes Example 1 given in [29]. LOQO is unable to solve this base model. The reason is that some of the Steiner points collapse onto existing points and so some of the distances in the objective function become zero. Hence, the nondifferentiability of the absolute

value function again is a problem. However, an $\epsilon^2 = 10^{-8}$ perturbation *steiner_eps.mod* solves in 81 iterations.

As we've done before, we reformulated the problem as a second-order cone programming problem, *steiner_socp.mod*, and we also made an $\epsilon^2 = 10^{-8}$ perturbation thereof, *steiner_socp_eps.mod* and a ratio reformulation, *steiner_socp_ratio.mod*. LOQO is unable to solve the straight second-order cone programming problem but easily solves the perturbed problem in 30 iterations and the ratio problem in 42 iterations.

LOQO cannot solve the nonconvex variant, *steiner_nonconvex.mod*, but is successful with the exponential variant, *steiner_exp.mod*, solving it in 195 iterations.

4.10. Minimal surfaces. Given a simply connected compact domain D in $\mathbb{R}^2$ with piecewise smooth boundary ∂D and a real-valued function γ defined on ∂D, the problem is to find a function u defined throughout D that agrees with γ on the boundary of D and has minimal surface area:

$$\begin{array}{ll} \text{minimize} & \displaystyle\iint_D \sqrt{1+\left(\frac{\partial u}{\partial x}\right)^2+\left(\frac{\partial u}{\partial y}\right)^2}\,dx\,dy \\ \text{subject to} & u(x,y)=\gamma(x,y), \quad (x,y)\in\partial D. \end{array} \tag{24}$$

To obtain a problem with a finite number of variables and constraints, we simply discretize the domain D using a rectangular lattice of grid points and then approximate the derivatives by finite differences. The resulting problem is a convex nonlinear optimization problem, but it is not in the format of a second-order cone programming problem. The ampl model *minsurf.mod* gives an instance of the problem in which the domain D is a square that is discretized into a 32×32 grid and the boundary function γ is a concave parabola on each of the four sides separately. LOQO solves this convex optimization problem in just 12 iterations.

It is easy reformulate the problem as a second-order cone program. To do it, rewrite (24) as

$$\begin{array}{ll} \text{minimize} & \displaystyle\iint_D t(x,y)\,dx\,dy \\ \text{subject to} & \displaystyle\sqrt{1+\left(\frac{\partial u}{\partial x}\right)^2+\left(\frac{\partial u}{\partial y}\right)^2} \le t(x,y), \quad (x,y)\in D, \\ & u(x,y)=\gamma(x,y), \quad (x,y)\in\partial D. \end{array}$$

and then discretize in the same way as before. Since the argument of the square root is bounded below by 1, LOQO is able to solve this problem. It takes 34 iterations. We should note, however, that each iteration here is more involved than before so that the solution time is about 200 times greater.

We also created a nonconvex variant *minsurf_nonconvex.mod* and a convex exponential variant *minsurf_exp.mod* as described in Sections 3.2 and 3.3. The nonconvex variant was solved in 82 iterations, but LOQO was able to find an optimal solution to the exponential variant in the default of 200 iterations.

5. Comparisons with other nonlinear programming optimizers.

As already mentioned, every model discussed in the previous section has been encoded in AMPL and is available on the internet [24]. For solver comparisons, we used LOQO (version 3.17), MINOS

(version 5.4), SNOPT (version 5.3-2), and LANCELOT (version 19980617). The experiments were run on a 100 MHz SGI workstation with the MIPS R4000 processor and 96 MBytes of memory. Each solver was called directly from AMPL (version 19980722) with AMPL's presolve option enabled.

