EFFICIENT CONVEX OPTIMIZATION FOR ENGINEERING DESIGN

Stephen Boyd[†], Lieven Vandenberghe[‡], Michael Grant[†]

† *Information Systems Laboratory, Stanford University, Stanford, California*

‡ *Electrical Engineering Department, K.U. Leuven, Leuven, Belgium*

Abstract. Many problems in engineering analysis and design can be cast as convex optimization problems, often nonlinear and nondifferentiable. We give a high-level description of recently developed interior-point methods for convex optimization, explain how problem structure can be exploited in these algorithms, and illustrate the general scheme with numerical experiments. To give a rough idea of the efficiencies obtained, we are able to solve convex optimization problems with over 1000 variables and 10000 constraints in around 10 minutes on a workstation.

Keywords. Optimization, numerical methods, linear programming, optimal control, robust control, convex programming, interior-point methods, FIR filter design, conjugate gradients

1. INTRODUCTION

Many problems in engineering analysis and design can be cast as convex optimization problems, *i.e.*,

$$\begin{array}{ll} \min & f_0(x) \\ \text{s.t.} & f_i(x) \leq 0, \quad i = 1, \ldots, L, \end{array}$$

where the functions f_i are convex. It is widely known that such problems have desirable properties, *e.g.*, locally optimal solutions are globally optimal. In fact much more is true — roughly speaking, convex optimization problems are *fundamentally tractable*, both in theory and in practice.

In this paper we consider the convex optimization problem

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & F(x) \geq 0, \end{array} \tag{1}$$

where

$$F(x) \stackrel{\Delta}{=} F_0 + x_1 F_1 + \cdots + x_m F_m.$$

The problem data are the vector $c \in \mathbf{R}^m$ and $m+1$ symmetric matrices $F_0, \ldots, F_m \in \mathbf{R}^{n \times n}$. The inequality sign in $F(x) \geq 0$ means that $F(x)$ is positive semidefinite. We call problem (1) a *positive definite program* or PDP.

Although the PDP (1) may appear quite specialized, it includes many important problems as special cases. For instance, consider the linear program (LP)

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax + b \geq 0 \end{array} \tag{2}$$

where the inequality denotes *componentwise* inequality. Since a vector $v \geq 0$ (componentwise) if and only if the matrix $\mathbf{diag}(v)$ (the diagonal matrix with the components of v on its diagonal) is positive semidefinite, we can express the linear program (2) as a PDP with $F(x) = \mathbf{diag}(Ax + b)$, *i.e.*,

$$F_0 = \mathbf{diag}(b), \quad F_i = \mathbf{diag}(a_i), \quad i = 1, \ldots, m,$$

where a_i is the ith column of A.

Many other problems, including most of the convex optimization problems encountered in engineering, can be written as PDPs. Positive definite constraints also arise directly in a number of important applications, for example, in control and system theory (Boyd *et al.* 1994, Boyd and Barratt 1991). Positive definite programming therefore offers a unified way to study the properties of, and derive algorithms for, a wide variety of convex optimization problems. Most importantly, however, PDPs can be solved very efficiently using recently developed interior-point methods.

Interior-point methods were first introduced for linear programming by Karmarkar in 1984 (Karmarkar 1984). Although controversial at the time, it is now generally accepted that interior-point methods for LPs are competitive with the simplex method, and even faster for large problems. An important breakthrough took place in 1988, when Nesterov and Nemirovsky generalized interior-point methods for LP to general convex programming (Nesterov and Nemirovsky 1994).

The experimental results reported in this paper were obtained using a *primal-dual potential reduction method.* The method is based on the method described by Nesterov and Nemirovsky (Nesterov and Nemirovsky 1994; §4.5), which generalizes Ye's method for linear programming (Ye 1991). We modify this basic interior-point method by using conjugate gradients to exploit problem structure, as explained in (Vandenberghe and Boyd 1993).

We give a brief description of the method in the next two sections. For more details, see (Vandenberghe and Boyd 1994). In §4 we show how to exploit problem structure, and in §5 we apply these techniques to three (families of) engineering problems.

2. DUALITY

The dual problem associated with the PDP (1) is

$$\begin{array}{ll} \max & -\mathbf{Tr} F_0 Z \\ \text{s.t.} & \mathbf{Tr} F_i Z = c_i,\ i = 1, \ldots, m \\ & Z \geq 0. \end{array} \tag{3}$$

Here the variable is the matrix $Z = Z^T \in \mathbf{R}^{n \times n}$, and $\mathbf{Tr} X$ denotes the trace of the matrix X.

Under mild conditions, the optimal values of the primal problem (1) and the dual problem (3) are equal (Rockafellar 1970). This fact has important consequences.

- Every dual feasible Z (*i.e.*, every matrix Z that satisfies the constraints in (3)), proves a lower bound $-\mathbf{Tr} F_0 Z$ on the optimal value of the PDP (1).
- If $F(x) \geq 0$, and Z is dual feasible, then the difference between the primal and the dual objective values is nonnegative,

  $$c^T x + \mathbf{Tr} F_0 Z \geq 0.$$

  We call the quantity on the left the *duality gap* of the pair (x, Z).
- If the duality gap is zero, then x and Z are optimal points, solving the primal PDP (1) and the dual (3).

