1 Interior-point methods for large-scale cone programming

Martin Andersen msa@ee.ucla.edu
University of California, Los Angeles
Los Angeles, CA 90095-1594, USA

Joachim Dahl dahl.joachim@gmail.com
MOSEK ApS
Fruebjergvej 3, 2100 København Ø, Denmark

Zhang Liu zhang.liu@gmail.com
Northrop Grumman Corporation
San Diego, CA 92127-2412, USA

Lieven Vandenberghe vandenbe@ee.ucla.edu
University of California, Los Angeles
Los Angeles, CA 90095-1594, USA

In the conic formulation of a convex optimization problem the constraints are expressed as linear inequalities with respect to a possibly non-polyhedral convex cone. This makes it possible to formulate elegant extensions of interior-point methods for linear programming to general nonlinear convex optimization. Recent research on cone programming algorithms has particularly focused on three convex cones, for which symmetric primal-dual methods have been developed: the nonnegative orthant, the second-order cone, and the positive semidefinite matrix cone. Although not all convex constraints can be expressed in terms of the three standard cones, cone programs associated with these cones are sufficiently general to serve as the basis of convex modeling packages. They are also widely used in machine learning. The main difficulty in the implementation of interior-point methods for cone programming is the complexity of the linear equations that need to be solved at each iteration. These equations are usually dense, unlike the equations that arise in linear programming, and it is therefore difficult to develop general-purpose strategies for exploiting problem structure based solely on sparse matrix methods. In this chapter we give an overview of ad hoc *techniques that can be used to exploit non-sparse structure in specific classes of applications. We illustrate the methods with examples from machine learning and present numerical results with CVXOPT, a software*

package that supports the rapid development of customized interior-point methods.

1.1 Introduction

1.1.1 Cone programming

The cone programming formulation has been popular in the recent literature on convex optimization. In this chapter we define a *cone linear program* (cone LP or conic LP) as an optimization problem of the form

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Gx \preceq_C h \\ & Ax = b \end{array} \tag{1.1}$$

with optimization variable x. The inequality $Gx \preceq_C h$ is a *generalized inequality*, which means that $h - Gx \in C$, where C is a closed, pointed, convex cone with nonempty interior. We will also encounter *cone quadratic programs* (cone QPs)

$$\begin{array}{ll} \text{minimize} & (1/2)x^T Px + c^T x \\ \text{subject to} & Gx \preceq_C h \\ & Ax = b, \end{array} \tag{1.2}$$

with P positive semidefinite.

If $C = \mathbb{R}^p_+$ (the nonnegative orthant in $\mathbb{R}^p$) the generalized inequality is a componentwise vector inequality, equivalent to p scalar linear inequalities, and problem (1.1) reduces to a linear program (LP). If C is a nonpolyhedral cone, the problem is substantially more general than an LP, in spite of the similarity in notation. In fact, as Nesterov and Nemirovskii (1994) point out, any convex optimization problem can be reformulated as a cone LP by a simple trick: a general constraint $x \in Q$, where Q is a closed convex set with nonempty interior, can be reformulated in a trivial way as $(x, t) \in C$, $t = 1$, if we define C as the *conic hull* of Q, *i.e.*, $C = \mathbf{cl}\{(x, t) \mid t > 0, (1/t)x \in Q\}$. More important in practice, it turns out that a surprisingly small number of cones is sufficient to express the convex constraints that are most commonly encountered in applications. In addition to the nonnegative orthant, the most common cones are the *second-order cone*

$$\mathcal{Q}_p = \{(y_0, y_1) \in \mathbb{R} \times \mathbb{R}^{p-1} \mid \|y_1\|_2 \leq y_0\}$$

and the *positive semidefinite cone*

$$\mathcal{S}_p = \left\{\mathbf{vec}(U) \mid U \in S^p_+\right\}.$$

Here S_+^p denotes the positive semidefinite matrices of order p and $\mathbf{vec}(U)$ is the symmetric matrix U stored as a vector:

$$\mathbf{vec}(U) = \sqrt{2}\left(\frac{U_{11}}{\sqrt{2}}, U_{21}, \ldots, U_{p1}, \frac{U_{22}}{\sqrt{2}}, U_{32}, \ldots, U_{p2}, \ldots, \frac{U_{p-1,p-1}}{\sqrt{2}}, U_{p,p-1}, \frac{U_{pp}}{\sqrt{2}}\right).$$

(The scaling of the off-diagonal entries ensures that the standard trace inner product of symmetric matrices is preserved, *i.e.*, $\mathrm{Tr}(UV) = \mathbf{vec}(U)^T\,\mathbf{vec}(V)$ for all U, V.) Since the early 1990s a great deal of research has been directed at developing a comprehensive theory and software for modeling optimization problems as cone programs involving the three 'canonical' cones (Nesterov and Nemirovskii, 1994; Boyd et al., 1994; Ben-Tal and Nemirovski, 2001; Alizadeh and Goldfarb, 2003; Boyd and Vandenberghe, 2004). YALMIP and CVX, two modeling packages for general convex optimization, use cone LPs with the three canonical cones as their standard format (Löfberg, 2004; Grant and Boyd, 2007, 2008).

In this chapter we assume that the cone C in (1.1) is a direct product

$$C = C_1 \times C_2 \times \cdots \times C_K, \tag{1.3}$$

where each cone C_i is of one of the three canonical types (nonnegative orthant, second-order cone, or positive semidefinite cone). These cones are self-dual and the dual of the cone LP therefore involves an inequality with respect to the same cone:

$$\begin{array}{ll} \text{maximize} & -h^T z - b^T y \\ \text{subject to} & G^T z + A^T y + c = 0 \\ & z \succeq_C 0. \end{array} \tag{1.4}$$

The cone LP (1.1) is called a *second-order cone program* (SOCP) if C is a direct product of one or more second-order cones. (The nonnegative orthant can be written as a product of second-order cones $\mathcal{Q}_1$ of order 1.) A common and more explicit standard form of an SOCP is

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & \|F_i x + g_i\|_2 \leq d_i^T x + f_i, \quad i = 1, \ldots, K \\ & Ax = b. \end{array} \tag{1.5}$$

This corresponds to choosing

$$G = \begin{bmatrix} -d_1^T \\ -F_1 \\ \vdots \\ -d_K^T \\ -F_K \end{bmatrix}, \qquad h = \begin{bmatrix} f_1 \\ g_1 \\ \vdots \\ f_K \\ g_K \end{bmatrix}, \qquad C = \mathcal{Q}_{p_1} \times \cdots \times \mathcal{Q}_{p_K}$$

in (1.1), if the row dimensions of the matrices F_k are equal to $p_k - 1$.