Solution times in cpu-seconds are shown in Table 1. In the table, an asterisk is shown whenever a problem was not solved to the full satisfaction of the solver's stopping rule. In many of these cases, the solver did find a primal feasible solution whose objective value matched the correct answer to several digits. The typical reason for a stopping rule to fail was that dual feasibility was not attained (also refered to as the reduced gradient being large). In this case, LOQO usually ran the default maximum number (200) of iterations and stopped. MINOS and SNOPT had better bail-out options and would terminate with an assertion that the solution appears to be optimal although the reduced gradient remains large. Nonetheless, for every solver there were several instances in which the solver just ran the maximum number of iterations specified for that solver and stopped. Hence, for these problems the solution times obtained were rather meaningless and quite a bit worse than what would be obtained had the problem been rerun with options set to relax the stopping rule appropriately. For this reason, we feel it is not relevant to show those solution times.

6. COMPARISON WITH SPECIAL PURPOSE OPTIMIZERS.

The models discussed were also coded in MATLAB and tested using LOQO's MATLAB interface, the Second-Order Cone Programming special purpose optimizer SOCP by Boyd et al. [14], and the Semidefinite Programming special purpose optimizer SEDUMI by Sturm [23].

For these comparisons, we used LOQO (Version 4.01), SOCP (Beta Version), and SEDUMI (Version 1.02), on a 180MHz SGI server with a MIPS R10000 processor and 512 MBytes of memory. SOCP and SEDUMI are, by design, called directly from MATLAB. While a MATLAB-LOQO interface exists for linear and quadratic programming, no such interface exists for second-order cone programming. Therefore, we used MATLAB to write an AMPL data file and invoke AMPL, which then called LOQO with this data and a generic second-order cone programming model. In this scheme, LOQO's abilities to handle linear constraints, a quadratic objective function and variable bounds were not used in the comparison with SOCP, but linear constraints were incorporated into the models for the comparison with SEDUMI. Finally, SEDUMI's built-in matrix ordering routines were used instead of LIPSOL by Zhang [30], since our system has MATLAB versions 4 and 5 and uses a 64-bit OS (problems noted by Zhang).

Solution times in cpu-seconds are shown in Table 2 and Table 3. Again, an asterisk is shown whenever a problem was not solved to the full satisfaction of the solver's stopping rule. The double asterisk in *antenna_socp* and in *grasp_eps* indicates that SOCP returned a solution that was not optimal. When there is an "*eps*" at the end of the filename, the CPU times provided are for LOQO called directly from AMPL and LOQO used with MATLAB using the ϵ-perturbed version of the AMPL model. The ϵ-perturbation was also implemented in the grasping force model for SOCP but the solver still returned a solution that was not optimal. For SEDUMI, no such perturbation was used on any of the problems.

For SOCP, the stopping rule was adjusted to give the same level of convergence as LOQO. The accuracy of all problems are as discussed previously in the text, with the exception of *minsurf*, where SOCP was unable to achieve a 10^{-8} relative tolerance. Therefore, the CPU times recorded in Table