Primal-dual interior-point methods generate a sequence of primal and dual feasible points $x^{(k)}$ and $Z^{(k)}$, where $k = 0, 1, \ldots$ denotes iteration number. We can interpret $x^{(k)}$ as a *suboptimal point* and $Z^{(k)}$ as a *certificate* that proves the lower bound $-\mathbf{Tr} F_0 Z^{(k)}$ on the optimal value. The iteration is terminated if the duality gap $c^T x^{(k)} + \mathbf{Tr} F_0 Z^{(k)}$ becomes less than a pre-specified tolerance ϵ.

3. ALGORITHM

The interior-point method is based on a potential function $\varphi(x, Z)$. The essential properties of the potential function are:

- φ is smooth on the interior of the feasible set, and infinite outside the feasible set.
- The duality gap is less than $\alpha \exp \varphi(x, Z)$, where α is a positive constant; in particular, if $\varphi(x, Z) \to -\infty$, then (x, Z) approach optimality.

At each iteration of the algorithm, the potential function decreases by at least a fixed amount:

$$\varphi(x^{(k+1)}, Z^{(k+1)}) \leq \varphi(x^{(k+1)}, Z^{(k+1)}) - \beta,$$

where β is an absolute constant. As a consequence, the iterates remain feasible, and converge to the optimum. The duality gap converges to zero exponentially.

For the PDP (1), we use the potential function

$$\begin{aligned} \varphi(x, Z) &\triangleq q \log(c^T x + \mathbf{Tr} F_0 Z) \\ &\quad + \log\det F(x)^{-1} + \log\det Z^{-1}. \end{aligned} \tag{4}$$

The first term (4) rewards a decrease in the duality gap. The second and third terms act as barrier functions that keep $F(x)$ and Z positive definite. The scalar q is a parameter that controls the relative weight of the different terms.

The updates $x^{(k+1)}$, $Z^{(k+1)}$ are generated from $x^{(k)}$, $Z^{(k)}$ as follows. A suitable pair of search directions, δx, δZ, are found by (approximately) solving a least-squares problem. Then, primal and dual step lengths α and β are chosen to (approximately) minimize the potential $\varphi(x^{(k)} + \alpha \delta x, Z^{(k)} + \beta \delta Z)$ in the plane defined by the current points and the search directions. This is called the *plane search.* We then set $x^{(k+1)} = x^{(k)} + \alpha \delta x$, $Z^{(k+1)} = Z^{(k)} + \beta \delta Z$, and repeat the process.

4. CONJUGATE GRADIENTS

One of the most remarkable properties of interior-point methods is their insensitivity to problem size. The number of iterations increases very slowly, typically as the logarithm of the problem size. For most practical purposes, the number of steps can be considered to be almost independent of the dimension. Typical numbers range from 10 to 50.

The overall computational effort is therefore determined by the amount of work per iteration. Skipping details, we can say that the main effort in every iteration is the solution of a least-squares problem of the form

$$\min_{v \in \mathbf{R}^m} \left\| D^{(k)} - \sum_{i=1}^{m} v_i S^{(k)} F_i S^{(k)} \right\|_F \qquad (5)$$

to compute suitable search directions. Here, $\|\cdot\|_F$ denotes the Frobenius norm, *i.e.*,

$$\|A\|_F^2 = \mathbf{Tr}\left(A^T A\right) = \sum_{i,j} A_{ij}^2.$$

As the superscripts suggest, the matrices $D^{(k)} = D^{(k)T}$ and $S^{(k)} = S^{(k)T}$ in (5) change every iteration. As a practical guideline, therefore, the total complexity of solving a convex problem is equal to the work of solving a relatively small (say, 10–50) and almost constant number of least-squares problems of the form (5).

Problem (5) has m variables and $n(n+1)/2$ equations. Using direct methods it can be solved in $\mathrm{O}(m^2n^2)$ operations. Important savings are possible when the matrices F_i are structured. The easiest type of structure to exploit is block-diagonal structure. Assume $F(x)$ consists of L diagonal blocks of size n_i, $i = 1, \ldots, L$. Then the number of equations in (5) is $\sum_{i=1}^{L} n_i(n_i+1)/2$, which is often an order less than $n(n+1)/2$. For instance, in the LP case (diagonal matrix $F(x)$), the number of variables is n, and solving the least-squares problem requires only $\mathrm{O}(m^2n)$ operations.

Usually much more can be gained by exploiting the internal structure of the diagonal blocks in F_i. The conjugate gradients method or the LSQR algorithm of Paige and Saunders (Paige and Saunders 1982) appear to be very well suited. In exact arithmetic, these algorithms solve (5) in $m+1$ iterations, where each iteration requires one evaluation of the 'forward' mapping,

$$(v_1, \ldots, v_m) \mapsto \sum_{i=1}^{m} v_i F_i, \qquad (6)$$

and one evaluation of its adjoint

$$W \mapsto (\mathbf{Tr} F_1 W, \ldots, \mathbf{Tr} F_m W) \qquad (7)$$

for some vector v and symmetric matrix $W = W^T$. When the matrices F_i are unstructured, these two operations take mn^2 operations. Hence, the cost of solving (5) using LSQR is $\mathrm{O}(n^2m^2)$, and nothing is gained over direct methods.