The cone LP (1.1) is called a *semidefinite program* (SDP) if C is a direct product of positive semidefinite matrix cones. For purposes of exposition a simple standard form with one matrix inequality is sufficient:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & \sum_{i=1}^n x_i F_i \preceq F_0 \\ & Ax = b, \end{array} \tag{1.6}$$

where the coefficients F_i are symmetric matrices of order p and the inequality denotes matrix inequality. This can be seen as the special case of (1.1) obtained by choosing

$$G = \left[\begin{array}{ccc} \mathbf{vec}(F_1) & \cdots & \mathbf{vec}(F_n) \end{array} \right], \qquad h = \mathbf{vec}(F_0), \qquad C = \mathcal{S}_p. \tag{1.7}$$

The SDP (1.6) is in fact as general as the cone LP (1.1) with an arbitrary combination of the three cone types. A componentwise vector inequality $Gx \preceq h$ can be represented as a diagonal matrix inequality $\mathrm{Diag}(Gx) \preceq \mathrm{Diag}(h)$. A second-order cone constraint $\|Fx + g\|_2 \leq d^T x + f$ is equivalent to the linear matrix inequality

$$\left[\begin{array}{cc} d^T x + f & (Fx + g)^T \\ Fx + g & (d^T x + f) I \end{array} \right] \succeq 0.$$

Multiple matrix inequalities can be represented by choosing block-diagonal matrices F_i. For algorithmic purposes, however, it is better to handle the three types of cones separately.

1.1.2 Interior-point methods

Interior-point algorithms have dominated the research on convex optimization methods from the early 1990s until recently. They are popular because they reach a high accuracy in a small number (10–50) of iterations, almost independent of problem size, type, and data. Each iteration requires the solution of a set of linear equations with fixed dimensions and known structure. As a result, the time needed to solve different instances of a given problem family can be estimated quite accurately. Interior-point methods can be extended to handle infeasibility gracefully (Nesterov et al., 1999; Andersen, 2000), by returning a certificate of infeasibility if a problem is primal or dual infeasible. Finally, interior-point methods depend on only a small number of algorithm parameters, which can be set to values that work well for a wide range of data, and do not need to be tuned for a specific problem.

The key to efficiency of an interior-point solver is the set of linear equations solved in each iteration. These equations are sometimes called Newton equations, because they can be interpreted as a linearization of the nonlinear equations that characterize the central path, or Karush-Kuhn-Tucker (KKT) equations, because they can be interpreted as optimality (or KKT) conditions of an equality-constrained quadratic optimization problem. The cost of solving the Newton equations deter-

mines the size of the problems that can be solved by an interior-point method. General-purpose convex optimization packages rely on sparse matrix factorizations to solve the Newton equations efficiently. This approach is very successful in linear programming where problems with several 100,000 variables and constraints are solved routinely. The success of general-purpose sparse linear programming solvers can be attributed to two facts. First, the Newton equations of a sparse LP can usually be reduced to sparse positive definite sets of equations, which can be solved very effectively by sparse Cholesky factorization methods. Second, dense linear programs, which of course are not uncommon in practice, can often be converted into sparse problems by introducing auxiliary variables and constraints. This increases the problem dimensions, but if the resulting problem is sufficiently sparse, the net gain in efficiency is often significant.

For other classes of cone optimization problems (for example, semidefinite programming), the 'sparse linear programming approach' to exploiting problem structure is less effective, either because the Newton equations are not sufficiently sparse, or because the translation of problem structure into sparsity requires an excessive number of auxiliary variables. For these problem classes, it is difficult to develop *general-purpose* techniques that are as efficient and scalable as linear programming solvers. Nevertheless, the recent literature contains many examples of large-scale convex optimization problems that were solved successfully by scalable *customized* implementations of interior-point algorithms (Benson et al., 2000; Roh and Vandenberghe, 2006; Gillberg and Hansson, 2003; Koh et al., 2007; Kim et al., 2007; Joshi and Boyd, 2008; Liu and Vandenberghe, 2009a; Wallin et al., 2009). These results were obtained by a variety of direct and iterative linear algebra techniques that take advantage of non-sparse problem structure. The purpose of this chapter is to give a survey of some of these techniques and illustrate them with applications from machine learning. There is of course a trade-off in how much effort one is prepared to make to optimize performance of an interior-point method for a specific application. We will present results for a software package, CVXOPT (Dahl and Vandenberghe, 2009), that was developed to assist in the development of custom interior-point solvers for specific problem families. It allows the user to specify an optimization problem via an operator description, *i.e.*, by providing functions for evaluating the linear mappings in the constraints, and to supply a custom method for solving the Newton equations. This makes it possible to develop efficient solvers that exploit various types of problem structure, in a fraction of the time needed to write a custom interior-point solver from scratch. Other examples of interior-point software packages that allow customization include the QP solver OOQP (Gertz and Wright, 2003) and the Matlab-based conic solver SDPT3 (Tütüncü et al., 2003).

1.2 Primal-dual interior-point methods

We first describe some implementation details for primal-dual interior-point methods based on the *Nesterov-Todd* scaling (Nesterov and Todd, 1997, 1998). How-

ever, much of the following discussion also applies to other types of primal-dual interior-point methods for second-order cone and semidefinite programming (Helmberg et al., 1996; Kojima et al., 1997; Monteiro and Zhang, 1998).

1.2.1 Newton equations

Consider the cone LP (1.1) and cone QP (1.2). The Newton equations for a primal-dual interior-point method based on the Nesterov-Todd scaling have the form

$$\begin{bmatrix} P & A^T & G^T \\ A & 0 & 0 \\ G & 0 & -W^TW \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} \tag{1.8}$$

(with $P = 0$ for the cone LP). The right-hand sides r_x, r_y, r_z change at each iteration and are defined differently in different algorithms. The matrix W is a *scaling matrix* that depends on the current primal and dual iterates. If the inequalities in (1.1) and (1.4) are generalized inequalities with respect to a cone of the form (1.3), then the scaling matrix W is block-diagonal with K diagonal blocks W_k, defined as follows.

- If C_k is a nonnegative orthant of dimension p ($C_k = \mathbb{R}^p_+$) then W_k is a positive diagonal matrix,

$$W_k = \mathrm{Diag}(d)$$

for some $d \in \mathbb{R}^p_{++}$.