Problem	Constraints	Variables	SOCP?	LOQO	MINOS	SNOPT	LANCELOT
antenna	166	49		21.36	*	*	*
antenna_socp	167	49	Y	16.65	*	*	*
antenna_ratio	167	49	Y	*	*	*	*
antenna_nonconvex	166	49	Y	*	*	*	*
antenna_exp	166	49	Y	57.85	*	*	*
grasp.mod	16	16	Y	*	*	*	*
grasp_eps.mod	16	16	Y	0.21	0.25	0.25	0.32
grasp_nonconvex.mod	21	16	Y	0.11	0.70	*	1.56
grasp_exp.mod	21	16	Y	0.19	*	*	1.19
fir_convex	243	11		0.99	1.38	1.52	11.58
fir_socp	244	12	Y	0.89	0.96	1.04	9.27
fir_ratio	244	12	Y	1.00	1.08	1.08	11.05
optriskreward.mod	1	8		0.04	0.02	0.04	0.03
optrisk.mod	2	8		0.05	0.02	0.06	0.10
optreward.mod	2	8		0.04	0.09	0.25	0.12
optreward_socp.mod	2	8	Y	0.05	0.06	0.10	0.05
structure2	1910	176		4.04	21.39	23.09	497.75
structure3	960	176		39.87	29.54	125.45	3043.75
structure_socp	1145	2886	Y	127.71	*	*	*
structure_socp_eps	1145	2886	Y	105.11	*	*	*
structure_ratio	1145	2886	Y	122.51	*	*	*
structure_nonconvex	1145	2886	Y	*	*	*	*
structure_exp	1145	2886	Y	*	*	*	*
springs.mod	10	28	Y	0.07	0.30	0.53	0.21
springs_nonconvex.mod	10	28	Y	0.06	0.95	0.69	0.36
springs_exp.mod	10	28	Y	0.09	*	0.70	0.36
esfl.mod	0	2		0.41	0.28	0.25	1.70
esfl_socp.mod	1000	1002	Y	9.07	1.73	18.70	2.31
fermat.mod	0	2		0.01	0.01	0.02	0.02
fermat_socp.mod	1	3	Y	0.02	0.04	0.03	0.02
fermat2.mod	0	2		*	*	*	0.11
fermat2_eps.mod	0	2		0.02	0.01	0.04	0.05
fermat2_socp.mod	1	3	Y	*	*	*	*
fermat2_socp_eps.mod	1	3	Y	0.03	0.06	0.08	0.04
emfl.mod	0	50		*	*	*	*
emfl_eps.mod	0	50		2.09	5.99	1.71	3.80
emfl_socp.mod	5300	5350	Y	*	*	*	*
emfl_socp_eps.mod	5300	5350	Y	753.01	*	3368.51	*
emfl_nonconvex.mod	5300	5350	Y	*	*	*	*
emfl_exp.mod	5300	5350	Y	*	*	*	*
emfl_ratio.mod	5300	5350	Y	868.79	*	*	*
steiner.mod	0	16		*	*	*	*
steiner_eps.mod	0	16		1.06	0.14	0.22	0.29
steiner_socp.mod	17	33	Y	*	*	*	*
steiner_socp_eps.mod	17	33	Y	0.16	2.36	0.73	0.58
steiner_ratio.mod	17	33	Y	0.25	*	*	5.86
steiner_nonconvex.mod	17	33	Y	*	*	*	*
steiner_exp.mod	17	33	Y	*	*	*	*
minsurf.mod	0	961		4.91	688.95	*	32.33
minsurf_socp.mod	2046	3009	Y	90.96	5634.04	*	*
minsurf_nonconvex2.mod	2048	3009	Y	731.75	*	*	*
minsurf_exp.mod	2048	3009	Y	*	*	*	*
minsurf_ratio.mod	2048	3009	Y	107.7	6856.60	*	*

TABLE 1. Solution times in seconds.

	AMPL			MATLAB			
Problem	Constraints	Variables	LOQO	Constraints	Variables	LOQO	SOCP
antenna_socp	167	49	4.21	187	59	9.03	**
grasp_eps	16	16	0.05	22	22	0.11	**
fir_socp	243	5	0.11	307	5	0.12	1.58
optreward_socp	2	8	0.01	3	9	0.12	0.05
structure_socp	155	336	0.67	442	357	2.35	*
springs	10	28	0.02	15	37	0.07	*
esfl_socp	300	302	0.44	300	302	0.47	136.70
fermat_socp_eps	3	5	0.004	3	5	0.004	0.03
fermat2_socp	0	2	0.004	3	5	0.008	0.02
emfl_socp_eps	86	94	0.12	172	94	0.12	18.62
steiner_socp	17	33	*	37	73	2.91	10.55
minsurf_socp	70	97	0.07	100	149	0.18	68.83

TABLE 2. Solution times in seconds for LOQO vs. SOCP. * indicates that the solver was unable to find an optimal solution, and ** indicates that the solution returned by the solver was not optimal.