In most cases, however, the two operations (6) and (7) are much cheaper than mn^2 because of the special structure of the matrices F_i. A well-known example is a sparse LP, but sparsity is not the only example. The equations are often dense, but still highly structured in the sense that the mappings (6) and (7) can be evaluated faster than $\mathrm{O}(mn^2)$ operations. We will see several examples in §5.

In practice, *i.e.*, with roundoff error, the standard conjugate gradients algorithms can perform quite poorly; the number of iterations required to solve the least-squares problem can be much higher than $m+1$. Two techniques can be used to improve the rate of convergence. The standard technique is pre-conditioning, described in (Golub and Loan 1989). There is no simple universal method for constructing a suitable pre-conditioner, but often a good choice follows from the properties of the underlying engineering problem. The design of a pre-conditioner is problem dependent; the engineer's experience with and intuition about the problem is invaluable.

The other technique is re-orthogonalization. Re-orthogonalizaton can always be applied. It makes the algorithm converge as in exact arithmetic (*i.e.*, in $\mathrm{O}(m)$ steps), but is expensive: it increases the cost of N LSQR iterations with $\mathrm{O}(mN^2)$ operations. This is too expensive if the algorithm is run to completion.

This brings us to another great advantage of using iterative methods to solve the least-squares problem (5): the option of early termination. The LSQR-algorithm produces good search directions even if the iteration is stopped before the exact solution of the least-squares problem (5) has been found. It can be proved that early termination does not affect the worst-case convergence rate, provided a good stopping criterion is used (Vandenberghe and Boyd 1993). In our experience, we often find that suitable search directions can be generated after only $\mathrm{O}(\sqrt{m})$ LSQR iterations. An important consequence is that the cost of re-orthogonalization is not nearly as high as it is when LSQR is run to completion, *i.e.*, $\mathrm{O}(m)$ iterations.

We should mention one important issue that arises when early termination of LSQR is used. When the least-squares problem (5) is solved exactly, the dual search direction δZ is computed from its residual. With early termination of LSQR, this search direction will not exactly satisfy the equality constraints $\mathbf{Tr} F_i \delta Z = 0$ required for dual feasibility. The user must supply a subroutine that perturbs a dual search direction that nearly satisfies these equalities into one that exactly satisfies them. The design of this subroutine is problem-dependent; see (Vandenberghe and Boyd 1993).

Let us summarize the main points. Many convex optimization problems arising in engineering can be cast in the form of the PDP (1). Roughly speaking, solving the PDP (1) requires the (approximate) solution of between 10 and 50 least-squares problems of the form (5). Using LSQR to approximately solving such a least-squares problem requires somewhere between $O(\sqrt{m})$ and $O(m)$ evaluations of the forward and adjoint mappings (6) and (7). It follows that if we can exploit problem structure to evaluate these mappings efficiently, we can solve the PDP (1) efficiently.

This general scheme is summarized in Figure 1.

can evaluate forward, adjoint fast *(exploiting structure)*

$\Downarrow$

can solve least-squares problem fast *(using LSQR)*

$\Downarrow$

can solve convex problem fast *(using interior-point methods)*

Fig. 1. Exploiting problem structure to efficiently solve convex problems.

5. EXPERIMENTAL RESULTS

In this section we illustrate the general scheme outlined above on three (convex) engineering problems. The first problem is FIR filter design. The second is a robust input design problem: we must determine an input that will work well with multiple, given plants. The third problem is robust input design for a multiple-input multiple-output plant. In each case we generate a family of problems indexed by a dimension M which is the number of free variables in the problem. Michael Grant developed the software, called INTPT below, as well as the examples.

Even though INTPT handles general PDPs, these three examples are in fact LPs. This allows us to compare the computational effort and memory usage of INTPT with that of LSSOL, a widely used package for solving linear and linearly-constrained quadratic programs (Gill *et al.* 1986). We must immediately point out that LSSOL was not designed to exploit structure; we are merely comparing INTPT, an interior-point code that exploits problem structure, to LSSOL, an optimized simplex-based LP code that does not exploit structure.

We should also point out that INTPT is still in development, and that no effort was expended in optimizing it for these examples. For instance, we used no pre-conditioning at all.

5.1 Testing environment

We used a DECstation 5000/240 with 64 megabytes of memory. DEC Fortran version 3.2 was used to compile LSSOL and the Fortran subroutines employed by INTPT; the GNU Foundation's `g++` was used to compile the C++ code. Full compiler optimization was enabled in all cases. We used the standard UNIX library routine `getrusage` to measure the total execution time, $t(M)$, and the amount of resident (in-core) memory usage, $m(M)$, for a given number of variables M. We collected data for a wide range of problem sizes, in an attempt to measure the general trends of each.