- If C_k is a second-order cone of dimension p ($C_k = \mathcal{Q}_p$) then W_k is a positive multiple of a hyperbolic Householder matrix

$$W_k = \beta(2vv^T - J), \qquad J = \begin{bmatrix} 1 & 0 \\ 0 & -I \end{bmatrix}, \tag{1.9}$$

where $\beta > 0$, $v \in \mathbb{R}^p$ satisfies $v^TJv = 1$, and I is the identity matrix of order $p-1$. The inverse of W_k is given by

$$W_k^{-1} = \frac{1}{\beta}(2Jvv^TJ - J).$$

- If C_k is a positive semidefinite cone of order p ($C_k = \mathcal{S}_p$) then W_k is the matrix representation of a congruence operation: W_k and its transpose are defined by the identities

$$W_k\,\mathbf{vec}(U) = \mathbf{vec}(R^TUR), \qquad W_k^T\,\mathbf{vec}(U) = \mathbf{vec}(RUR^T), \tag{1.10}$$

for all U, where $R \in \mathbb{R}^{p\times p}$ is a nonsingular matrix. The inverses of W_k and W_k^T are

defined by

$$W_k^{-1}\,\mathbf{vec}(U) = \mathbf{vec}(R^{-T}UR^{-1}), \qquad W_k^{-T}\,\mathbf{vec}(U) = \mathbf{vec}(R^{-1}UR^{-T}).$$

The values of the parameters d, β, v, R (or R^{-1}) in these definitions depend on the current primal and dual iterates and are updated after each iteration of the interior-point algorithm.

The number of Newton equations solved per iteration varies with the type of algorithm. It is equal to two in a predictor-corrector method, three in a predictor-corrector method that uses a self-dual embedding, and it can be higher than three if iterative refinement is used. However, since the scaling W is identical for all the Newton equations solved in a single iteration, only one factorization step is required per iteration and the cost per iteration is roughly equal to the cost of solving one Newton equation.

By eliminating Δz, the Newton equation can be reduced to a smaller system

$$\begin{bmatrix} P + G^TW^{-1}W^{-T}G & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_x + G^TW^{-1}W^{-T}r_z \\ r_y \end{bmatrix}. \quad (1.11)$$

The main challenge in an efficient implementation is to exploit structure in the matrices P, G, A, when assembling the matrix

$$P + G^TW^{-1}W^{-T}G = P + \sum_{k=1}^{K} G_k^TW_k^{-1}W_k^{-T}G_k, \quad (1.12)$$

(where G_k is the block row of G corresponding to the kth inequality) and when solving the equation (1.11).

General-purpose solvers for cone programming rely on sparsity in P, G, and A to solve large-scale problems. For example, if the problem does not include equality constraints, one can solve (1.11) by a Cholesky factorization of the matrix (1.12). For pure LPs or QPs (W diagonal) this matrix is typically sparse if P and G are sparse and a sparse Cholesky factorization can be used. In problems that involve all three types of cones it is more difficult to exploit sparsity. Even when P and G are sparse, the matrix (1.12) is often dense. In addition, forming the matrix can be expensive.

1.2.2 Customized implementations

In the following sections we will give examples of techniques for exploiting certain types of non-sparse problem structure in the Newton equations (1.8). The numerical results are obtained using the Python software package CVXOPT, which provides two mechanisms for customizing the interior-point solvers.

- Users can specify the matrices G, A, P in (1.1) and (1.2) as operators, by providing Python functions that evaluate the matrix-vector products and their adjoints.
- Users can provide a Python function for solving the Newton equation (1.8).

This is made straightforward by certain elements of the Python syntax, as the following example illustrates. Suppose we are interested in solving several equations of the form

$$\begin{bmatrix} -I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \tag{1.13}$$

with the same matrix $A \in \mathbb{R}^{m \times n}$ and different right-hand sides b_1, b_2. (We assume $m \leq n$ and $\mathrm{rank}(A) = m$.) The equations can be solved by first solving

$$AA^T x_2 = b_2 + Ab_1,$$

using a Cholesky factorization of AA^T, and then computing x_1 from $x_1 = A^T x_2 - b_1$. The following code defines a Python function `factor()` that computes the Cholesky factorization of $C = AA^T$, and returns a function `solve()` that calculates x_1 and x_2 for a given right-hand side b. A function call `f = factor(A)` therefore returns a function `f` that can be used to compute the solution for a particular right-hand side b as `x1, x2 = f(b)`.

```
from cvxopt import blas, lapack, matrix

def factor(A):
    m, n = A.size
    C = matrix(0.0, (m, m))
    blas.syrk(A, C)             # C := A * A^T.
    lapack.potrf(C)             # Factor C = L * L^T and set C := L.
    def solve(b):
        x2 = b[-m:] + A * b[:n]
        lapack.potrs(C, x2)     # x2 := L^-T * L^-1 * x2.
        x1 = A.T * x2 - b[:n]
        return x1, x2
    return solve
```

Note that the Python syntax proves very useful in this type of applications. For example, Python treats functions as other objects, so the 'factor' function can simply return a 'solve' function. Note also that the symbols `A` and `C` are used in the body of the function `solve()` but are not defined there. To resolve these names, Python therefore looks at the enclosing scope (the function block with the definition of `factor()`). These scope rules make it possible to pass problem-dependent parameters to functions without using global variables.

1.3 Linear and quadratic programming

In the case of a (non-conic) LP or QP the scaling matrix W in the Newton equation (1.8) and (1.11) is a positive diagonal matrix. As already mentioned, general-purpose interior-point codes for linear and quadratic programming are very effective at exploiting sparsity in the data matrices P, G, A. Moreover many types of non-sparse problem structure can be translated into sparsity by adding auxiliary variables and constraints. Nevertheless, even in the case of LPs or QPs, it is sometimes advantageous to exploit problem structure directly by customizing the Newton equation solver. In this section we discuss a few examples.