	Constraints	Variables	SigFigs	LOQO	SEDUMI
antenna_socp	177	49	6	28.6	*
grasp_eps	22	16	9	0.05	0.25
fir_socp	307	5	8	0.12	0.39
optreward_socp	3	8	12	0.014	0.27
structure_socp	421	336	8	1.08	4.54
springs	11	29	12	0.03	0.33
esfl_socp	300	302	13	0.62	1.45
fermat_socp_eps	3	5	12	0.005	0.24
fermat2_socp	3	5	10	0.008	0.26
emfl_socp_eps	86	94	11	0.13	0.51
steiner_socp	17	33	12	0.05	0.39
minsurf_socp	70	97	10	0.08	0.60

TABLE 3. Solution times in seconds for LOQO vs. SEDUMI. * indicates that the solver was unable to find an optimal solution

2 reflect 6 figures of agreement between the primal and the dual solutions. For the comparison with SEDUMI, LOQO's stopping rule was adjusted to agree with the number of significant figures reported by SEDUMI. These figures are reported in Table 3.

In both comparisons, the instances of the problems *emfl*, *esfl*, *fir*, *springs*, and *minsurf* we consider are actually smaller than those reported in the previous section. There are three reasons for this difference. First, the setup of these problems by MATLAB takes a long time, about 7 hours for the original truss problem. Secondly, SOCP's Phase I algorithm is quite inefficient on large problems and unable to solve feasibility problems of more than a few hundred variables. Lastly, SOCP treats all matrices as dense, whereas LOQO and SEDUMI exploit sparsity, which could also account for the large differences in runtimes.

For LOQO, we provided the same initial point that was given in the original AMPL models, and when there was no solution provided, LOQO's MATLAB interface assigned a randomized point to avoid nondifferentiability at 0 (the default initial solution provided by AMPL).

We believe that two of the most important differences between the use of the optimizers were in the modelling environment and the syntax of the models, namely the expression of equality constraints. The modelling language AMPL provided a straightforward implementation of the models, whereas for-loops had to be used in MATLAB to assemble the data, and for-loops are known to be very inefficient.

Figure 1 in the Appendix illustrates set definition in both environments. Our MATLAB implementation uses matrices, in this case four-dimensional matrices. This setup takes up a large amount of memory without a sparse implementation and is quite slow due to the multiply embedded for-loops. AMPL, on the other hand, offers a more abstract set notation, which is also quite efficient.

Figure 2 in the Appendix illustrates how to define parameters over these sets. Again, our MATLAB implementation is quite slow.

The codes in Figure 3 in the Appendix are for the load constraints in the structural optimization example. The MATLAB version uses for-loops to assemble the coefficient matrix, instead of the more efficient indexing. The constraints that are being assembled in that figure correspond to a group of equality constraints, which is another source of difficulty in using SOCP and MATLAB. SOCP required us to break up these constraints as follows:

$$f(x) = a$$

became

$$\begin{aligned} f(x) - a + w &\geq 0 \\ -f(x) + a &\geq 0 \\ w &\geq 0. \end{aligned}$$

Thus, for each equality constraint, we end up with two extra constraints and one extra slack variable. These slack variables are added to keep the rows of the coefficient matrix linearly independent, and a large contribution from each of the slacks is added to the objective function to guarantee that their values go to zero, giving us equalities. In SEDUMI, the equality constraints had to be broken up into two inequalities, without the need for a slack variable.

7. CONCLUSIONS

Of all the models discussed, only the grasp and spring models are naturally formulated as second-order cone programming problems. All other models have equivalent simpler convex formulations that generally solve more efficiently. The original (SOCP) formulation of the spring model solves easily as is. For the grasp model, ϵ-perturbation provides an effective technique to obtain practical solutions.

In general, LOQO, a general nonlinear programming solver, was able to solve second-order cone programming problems more efficiently than special purpose optimizers SOCP and SEDUMI.

Acknowledgements. The ratio reformulation for second-order cone constraints was suggested to us by Lieven Vandenberghe who credits Erling Andersen with the idea. We would like to thank Michael Todd for carefully reading an early draft of the paper and suggesting a number of improvements.

REFERENCES

[1] K.D. Andersen, E. Christiansen, A.R. Conn, and M.L. Overton. An efficient primal-dual interior-point method for minimizing a sum of euclidean norms. Technical report, NYU Comp. Sci. Dept., 1998.

[2] M.P. Bendsøe, A. Ben-Tal, and J. Zowe. Optimization methods for truss geometry and topology design. *Structural Optimization*, 7:141–159, 1994.