The execution time $t(M)$ was fit to a function of the form $\tilde{t}(M) = a + bM^c$ by minimizing

$$\sum_i (\log(a + bM_i^c) - \log t(M_i))^2 \qquad (8)$$

over a, b, and c. Likewise, the memory consumption was fit to $\tilde{m}(M) = a + bM + cM^2$ by minimizing

$$\sum_i (a + bM_i + cM_i^2 - m(M_i))^2 \qquad (9)$$

over a, b, and c. These curves are included in the plots presented below. The constant factor in $\tilde{m}(M)$ was subtracted from the data before plotting, in order to remove the contributions of the programs themselves and to better show the actual growth rates on a logarithmic plot.

5.2 FIR filter design

Suppose we wish to design a (non-causal) zero-phase filter which satisfies a set of frequency response constraints while minimizing its peak impulse response. We focus on the Type I low-pass case, a symmetric filter with $2L + 1$ taps, and frequence response

$$H(f) = \sum_{k=-L}^{L} h(k) e^{-j2\pi k T_s f}$$

$$= h(0) + \sum_{k=1}^{L} 2h(k)\cos(2\pi k T_s f)$$

Other filter design problems follow similarly.

Given a desired frequency response $H_r(f)$ and permissible deviation $\Delta(f)$, we might specify the design problem as

$$\begin{array}{ll} \min & \max |h(k)| \\ \text{s.t.} & |H(f_i) - H_r(f_i)| \le \Delta(f_i) \\ & i = 1, \ldots, N, \; k = 0, \ldots, L, \end{array} \qquad (10)$$

where the f_is denote frequencies of interest within the passband and stopband of the filter. The unknowns are the coefficients $h(0), \ldots, h(L)$.

We can write (10) as an LP (2) by introducing a new variable w:

$$\begin{array}{ll} \min & w \\ \text{s.t.} & -w \le h(k) \le w, \; k = 0, \ldots, L \\ & \Delta(f_i) \le H(f_i) - H_r(f_i) \le \Delta(f_i) \\ & i = 1, 2, \ldots, N. \end{array}$$

This LP is very structured. In order to evaluate the forward mapping associated with the constraints, we need to compute the frequency response of the filter, given a set of coefficients $h(k)$. This can be done very efficiently using the Fast Fourier Transform (FFT). The adjoint mapping can be computed very efficiently using the inverse FFT.

To generate example problems of various sizes, we began with the following specification of a *continuous-time*, zero-phase FIR low-pass filter:

$$\begin{array}{lll} \min & \|h\| & \\ \text{s.t.} & h(t) = h(-t) & \\ & h(t) = 0 & t > 1\text{sec} \\ & |H(f) - 1| < 0.01 & f \le 4\text{Hz} \\ & |H(f)| < 0.01 & f \ge 8\text{Hz} \end{array} \qquad (11)$$

where $\|h\|$ denotes the $\mathbf{l}^\infty$ or peak norm of the impulse response h. The impulse response of the optimal filter is shown in Figure 2. Discrete approximations to this problem were created by assuming a piecewise-linear impulse response with $2M$ evenly-spaced segments. This is equivalent to a $2M-1$-tap FIR filter (M unique taps) with a triangle hold at the output. The frequency-response constraints were discretized at $4M+1$ frequencies and compensated to include the contribution of the triangle-hold response. The result was a linear program with $M+1$ variables and $10M+2$ constraints.

INTPT exploited the structure of this problem by using a mixed-radix, fast discrete cosine transform (DCT) to evaluate the frequency-response

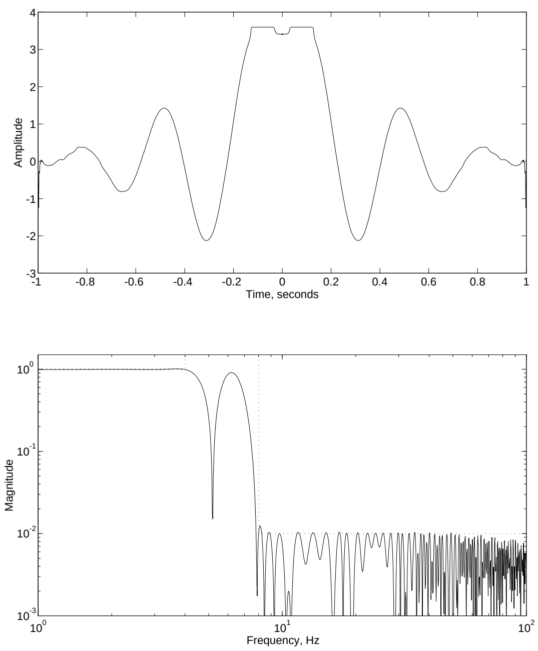

Fig. 2. The impulse response and frequency response of the optimal filter.

constraints. We chose problem sizes for which the DCT was most efficient, so the evaluation time of the forward and adjoint operators was reduced by a factor of $O(M/\log M)$. To obtain $4M+1$ frequencies from the DCT we padded the M unique taps with $3M+1$ zeros; yet despite this apparent waste, the logarithmic growth of the DCT's complexity still yielded big gains.

The plots in Figure 3 compare the execution times and data size, respectively, of the two solvers for various problem sizes. In this case, the advantage in both execution time and memory consumption lies solely with INTPT. The execution time grew as $O(M^{1.67})$ for INTPT and $O(M^{3.28})$ for LSSOL.