1.3.1 ℓ_1-Norm approximation

The basic idea is illustrated by the ℓ_1-norm approximation problem

$$\text{minimize} \quad \|Xu - d\|_1, \tag{1.14}$$

with $X \in \mathbb{R}^{m\times n}$, $d \in \mathbb{R}^m$, and variable $u \in \mathbb{R}^n$. This is equivalent to an LP with $m + n$ variables and $2m$ constraints,

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T v \\ \text{subject to} & \left[\begin{array}{cc} X & -I \\ -X & -I \end{array}\right] \left[\begin{array}{c} u \\ v \end{array}\right] \preceq \left[\begin{array}{c} d \\ -d \end{array}\right] \end{array} \tag{1.15}$$

with $\mathbf{1}$ the m-vector with entries equal to one. The reduced Newton equation (1.11) for this LP is

$$\left[\begin{array}{cc} X^T(W_1^{-2} + W_2^{-2})X & X^T(W_2^{-2} - W_1^{-2}) \\ (W_2^{-2} - W_1^{-2})X & W_1^{-2} + W_2^{-2} \end{array}\right] \left[\begin{array}{c} \Delta u \\ \Delta v \end{array}\right] = \left[\begin{array}{c} r_u \\ r_v \end{array}\right] \tag{1.16}$$

where W_1 and W_2 are positive diagonal matrices. (To simplify the notation, we do not propagate the expressions for the right-hand sides when applying block elimination.) By eliminating the variable Δv the Newton equation can be further reduced to an equation

$$X^T D X \Delta u = r,$$

where D is the positive diagonal matrix

$$D = 4W_1^{-2}W_2^{-2}(W_1^{-2} + W_2^{-2})^{-1} = 4(W_1^2 + W_2^2)^{-1}.$$

The cost of solving the ℓ_1-norm approximation problem is therefore equal to a small multiple (10–50) of the cost of solving the same problem in ℓ_2-norm, *i.e.*, solving the normal equations $X^T X u = X^T d$ of the corresponding least-squares problem (Boyd and Vandenberghe, 2004, page 617).

The Python code shown below exploits this fact. The matrix

$$G = \begin{bmatrix} X & -I \\ -X & -I \end{bmatrix}$$

is specified via a Python function `G` that evaluates the matrix-vector products with G and G^T. The function `F` factors the matrix $X^T D X$ and returns a 'solve' routine `f` that takes the right-hand side of (1.8) as its input argument and replaces it with the solution. The input argument of `F` is the scaling matrix W stored as a Python dictionary `W` containing the various parameters of W. The last line calls the CVXOPT cone LP solver. The code can be further optimized by a more extensive use of the BLAS.

```
from cvxopt import lapack, solvers, matrix, mul, div

m, n = X.size

def G(x, y, alpha = 1.0, beta = 0.0, trans = 'N'):
    if trans == 'N':  # y := alpha * G * x + beta * y
        u = X * x[:n]
        y[:m] = alpha * ( u - x[n:] ) + beta * y[:m]
        y[m:] = alpha * (-u - x[n:] ) + beta * y[m:]
    else:             # y := alpha * G' * x + beta * y
        y[:n] =  alpha * X.T * (x[:m] - x[m:]) + beta * y[:n]
        y[n:] = -alpha * (x[:m] + x[m:]) + beta * y[n:]

def F(W):
    d1, d2 = W['d'][:m]**2, W['d'][m:]**2
    D = 4*(d1 + d2)**-1
    A = X.T * spdiag(D) * X
    lapack.potrf(A)
    def f(x, y, z):
        x[:n] += X.T * ( mul( div(d2 - d1, d1 + d2), x[n:] ) +
            mul( .5*D, z[:m] - z[m:] ) )
        lapack.potrs(A, x)
        u = X * x[:n]
        x[n:] = div( x[n:] - div(z[:m], d1) - div(z[m:], d2) +
            mul(d1**-1 - d2**-1, u), d1**-1 + d2**-1 )
        z[:m] = div(u - x[n:] - z[:m], W['d'][:m])
        z[m:] = div(-u - x[n:] - z[m:], W['d'][m:])
    return f

c = matrix(n*[0.0] + m*[1.0])
h = matrix([d, -d])
sol = solvers.conelp(c, G, h, kktsolver = F)
```

Table 1.1 shows the result of an experiment with six randomly generated dense

m	n	CVXOPT	CVXOPT/BLAS	MOSEK (1.15)	MOSEK (1.17)
500	100	0.12	0.06	0.75	0.40
1000	100	0.22	0.11	1.53	0.81
1000	200	0.52	0.29	1.95	1.06
2000	200	1.23	0.60	3.87	2.19
1000	500	2.44	1.32	3.63	2.38
2000	500	5.00	2.68	7.44	5.11
2000	1000	17.1	9.52	32.4	12.8

Table 1.1: Solution times (seconds) for six randomly generated dense ℓ_1-norm approximation problems of dimension $m \times n$. Column 3 gives the CPU times for the customized CVXOPT code. Column 4 gives the CPU times for a customized CVXOPT code with more extensive use of the BLAS for matrix-vector and matrix-matrix multiplications. Columns 5 and 6 shows the times for the interior point solver in MOSEK v6 (with basis identification turned off) applied to the LPs (1.15) and (1.17), respectively.

matrices X. We compare the speed of the customized CVXOPT solver shown above, the same solver with further BLAS optimizations, and the general-purpose LP solver in MOSEK (MOSEK ApS, 2010), applied to the LP (1.15). The last column shows the results for MOSEK applied to the equivalent formulation

$$\begin{array}{ll} \text{minimize} & \mathbf{1}^T v + \mathbf{1}^T w \\ \text{subject to} & Xu - d = v - w \\ & v \succeq 0, \quad w \succeq 0. \end{array} \tag{1.17}$$

The times are in seconds on an Intel Core 2 Quad Q9550 (2.83 GHz) with 4GB of memory.

The table shows that a customized solver, implemented in Python with a modest programming effort, can be competitive with one of the best general-purpose sparse linear programming codes. In this example, the customized solver takes advantage of the fact that the dense matrix X appears in two positions of the matrix G. This property is not exploited by a general-purpose sparse solver.