[3] S. Boyd and C. Barratt. *Linear Controller Design: Limits of Performance*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[4] M. Buss, L. Faybusovich, and J.B. Moore. Recursive algorithms for real-time grasping force optimization. In *Proceedings of International Conference on Robotics and Automation*, 1997.

[5] M. Buss, H. Hashimoto, and J.B. Moore. Dextrous hand grasping force optimization. *IEEE Trans. on Robotics and Automation*, 12:406–418, 1996.

[6] A.V. Fiacco and G.P. McCormick. *Nonlinear Programming: Sequential Unconstrainted Minimization Techniques*. Research Analysis Corporation, McLean Virginia, 1968. Republished in 1990 by SIAM, Philadelphia.

[7] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Scientific Press, 1993.

[8] E.N. Gilbert and H.O. Pollak. Steiner minimal trees. *SIAM J. on Applied Mathematics*, 16:1–29, 1968.

[9] F.K. Hwang. A linear time algorithm for full steiner trees. *Operations Research Letters*, 4:235–237, 1986.

[10] F.K. Hwang and J.F. Weng. The shortest network under a given topology. *J. of Algorithms*, 13:468–488, 1992.

[11] H.W. Kuhn. On a pair of dual nonlinear programs. In J. Abadie, editor, *Nonlinear Programming*, pages 39–54. North-Holland, 1967.

[12] H.W. Kuhn. A note on fermat's problem. *Mathematical Programming*, 4:98–107, 1973.

[13] H. Lebret and S. Boyd. Antenna array pattern synthesis via convex optimization. *IEEE Transactions on Signal Processing*, 45:526–532, 1997.

[14] M.S. Lobo, L. Vandenberghe, and S. Boyd. Socp:software for second-order cone programming—a user's guide. Electrical Engineering Department, Stanford University, Stanford, CA 94305, 1997.

[15] M.S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. Technical report, Electrical Engineering Department, Stanford University, Stanford, CA 94305, 1998. To appear in *Linear Algebra and Applications* special issue on linear algebra in control, signals and imaging.

[16] R.F. Love, J.G. Morris, and G.O Wesolowsky. *Facilities Location: Models and Methods*. North-Holland, 1988.

[17] H.M. Markowitz. *Portfolio Selection: Efficient Diversification of Investments*. Wiley, New York, 1959.

[18] Z.A. Melzak. On the problem of steiner. *Canadian Mathematical Bulletin*, 16:143–148, 1961.

[19] A.G.M. Michell. The limits of economy of material in frame structures. *Phil. Mag.*, 8:589–597, 1904.

[20] A.V. Oppenheim and R.W. Schafer. *Digital Signal Processing*. Prentice Hall, Englewood Cliffs, NJ, 1970.

[21] M.L. Overton. A quadratically convergent method for minimizing a sum of Euclidean norms. *Mathematical Programming*, 27:34–63, 1983.

[22] W.D. Smith. How to find steiner minimal trees in euclidean d-space. *Algorithmica*, 7:137–177, 1992.

[23] Jos F. Sturm. Using sedumi 1.02, a matlab* toolbox for optimization over symmetric cones. Communications Research Laboratory, McMaster University, Hamilton, Canada, August 1998.

[24] R.J. Vanderbei. Large-scale nonlinear AMPL models. http://www.sor.princeton.edu/˜rvdb/ampl/nlmodels/.

[25] R.J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, 1997.

[26] R.J. Vanderbei and D.F. Shanno. An interior-point algorithm for nonconvex nonlinear programming. Technical Report SOR-97-21, Statistics and Operations Research, Princeton University, 1997. To appear in *Computational Optimization and Applications*.

[27] E. Weiszfeld. Sur le point pour lequel la somme des distances de n points donné est minimum. *Tôhoku Mathematics Journal*, 43:355–386, 1937.

[28] S.-P. Wu, S. Boyd, and L. Vandenberghe. Magnitude filter design via spectral factorization and convex optimization. *Applied and Computational Control, Signals and Circuits*, 1997. To appear.