The largest problem solved had over 1000 (independent) variables (over 2000 taps in the FIR filter) and 10000 constraints. INTPT solved this problem in about 4 minutes, using about 4Mb of memory.

5.3 Robust open-loop input design

In the *open-loop input design problem*, we are given a discrete-time linear system with input $u(k) \in \mathbf{R}^{n_i}$ and output $y(k) \in \mathbf{R}^{n_o}$. The problem is to design an input trajectory that makes the output track a given reference signal $y^{\text{des}}(k)$. For example, one may want to minimize the *peak tracking error*

$$\max_k \left\| y(k) - y^{\text{des}}(k) \right\|_\infty .$$

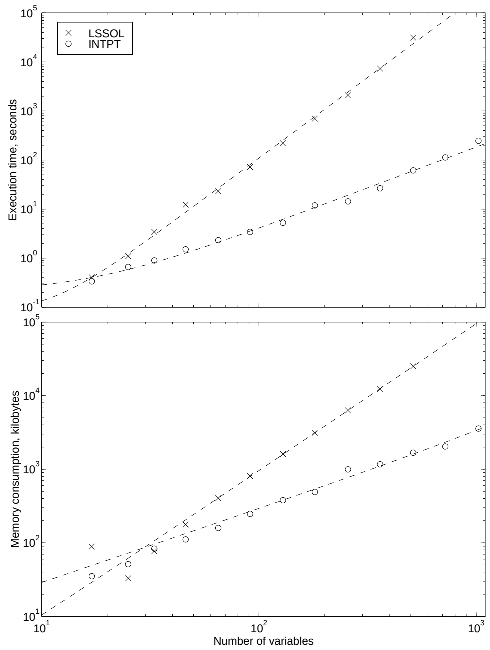

Fig. 3. Execution times and memory consumption for the filter design example.

The *robust* input design problem involves multiple plants

$$
\begin{aligned}
x^{(i)}(k+1) &= \Phi^{(i)} x^{(i)}(k) + \Gamma^{(i)} u(k) \\
y^{(i)}(k) &= H^{(i)} x^{(i)}(k) + J^{(i)} u(k),
\end{aligned}
\tag{12}
$$

for $i = 1, \ldots, L$. The same input is applied to each plant. The purpose is to design an input trajectory that works well for all plants simultaneously, *e.g.*, by minimizing the maximum of the peak tracking errors

$$
\max_{i,k} \left\| y^{(i)}(k) - y^{\mathrm{des}}(k) \right\|_\infty .
$$

We can also take into account various convex constraints on inputs and outputs, *e.g.*, limits on the input amplitude,

$$
\|u(k)\|_\infty \le u_{\max},
$$

slew rate constraints on the input,

$$
\|u(k+1) - u(k)\|_\infty \le s_{\max},
$$

or envelope bounds on the output,

$$
y_{\min} \le y_j(k) \le y_{\max}.
$$

A convex optimization problem that includes some of these constraints is

$$
\begin{aligned}
\min \quad & \max_{i,k} \left\| y^{(i)}(k) - y^{\mathrm{des}}(k) \right\|_\infty \\
\text{s.t.} \quad & 0 \le u_j(k) \le u_{\max} \\
& |u_j(k+1) - u_j(k)| \le s_{\max} \\
& \text{the state equations (12)} \\
& x^{(i)}(0) = 0 \\
& 0 \le k < N,\ 1 \le i \le L,\ j = 1, \ldots, n_i.
\end{aligned}
\tag{13}
$$

The variables are the input vectors $u(k)$, $k = 0, \ldots, N-1$.

Again, problem (13) can be written as an LP with very structured equations. To evaluate the forward mapping associated with the constraints, we need to compute the output vectors $y^{(i)}(k)$ for a given input trajectory $u(k)$, $k = 0, \ldots, N-1$. This can be done very efficiently by directly simulating the linear systems (12). The adjoint mapping can be evaluated by simulating the *adjoint linear systems* (Kailath 1980). This reduces the calculation time for the linear operators from $O(N^2)$, if no structure is exploited, to $O(N)$.

In the experiment we take two single-input single-output systems, obtained from the continuous-time systems

$$
\begin{aligned}
H_1(s) &= 16/(s^2 + 1.2s + 16) \\
H_2(s) &= 25/(s^2 + 1.2s + 25)
\end{aligned}
$$

using a first-order hold on the input, with sampling interval $\Delta T = 5/M$, over the time interval $0 \le t \le 5$. The input is constrained to lie between zero and one. The maximum slew rate is 1.25/sec. The reference trajectory is piecewise linear: zero for $0 \le t \le 2$, one for $3 \le t \le 5$, with a linear transition (slew) between $t = 2$ and $t = 3$. This results in an LP with $M+2$ variables and $10M+12$ constraints.

Figure 4 shows the step responses of the two systems, an optimal input u, and the resulting output trajectories. The optimal tracking error is about 0.05, which is plotted in dotted line type above and below the reference trajectory. The figures demonstrate several interesting properties of the problem. First of all, the impulse responses of the two systems often have opposite sign; consequently the optimal control inputs for the individual plants are quite different. Secondly, the optimal control input contains the very interesting "preparatory" behavior for $t \le 1$ that standard control input design strategies would not suggest.