1.3.2 Least-squares with ℓ_1-norm regularization

As a second example, we consider a least-squares problem with ℓ_1-norm regularization,

$$\text{minimize} \quad \|Xu - d\|_2^2 + \|u\|_1,$$

with $X \in \mathbb{R}^{m \times n}$. The problem is equivalent to a QP

$$\begin{array}{ll} \text{minimize} & (1/2)\|Xu - d\|_2^2 + \mathbf{1}^T v \\ \text{subject to} & -v \preceq u \preceq v, \end{array} \tag{1.18}$$

m	n	CVXOPT	MOSEK (1.18)	MOSEK (1.20)
50	200	0.02	0.35	0.32
50	400	0.03	1.06	0.59
100	1000	0.12	9.57	1.69
100	2000	0.24	66.5	3.43
500	1000	1.19	10.1	7.54
500	2000	2.38	68.6	17.6

Table 1.2: Solution times (seconds) for 6 randomly generated dense least-squares problems with ℓ_1-norm regularization. The matrix X has dimension $m \times n$. Column 3 gives the CPU times for the customized CVXOPT code. Column 4 shows the times for MOSEK applied to (1.18). Column 5 shows the times for MOSEK applied to (1.20).

with $2n$ variables and $2n$ constraints. The reduced Newton equation (1.11) for this QP is

$$\begin{bmatrix} X^T X + W_1^{-2} + W_2^{-2} & W_2^{-2} - W_1^{-2} \\ W_2^{-2} - W_1^{-2} & W_1^{-2} + W_2^{-2} \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta v \end{bmatrix} = \begin{bmatrix} r_u \\ r_v \end{bmatrix}$$

where W_1 and W_2 are diagonal. Eliminating Δv as in the example of section 1.3.1 results in a positive definite equation of order n

$$(X^T X + D)\Delta u = r,$$

where $D = 4(W_1^2 + W_2^2)^{-1}$. Alternatively, we can apply the matrix inversion lemma and convert this to an equation of order m

$$(XD^{-1}X^T + I)\Delta\tilde{u} = \tilde{r}. \tag{1.19}$$

The second option is attractive when $n \gg m$, but requires a customized interior-point solver, since the matrix D depends on the current iterates. A general-purpose QP solver applied to (1.18), on the other hand, is expensive if $n \gg m$, since it does not recognize the low-rank structure of the matrix $X^T X$ in the objective.

Table 1.2 shows the result of an experiment with randomly generated dense matrices X. We compare the speed of a customized QP solver with the general-purpose QP solver in MOSEK applied to the QP (1.18) and the equivalent QP

$$\begin{array}{ll} \text{minimize} & (1/2)w^T w + \mathbf{1}^T v \\ \text{subject to} & -v \preceq x \preceq v \\ & Xu - w = d \end{array} \tag{1.20}$$

with variables u, v, w. Although this last formulation has more variables and constraints than (1.18), MOSEK solves it more efficiently because it is sparser. For the custom solver the choice between (1.18) and (1.20) is irrelevant because the

Newton equations for both QPs reduce to an equation of the form (1.19).

1.3.3 Support vector machine training

A well-known example of the technique in the previous section arises in the training of support vector machine classifiers, via the QP

$$\begin{array}{ll} \text{minimize} & (1/2)u^TQu - d^Tu \\ \text{subject to} & 0 \preceq \text{Diag}(d)u \preceq \gamma\mathbf{1} \\ & \mathbf{1}^Tu = 0 \end{array} \tag{1.21}$$

In this problem Q is the kernel matrix and has entries $Q_{ij} = k(x_i, x_j)$, $i, j = 1, \ldots, N$, where $x_1, \ldots, x_N \in \mathbb{R}^n$ are the training examples and $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a positive definite kernel function. The vector $d \in \{-1, +1\}^N$ contains the labels of the training vectors. The parameter γ is given. The reduced Newton equation for (1.21) is

$$\begin{bmatrix} Q + W_1^{-2} + W_2^{-2} & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix} \begin{bmatrix} \Delta u \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_u \\ r_y \end{bmatrix}. \tag{1.22}$$

This equation is expensive to solve when N is large because the kernel matrix Q is generally dense. If the linear kernel $k(v, \tilde{v}) = v^T\tilde{v}$ is used, the kernel matrix can be written as $Q = XX^T$ where $X \in \mathbb{R}^{N\times n}$ is the matrix with rows x_i^T. If $N \gg n$, we can apply the matrix inversion lemma as in the previous example, and reduce the Newton equation to an equation

$$\left(I + X^T(W_1^{-2} + W_2^{-2})^{-1}X\right)\Delta w = r$$

of order n. This method for exploiting low-rank structure or diagonal-plus-low-rank structure in the kernel matrix Q is well known in machine learning (Ferris and Munson, 2003; Fine and Scheinberg, 2002).

Crammer and Singer (2001) have extended the binary SVM classifier to classification problems with more than two classes. The training problem of the Crammer-Singer multiclass SVM can be expressed as a QP

$$\begin{array}{ll} \text{minimize} & (1/2)\,\text{Tr}(U^TQU) - \text{Tr}(E^TU) \\ \text{subject to} & U \preceq \gamma E \\ & U\mathbf{1}_m = 0 \end{array} \tag{1.23}$$

with a variable $U \in \mathbb{R}^{N\times m}$, where N is the number of training examples and m the number of classes. As in the previous section, Q is a kernel matrix with entries $Q_{ij} = k(x_i, x_j)$, $i, j = 1, \ldots, N$. The matrix $E \in \mathbb{R}^{N\times m}$ is defined as

$$E_{ij} = \begin{cases} 1 & \text{training example } i \text{ belongs to class } j \\ 0 & \text{otherwise.} \end{cases}$$

The inequality $U \preceq \gamma E$ denotes componentwise inequality between matrices. From the optimal solution U one obtains the multiclass classifier, which maps a test point x to the class number

$$\operatorname*{argmax}_{j=1,\ldots,m} \sum_{i=1}^{N} U_{ij} k(x_i, x).$$

An important drawback of this formulation, compared to multiclass classifiers based on a combination of binary classifiers, is the high cost of solving the QP (1.23), which has Nm variables, Nm inequality constraints, and N equality constraints. Let us therefore examine the reduced Newton equations,

$$\begin{bmatrix} Q + W_1^{-2} & 0 & \cdots & 0 & I \\ 0 & Q + W_2^{-2} & \cdots & 0 & I \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & Q + W_m^{-2} & I \\ I & I & \cdots & I & 0 \end{bmatrix} \begin{bmatrix} \Delta u_1 \\ \Delta u_2 \\ \vdots \\ \Delta u_m \\ \Delta y \end{bmatrix} = \begin{bmatrix} r_{u_1} \\ r_{u_2} \\ \vdots \\ r_{u_m} \\ r_y \end{bmatrix}$$

with variables Δu_k, $\Delta y \in \mathbb{R}^N$. The variables Δu_k are the columns of the search direction ΔU corresponding to the variable U in (1.23). Eliminating the variables Δu_k gives an equation $H \Delta y = r$ with