[29] G. Xue and Y. Ye. Efficient algorithms for minimizing a sum of euclidean norms with applications. *SIAM J. Optimization*, 7:1017–1036, 1997.
[30] Yin Zhang. Lipsol: Linear-programming interior-point solvers. Dept. of Mathematics and Statistics, Univ. of Maryland Baltimore County, Baltimore, Maryland, 1995.

APPENDIX

MATLAB*'s set implementation using matrices*

```
ARCS=zeros(n+1,m+1,n+1,m+1);
nv=0;
for i=1:n+1, for j=1:m+1, for k=1:n+1, for l=1:m+1,
    if ((abs(k-i)<=3) & (abs(l-j)<=3) &
        (abs(gcd(k-i,l-j))==1) & ((i>k | (i==k & j>l)))
       nv=nv+1;
       ARCS(i,j,k,l)=nv;
    end;
end; end; end; end;
```

Abstract set definition in AMPL

```
set ARCS := { (xi,yi) in NODES, (xj,yj) in NODES:
    abs( xj-xi ) <= 3 && abs(yj-yi) <=3 && abs(gcd[ xj-xi, yj-yi ]) == 1
    && ( xi > xj || (xi == xj && yi > yj) )
    };
```

FIGURE 1. Set definition in the structural optimization model in AMPL and MATLAB

MATLAB*'s parameter definition*

```
length=zeros(nv,1);
for i=1:n+1, for j=1:m+1, for k=1:n+1, for l=1:m+1,
    if ARCS(i,j,k,l)>0
       length(ARCS(i,j,k,l))=sqrt( (k-i)^2 + (l-j)^2 );
    end;
end; end; end; end;
```

AMPL*'s parameter definition*

```
param length {(xi,yi,xj,yj) in ARCS} := sqrt( (xj-xi)^2 + (yj-yi)^2 );
```

FIGURE 2. Parameter definition over a set in the structural optimization model in AMPL and MATLAB

Assembling the coefficient matrices in MATLAB

```
c=[];
for k=1:n+1, for l=1:m+1,
    c1=zeros(1,3*nv+(m+1)*(n+1)); c2=zeros(1,3*nv+(m+1)*(n+1));
    for i=1:n+1, for j=1:m+1,
        if ARCS(i,j,k,l)>0
          c1(2*nv+ARCS(i,j,k,l))=(i-k)/(length(ARCS(i,j,k,l))^1.5);
          c2(2*nv+ARCS(i,j,k,l))=(j-l)/(length(ARCS(i,j,k,l))^1.5);
        end;
        if ARCS(k,l,i,j)>0
          c1(2*nv+ARCS(k,l,i,j))=(i-k)/(length(ARCS(k,l,i,j))^1.5);
          c2(2*nv+ARCS(k,l,i,j))=(j-l)/(length(ARCS(k,l,i,j))^1.5);
        end;
    end; end;
    c=[c;-c1;-c2];
    c1(3*nv+(l-1)*(n+1)+k)=1; c2(3*nv+(l-1)*(n+1)+k)=1;
    c=[c;c1;c2];
    d=[d;[f1(k,l);f2(k,l);-f1(k,l);-f2(k,l)]];
end; end;
C=[C;c];
```

Defining the same constraints in AMPL

```
subject to load_constraints {(xk,yk) in NODES, d in 1..2}:
    sum {(xi,yi,xk,yk) in ARCS} y[xi,yi,xk,yk]*
        (if d==1 then xi-xk else yi-yk)/length[xi,yi,xk,yk]^1.5
  + sum {(xk,yk,xj,yj) in ARCS} y[xk,yk,xj,yj]*
        (if d==1 then xj-xk else yj-yk)/length[xk,yk,xj,yj]^1.5
  = f[xk,yk,d];
```

FIGURE 3. Defining constraints in the structural optimization model in AMPL and MATLAB

ROBERT J. VANDERBEI, PRINCETON UNIVERSITY, PRINCETON, NJ

HANDE YURTTAN, PRINCETON UNIVERSITY, PRINCETON, NJ