The plot in Figure 5 compares the execution time and memory consumption of the two methods for various problem sizes. While LSSOL is faster for smaller problems, the execution time grows faster

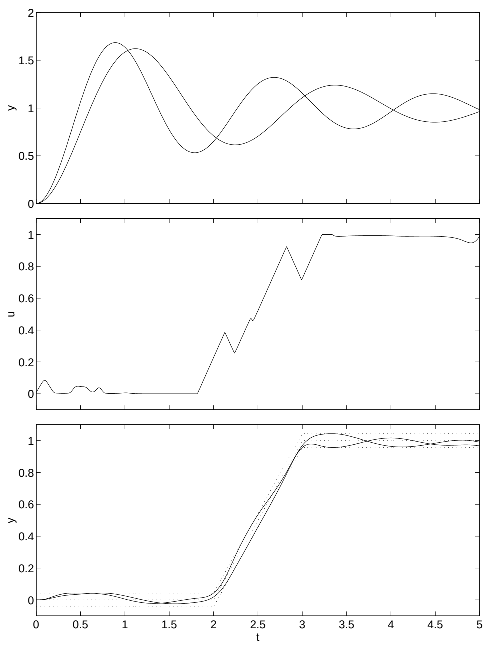

Fig. 4. Step responses, control input, and output trajectories, respectively, for the robust input design example.

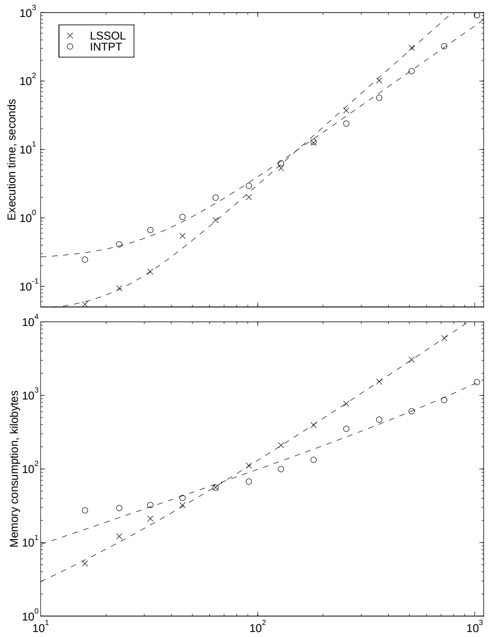

Fig. 5. Execution times and memory consumption for the robust input design example.

with size than for INTPT, and INTPT overtakes it at approximately $M = 170$. Specifically, the execution time for INTPT grew as approximately $O(M^{2.2})$, while that for LSSOL grew as $O(M^{2.8})$.

Comparing the memory consumption of the two programs exposes another significant difference. LSSOL consumes a much larger amount of memory than INTPT, and those requirements grow with $O(M^2)$ behavior. As a result, the largest problems exceeded our system-imposed memory usage limits; but without those limits, virtual memory paging would begin to degrade the performance of either solver. The iterative methods of INTPT, however, allow for much more modest growth in memory needs with problem size, so much larger problems can be handled. The growth does contain a small $O(M^2)$ component, but its coefficient is much smaller than for LSSOL, resulting in reasonable memory demands even for the largest problem sizes.

The largest problem considered had over 1000 variables and 10000 constraints. INTPT solved it in about 15 minutes, using negligible memory (under 2Mb).

5.4 Multi-input, multi-output input design

The third problum is a robust input design problem for a multi-input muti-output system. We consider a (simplified) model of a rapid thermal processor system, used for semiconductor manufacturing. An array of n_i tungsten-halogen lamps

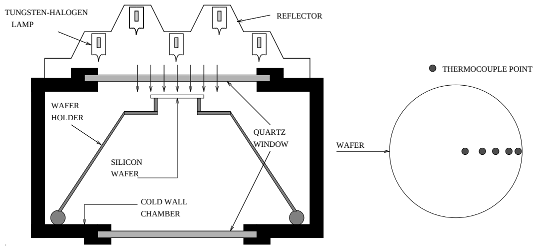

Fig. 6. An experimental rapid thermal processing (RTP) configuration.

is used to rapidly control the temperature of a semiconductor wafer, which for testing purposes is outfitted with n_o thermocouples, as shown in Figure 6 (Norman 1992, Gyugyi 1993).

A linearized model of the dynamics between the lamp powers and the thermocouple readings was taken from (Gyugyi 1993), in which 3 lamps and 5 thermocouples are employed. The dynamics from the lamp outputs $l(t)$ to the thermocouple readings $o(t)$ were described by a 5-state state-space model $(\Phi, \Gamma, H = I, J = 0)$ with eigenvalues ranging from -1.113 to -0.0181. A first-order model $H_{\mathrm{lamp}}(s) = 8/(s+8)$ was assigned to the dynamics between each lamp input $u_i(t)$ and its output $l_i(t)$. We also placed hard limits on the lamp inputs, and slew-rate limits on the lamp outputs.