$$H = \sum_{k=1}^{m} (Q + W_k^{-2})^{-1}.$$

Now suppose the linear kernel is used, and $Q = XX^T$ with $X \in \mathbb{R}^{N \times n}$ and N large (compared to mn). Then we can exploit the low rank structure in Q and write H as

$$\begin{aligned} H &= \sum_{k=1}^{m} \left(W_k^2 - W_k^2 X (I + X^T W_k^2 X)^{-1} X^T W_k^2 \right) \\ &= D - YY^T \end{aligned}$$

where $D = \sum_k W_k^2$ is diagonal and Y is an $N \times mn$ matrix

$$Y = \begin{bmatrix} W_1^2 X L_1^{-1} & W_2^2 X L_2^{-1} & \cdots & W_m^2 X L_m^{-1} \end{bmatrix}$$

where L_k is a Cholesky factor of $I + X^T W_k^2 X = L_k L_k^T$. A second application of the matrix inversion lemma gives

$$\begin{aligned} \Delta y &= (D - YY^T)^{-1} r \\ &= \left(D^{-1} + D^{-1} Y (I + Y^T D^{-1} Y)^{-1} Y^T D^{-1} \right) r. \end{aligned}$$

The largest dense matrix that needs to be factored in this method is the $mn \times mn$ matrix $I + Y^T D^{-1} Y$. For large N the cost is dominated by the matrix products $X^T W_i^2 D^{-1} W_j^2 X$, $i, j = 1, \ldots, m$, needed to compute $Y^T D^{-1} Y$. This

N	time	iterations	test error
10000	5699	27	8.6%
20000	12213	33	4.0%
30000	35738	38	2.7%
40000	47950	39	2.0%
50000	63592	42	1.6%
60000	82810	46	1.3%

Table 1.3: Solution times (seconds) and numbers of iterations for the multiclass SVM training problem applied to the MNIST set of handwritten digits ($m = 10$ classes, $n = 785$ features).

takes $O(m^2n^2N)$ operations.

In table 1.3 we show computational results for the multiclass classifier applied to the MNIST handwritten digit data set (LeCun and Cortes, 1998). The images are 28×28. We add a constant feature to each training example, so the dimension of the feature space is $n = 1 + 28^2 = 785$. We use $\gamma = 10^5/N$. For the largest N, the QP (1.23) has $600,000$ variables and inequality constraints, and 60000 equality constraints.

1.4 Second-order cone programming

Several authors have provided detailed studies of techniques for exploiting sparsity in SOCPs (Andersen et al., 2003; Goldfarb and Scheinberg, 2005). The coefficient matrix (1.12) of the reduced Newton equation of a linear and quadratic cone program with K second-order cone constraints of dimension $p_1, \ldots, p_K$, is

$$P + \sum_{k=1}^{K} G_k^T W_k^{-2} G_k, \qquad W_k^{-1} = \frac{1}{\beta_k}(2Jv_kv_k^TJ - J). \tag{1.24}$$

The scaling matrices are parameterized by parameters $\beta_k > 0$ and $v_k \in \mathbb{R}^{p_k}$ with $v_k^TJv_k = 1$ and J the sign matrix defined in (1.9). We note that

$$W_k^{-2} = \frac{1}{\beta^2}(2w_kw_k^T - J) = \frac{1}{\beta^2}(I + 2w_kw_k^T - 2e_0e_0^T), \qquad w_k = \begin{bmatrix} v_k^Tv_k \\ -2v_{k0}v_{k1} \end{bmatrix}$$

where e_0 is the first unit vector in $\mathbb{R}^p$, v_{k0} is the first entry of v_k, and v_{k1} is the $(p-1)$-vector of the other entries. Therefore

$$G_k^TW_k^{-2}G_k = \frac{1}{\beta^2}\left(G_k^TG_k + 2(G_k^Tw_k)(G_k^Tw_k)^T - 2(G_k^Te_0)(G_k^Te_0)^T\right),$$

i.e., a multiple of $G_k^TG_k$ plus a rank-two term.

We can distinguish two cases when examining the sparsity of the sum (1.24). If

the dimensions p_k of the second-order cones are small, then the matrices G_k are likely to have many zero columns and the vectors $G_k^T w_k$ will be sparse (for generic dense w_k). Therefore the products $G_k^T W_k^{-2} G_k$ and the entire matrix (1.24) are likely to be sparse. At the extreme end ($p_k = 1$) this reduces to the situation in linear programming where the matrix (1.12) has the sparsity of $P + G^T G$.

The second case arises when the dimensions p_k are large. Then $G_k^T w_k$ is likely to be dense which results in a dense matrix (1.24). If $K \ll n$, we can still separate the sum (1.24) in a sparse part and a few dense rank-one terms, and apply techniques for handling dense rows in sparse linear programs (Andersen et al., 2003; Goldfarb and Scheinberg, 2005).

Robust support vector machine training

Second-order cone programming has found wide application in *robust optimization*. As an example, we discuss the robust SVM formulation of Shivaswamy et al. (2006). This problem can be expressed as a cone QP with second-order cone constraints:

$$\begin{array}{ll} \text{minimize} & (1/2)w^T w + \gamma \mathbf{1}^T v \\ \text{subject to} & \mathrm{Diag}(d)(Xw + b\mathbf{1}) \succeq \mathbf{1} - v + Eu \\ & v \succeq 0 \\ & \|S_j w\|_2 \leq u_j, \quad j = 1, \ldots, t. \end{array} \tag{1.25}$$

The variables are $w \in \mathbb{R}^n$, $b \in \mathbb{R}$, $v \in \mathbb{R}^N$, and $u \in \mathbb{R}^t$. The matrix $X \in \mathbb{R}^{N \times n}$ has as its rows the training examples x_i^T and the vector $d \in \{-1, 1\}^N$ contains the training labels. For $t = 0$, the term Eu and the norm constraints are absent, and the problem reduces to the standard linear SVM

$$\begin{array}{ll} \text{minimize} & (1/2)w^T w + \gamma \mathbf{1}^T v \\ \text{subject to} & d_i(x_i^T w + b) \geq 1 - v_i, \quad i = 1, \ldots, N \\ & v \succeq 0. \end{array} \tag{1.26}$$

In problem (1.25) the inequality constraints in (1.26) are replaced by a robust version that incorporates a model of the uncertainty in the training examples. The uncertainty is described by t matrices S_j, with t ranging from 1 to N, and an $N \times n$-matrix E with 0-1 entries and exactly one entry equal to one in each row. The matrices S_j can be assumed to be symmetric positive semidefinite. To interpret the constraints, suppose $E_{ij} = 1$. Then the constraint in (1.25) that involves training example x_i can be written as a second-order cone constraint

$$d_i(x_i^T w + b) - \|S_j w\|_2 \geq 1 - v_i.$$

This is equivalent to

$$\inf_{\|\eta\|_2 \leq 1} \left(d_i(x_i + S_j \eta)^T w + b \right) \geq 1 - v_i.$$

N	$t = 2$	$t = 10$	$t = 50$	$t = 100$
4000	2.5	2.8	4.1	5.0
8000	5.4	5.3	6.0	6.9
16000	12.5	12.5	12.7	13.7

Table 1.4: Solution times (seconds) for customized interior-point method for robust SVM training ($n = 200$ features and t different uncertainty models).