A typical reference trajectory for rapid thermal processing involves ramping the temperature up to a desired value, holding, and then ramping back

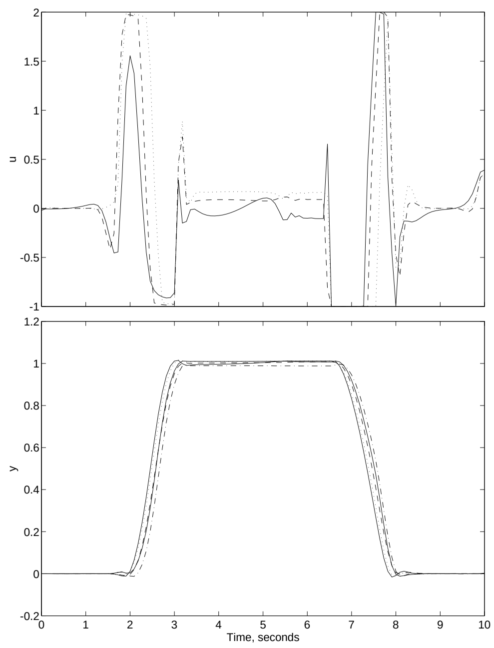

Fig. 7. Control inputs and output trajectories of the MIMO control design example.

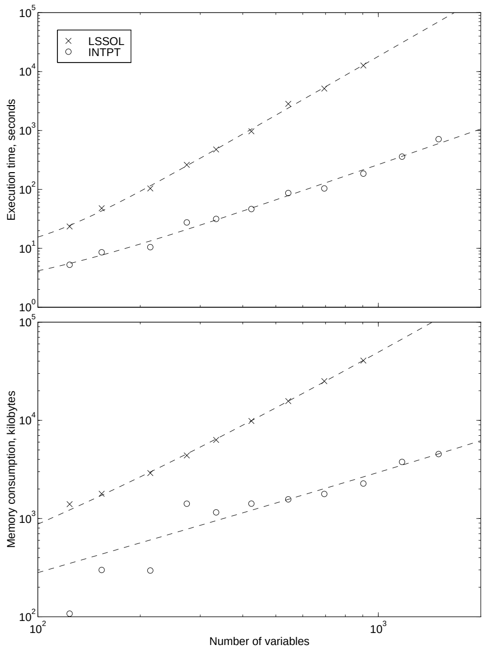

Fig. 8. Execution times and memory consumption for the MIMO input design example.

down:

$$r(t) = \begin{cases} 0 & 0 \le t \le 2 \\ 0.8(t-2) & 2 \le t \le 3.25 \\ 1 & 3.25 \le t \le 6.75 \\ 0.8(8-t) & 6.75 \le t \le 8 \\ 0 & 8 \le t \le 10 \end{cases} \quad (14)$$

If we measure tracking error only during the time periods in which the reference trajectory is constant, and ignore the behavior during the transitions, the design problem becomes

$$\begin{array}{ll} \min & w \\ \text{s.t.} & \dot{o}(t) = \Phi o(t) + \Gamma l(t) \\ & \dot{l}(t) = 8(u(t) - l(t)) \\ & \|o(t)\|_\infty \le w \quad\quad 0 \le t \le 2 \\ & \|o(t) - \vec{1}\|_\infty \le w \quad 3.25 \le t \le 6.75 \\ & \|o(t)\|_\infty \le w \quad\quad 8 \le t \le 10 \\ & -1 \le u_i(t) \le 2 \\ & \|\dot{l}_i(t)\|_\infty \le 12 \\ & 0 \le t \le 10 \end{array} \quad (15)$$

Plots of optimal control inputs and the resulting trajectories are given in Figure 7.

We form an LP from this infinite-dimensional problem by assuming a zero-order hold on the inputs u, discretizing the dynamics, and sampling the constraints. Given a sampling rate $\Delta t = \bar{t}/M$, the result is an LP with $M + 2$ variables and approximately $13.5(M + 1)$-constraints.

Once again, using state and co-state simulation for the dynamic system, INTPT significantly reduced the complexity of the forward and adjoint operators used to solve the problem. The plots in Figure 8 document the significant performance gains made over LSSOL. The execution time grew as $O(M^{2.01})$ for INTPT and $O(M^{3.32})$ for LSSOL.

The largest problem considered had over 1500 variables and 20000 constraints. INTPT solved it in about 18 minutes, using about 5Mb memory.

5.5 Future improvements

The results shown here are the first complete tests of INTPT that have been performed. Profiling the code reveals that there is significant room for problem-independent improvement in the performance of the engine which could not be incorporated into the code before the completion of this article. Therefore, we can expect that future versions of INTPT will show much improved performance.

Execution profiling has revealed that a very large component of the execution time is spent performing the reorthogonalizations in the iterative least-squares solver. Currently, LSQR (Paige and Saunders 1982) with early termination is used to solve the least-squares problems associated with the interior-point technique, and complete reorthogonalization is used at each iteration to insure convergence. Methods of selective reorthogonalization, such as that described in (Parlett and Scott 1979), promise to greatly reduce the number of orthogonalizations performed.

For convex programs consisting primarily of linear, quadratic, and other simple nonlinear constraints, the second largest consumer of CPU time is the plane search algorithm. The plane search reduces to minimizing over α and β the function

$$\begin{aligned} f(\alpha,\beta) \;=\; & q\log(c_1+c_2\alpha+c_3\beta) \\ & -\sum_{i=1}^{N}\log(1+\mu_i\alpha) \\ & -\sum_{i=1}^{N}\log(1+\nu_i\beta) \end{aligned} \qquad (16)$$

where $c_1 > 0$, and the constants c_i, μ_i and ν_i are given (Vandenberghe and Boyd 1993). The current implementation uses damped Newton line-search iterations. We are currently experimenting with rational approximations of f, which appear to greatly reduce the number of necessary plane search steps.

Finally, we note that the architecture of the C++-based INTPT package does not preclude the use of FORTRAN to implement portions of numerically-intensive code. In fact, the FORTRAN Basic Linear Algebra Subroutines (BLAS) have been used extensively throughout, and the DCT algorithm used above to speed up the filter design examples is also coded in FORTRAN. The optimizing power of FORTRAN compilers (and FORTRAN programmers) is still generally superior to that of C++, so implementing the constraint calculations in FORTRAN should insure maximum performance; and since those calculations necessarily comprise a large portion of the overall execution time, healthy increases in speed can result.

6. CONCLUSIONS

The sizes of the test problems vary between a few hundred and a few thousand variables. In the (sparse) linear programming literature these problems would be considered small to medium sized. We do not claim that interior-point methods are intrinsically faster than simplex for problems in this size range. We feel that interior-point methods do have two strong advantages:

- The problems arising in engineering are often dense but highly structured. Interior-point methods offer a straightforward way to exploit this structure.
- Although the examples presented in this paper are LPs, many problems arising in engineering are nonlinear; they can be cast as PDPs but not LPs. Interior-point methods readily handle such problems.

In optimization-based engineering (and indeed, in dense linear programming), the problems we consider here are considered large scale. The methods described in this paper open the possibility of routinely solving large scale convex engineering problems on a workstation.

ACKNOWLEDGMENTS

We thank Gene Golub and Michael Saunders for invaluable advice on iterative least-squares methods, including pointing us to the LSQR algorithm.

The research of S. Boyd was supported in part by AFOSR (under F49620-92-J-0013), NSF (under ECS-9222391), and ARPA (under F49620-93-1-0085). L. Vandenberghe is Postdoctoral Researcher of the Belgian National Fund for Scientific Research (NFWO). His research was supported in part by the Belgian program on Interuniversity Attraction Poles (IUAP 17 and 50) initiated by the Belgian State, Prime Minister's Office, Science Policy Programming. Michael Grant was supported by an AASERT grant.

REFERENCES

Boyd, S. and C. Barratt (1991). *Linear Controller Design: Limits of Performance.* Prentice-Hall.

Boyd, S., L. El Ghaoui, E. Feron and V. Balakrishnan (1994). *Linear Matrix Inequalities in System and Control Theory.* Vol. 15 of *Studies in Applied Mathematics.* SIAM. Philadelphia, PA.

Gill, P. E., S. J. Hammarling, W. Murray, M. A. Saunders and M. H. Wright (1986). User's guide for LSSOL (Version 1.0): A FORTRAN package for constrained least-squares and convex quadratic programming. Technical Report SOL 86-1. Operations Research Dept., Stanford University. Stanford, CA 94305.

Golub, G. and C. V. Loan (1989). *Matrix Computations.* second edn. Johns Hopkins Univ. Press. Baltimore.

Gyugyi, P. (1993). Model-Based Control Applied to Rapid Thermal Processing. PhD thesis. Stanford University.

Kailath, T. (1980). *Linear Systems.* Prentice-Hall. New Jersey.

Karmarkar, N. (1984). 'A new polynomial-time algorithm for linear programming'. *Combinatorica* **4**(4), 373–395.

Nesterov, Y. and A. Nemirovsky (1994). *Interior-point polynomial methods in convex programming.* Vol. 13 of *Studies in Applied Mathematics.* SIAM. Philadelphia, PA.

Norman, S. A. (1992). Wafer Temperature Control in Rapid Thermal Processing. PhD thesis. Stanford University.

Paige, C. C. and M. S. Saunders (1982). 'LSQR: An algorithm for sparse linear equations and sparse least squares'. *ACM Transactions on Mathematical Software* **8**(1), 43–71.

Parlett, B. N. and D. S. Scott (1979). 'The Lanczos algorithm with selective orthogonalization'. *Mathematics of Computation* **33**(145), 217–238. American Mathematical Society.

Rockafellar, R. T. (1970). *Convex Analysis*. second edn. Princeton Univ. Press. Princeton.

Vandenberghe, L. and S. Boyd (1993). 'Primal-dual potential reduction method for problems involving matrix inequalities'. To be published in *Math. Programming*.

Vandenberghe, L. and S. Boyd (1994). 'Positive-definite programming'. Submitted to *SIAM Review*.

Ye, Y. (1991). 'An $O(n^3L)$ potential reduction algorithm for linear programming'. *Mathematical Programming* **50**, 239–258.