In other words, we replace the training example x_i with an ellipsoid $\{x_i + S_j\eta \mid \|\eta\|_2 \leq 1\}$ and require that $d_i(x^T w + b) \geq 1 - v_i$ holds for all x in the ellipsoid. The matrix S_j defines the shape and magnitude of the uncertainty about training example i. If we take $t = 1$, we assume that all training examples are subject to the same type of uncertainty. Values of t larger than one allow us to use different uncertainty models for different subsets of the training examples.

To evaluate the merits of the robust formulation it is useful to compare the cost of solving the robust and non-robust problems. Recall that the cost per iteration of an interior-point method applied to the QP (1.26) is of order Nn^2 if $N \geq n$, and dominated by an equation of the form $(I + X^T DX)\Delta w = r$ with D positive diagonal. To determine the cost of solving the robust problem we write it in the standard cone QP form (1.2) by choosing $x = (w, b, v, u) \in \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^N \times \mathbb{R}^t$, $K = 1 + t$, $C = \mathbb{R}^{2N}_+ \times \mathcal{Q}_{n+1} \times \cdots \mathcal{Q}_{n+1}$. We have

$$P = \begin{bmatrix} I & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad G = \begin{bmatrix} -\operatorname{Diag}(d)X & -d & -I & E \\ 0 & 0 & -I & 0 \\ 0 & 0 & 0 & -e_1^T \\ -S_1 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & -e_t^T \\ -S_t & 0 & 0 & 0 \end{bmatrix}$$

where e_k is the kth unit vector in $\mathbb{R}^t$. We can note that $E^T DE$ is diagonal for any diagonal matrix D, and this property makes it inexpensive to eliminate the extra variable Δu from the Newton equations. As in the nonrobust case, the Newton equations can then further be reduced to an equation in n variables Δw. The cost of forming the reduced coefficient matrix is of order $Nn^2 + tn^3$. When $n \leq N$ and for modest values of t, the cost of solving the robust counterpart of the linear SVM training problem is therefore comparable with the standard non-robust linear SVM.

Table 1.4 shows the solution times for a customized CVXOPT interior-point method applied to randomly generated test problems with $n = 200$ features. Each training vector is assigned to one of t uncertainty models. For comparison, the general-purpose solver SDPT3 v.4 called from CVX takes about 130 seconds for $t = 50$ and $N = 4000$ training vectors.

1.5 Semidefinite programming

We now turn to the question of exploiting problem structure in cone programs that include linear matrix inequalities. To simplify the notation we explain the ideas for the inequality form SDP (1.6).

Consider the coefficient matrix $H = G^T W^{-1} W^{-T} G$ of the reduced Newton equations, with G defined in (1.7) and the scaling matrix W in (1.10). The entries of H are

$$H_{ij} = \mathrm{Tr}\left(R^{-1} F_i R^{-T} R^{-1} F_j R^{-T}\right), \quad i, j = 1, \ldots, n. \tag{1.27}$$

The matrix R is generally dense, and therefore the matrix H is usually dense, so the equation $H\Delta x = r$ must be solved by a dense Cholesky factorization. The cost of evaluating the expressions (1.27) is also significant and often exceeds the cost of solving the system. For example, if $p = O(n)$ and the matrices F_i are dense, then it takes $O(n^4)$ operations to compute the entire matrix H and $O(n^3)$ to solve the system.

Efforts to exploit problem structure in SDPs have focused on using sparsity and low-rank structure in the coefficient matrices F_i to reduce the cost of assembling H. Sparsity is exploited, in varying degrees, by all general-purpose SDP solvers (Sturm, 1999, 2002; Tütüncü et al., 2003; Yamashita et al., 2003; Benson and Ye, 2005; Borchers, 1999). Several of these techniques use ideas from the theory of chordal sparse matrices and positive definite matrix completion theory to reduce the problem size or speed up critical calculations (Fukuda et al., 2000; Nakata et al., 2003; Burer, 2003; Andersen et al., 2010). It was also recognized early on that low-rank structure in the coefficients F_i can be very useful to reduce the complexity of interior-point methods (Gahinet and Nemirovski, 1997; Benson et al., 2000). For example if $F_i = a_i a_i^T$, then it can be verified that

$$H = (A^T R^{-T} R^{-1} A) \circ (A^T R^{-T} R^{-1} A)$$

where A is the matrix with columns a_i and $\circ$ is componentwise matrix multiplication. This expression for H takes only $O(n^3)$ operations to evaluate if $p = O(n)$. Low-rank structure is exploited in the LMI Control Toolbox (Gahinet et al., 1995), DSDP (Benson and Ye, 2005), and SDPT3 (Tütüncü et al., 2003). Recent applications of dense, low-rank structure include SDPs derived from sum-of-squares formulations of nonnegative polynomials (Löfberg and Parrilo, 2004; Roh and Vandenberghe, 2006; Roh et al., 2007; Liu and Vandenberghe, 2007). Kandola et al. (2003) describe an application in machine learning.

Sparsity and low-rank structure do not exhaust the useful types of problem structure that can be exploited in SDP interior-point methods, as demonstrated by the following two examples.

1.5.1 SDPs with upper bounds

A simple example from (Toh et al., 2007; Nouralishahi et al., 2008) will illustrate the limitations of techniques based on sparsity. Consider a standard form SDP with an added upper bound

$$\begin{array}{ll} \text{minimize} & \mathrm{Tr}(CX) \\ \text{subject to} & \mathrm{Tr}(A_i X) = b_i, \quad i = 1, \ldots, m \\ & 0 \preceq X \preceq I. \end{array} \tag{1.28}$$

The variable X is a symmetric matrix of order p. Since general-purpose SDP solvers do not accept this format directly, the problem needs to be reformulated as one without upper bounds. An obvious reformulation is to introduce a slack variable S and solve the standard form SDP

$$\begin{array}{ll} \text{minimize} & \mathrm{Tr}(CX) \\ \text{subject to} & \mathrm{Tr}(A_i X) = b_i, \quad i = 1, \ldots, m \\ & X + S = I \\ & X \succeq 0, \quad S \succeq 0. \end{array} \tag{1.29}$$

This is the semidefinite programming analog of converting an LP with variable bounds,

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & 0 \preceq x \preceq \mathbf{1}, \end{array}$$

into a standard form LP

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b, \quad x + s = \mathbf{1} \\ & x \succeq 0, \quad s \succeq 0. \end{array} \tag{1.30}$$

Even though this is unnecessary in practice (LP solvers usually handle variable upper bounds directly), the transformation to (1.30) would have only a minor effect on the complexity. In (1.30) we add n extra variables (assuming the dimension of x is n), and n extremely sparse equality constraints. A good LP solver that exploits the sparsity will solve the LP at roughly the same cost as the corresponding problem without upper bounds. The situation is very different for SDPs. In (1.29) we increase the number of equality constraints from m to $m + p(p+1)/2$. SDP solvers are not as good at exploiting sparsity as LP solvers, so (1.29) is much harder to solve using general-purpose solvers than the corresponding problem without upper bound.

Nevertheless, the SDP with upper bounds can be solved at a cost comparable to the standard form problem, via a technique proposed by Toh et al. (2007); Nouralishahi et al. (2008). The reduced Newton equations (1.11) for the SDP with

upper bounds (1.29) are

$$T_1\Delta X T_1 + T_2\Delta X T_2 + \sum_{i=1}^m \Delta y_i A_i = r_X \tag{1.31a}$$

$$\mathrm{Tr}(A_i\Delta X) = r_{y_i}, \quad i = 1,\ldots,m \tag{1.31b}$$

where $T_1 = R_1^{-T}R_1^{-1}$ and $T_2 = R_2^{-T}R_2^{-1}$ are positive definite matrices. (The Newton equations for the standard form problem (1.28) are similar, but have only one term $T\Delta XT$ in the first equation, making it easy to eliminate ΔX.)

To solve (1.31) we first determine a congruence transformation that simultaneously diagonalizes T_1 and T_2,

$$V^T T_1 V = I, \qquad V^T T_2 V = \mathrm{Diag}(\gamma),$$

where γ is a positive vector (see (Golub and Loan, 1996, §8.7.2)). If we define $\Delta\tilde{X} = V^{-1}\Delta X V^{-T}$, $\tilde{A}_i = V^T A_i V$, the equations reduce to

$$\begin{aligned} \Delta\tilde{X} + \mathrm{Diag}(\gamma)\Delta\tilde{X}\,\mathrm{Diag}(\gamma) + \sum_{i=1}^m \Delta y_i\tilde{A}_i &= V^T r_X V \\ \mathrm{Tr}(\tilde{A}_i\Delta\tilde{X}) &= r_{y_i}, \quad i = 1,\ldots,m. \end{aligned}$$

From the first equation, we can express $\Delta\tilde{X}$ in terms of Δy:

$$\Delta\tilde{X} = (V^T r_X V)\circ\Gamma - \sum_{i=1}^m \Delta y_i(\tilde{A}_i\circ\Gamma) \tag{1.32}$$

where Γ is the symmetric matrix with entries $\Gamma_{ij} = 1/(1+\gamma_i\gamma_j)$. Substituting this in the second equation gives a set of equations $H\Delta y = r$ where

$$H_{ij} = \mathrm{Tr}(\tilde{A}_i(\tilde{A}_j\circ\Gamma)) = \mathrm{Tr}((\tilde{A}_i\circ\tilde{A}_j)\Gamma)), \quad i,j = 1,\ldots,m.$$

After solving for Δy, one easily obtains ΔX from (1.32). The cost of this method is dominated by the cost of computing the matrices $\tilde{A}_i$ ($O(p^4)$ flops if $m = O(p)$), the cost of assembling H ($O(p^4)$), and the cost of solving for Δy ($O(p^3)$). For dense coefficient matrices A_i, the overall cost is comparable to the cost of solving the Newton equations for the standard form SDP (1.28) without upper bound.

Table 1.5 shows the time per iteration of a CVXOPT implementation of the method described above. The test problems are randomly generated, with $m = p$, and dense coefficient matrices A_i. The general-purpose SDP solver SDPT3 v.4 called from CVX applied to the problem (1.29) with $m = p = 100$ takes about 23 seconds per iteration.

1.5.2 Nuclear norm approximation

In section 1.3.1 we discussed the ℓ_1-norm approximation problem (1.14) and showed that the cost per iteration of an interior-point method is comparable to the cost of solving the corresponding least-squares problem (*i.e.*, $O(mn^2)$ operations). We can

$m = p$	time per iteration
50	0.05
100	0.33
200	2.62
300	10.5
400	30.4
500	70.8

Table 1.5: Time (seconds) per iteration of a customized interior-point method for SDPs with upper bounds.

ask the same question about the matrix counterpart of ℓ_1-norm approximation, the *nuclear norm* approximation problem

$$\text{minimize} \quad \|X(u) - D\|_*. \tag{1.33}$$

Here $\|\cdot\|_*$ denotes the nuclear matrix norm (sum of singular values) and $X(u) = \sum_{i=1}^n u_i X_i$ is a linear mapping from $\mathbb{R}^n$ to $\mathbb{R}^{p\times q}$. The nuclear norm is popular in convex heuristics for rank minimization problems in system theory and machine learning (Fazel et al., 2001; Fazel, 2002; Fazel et al., 2004; Recht et al., 2010; Candès and Plan, 2010). These heuristics extend ℓ_1-norm heuristics for sparse optimization.

Problem (1.33) is equivalent to an SDP

$$\begin{array}{ll} \text{minimize} & (\operatorname{Tr} V_1 + \operatorname{Tr} V_2)/2 \\ \text{subject to} & \left[\begin{array}{cc} V_1 & X(u) - D \\ (X(u) - D)^T & V_2 \end{array}\right] \succeq 0, \end{array} \tag{1.34}$$

with auxiliary symmetric matrix variables V_1, V_2. The presence of the extra variables V_1 and V_2 clearly makes solving (1.34) using a general-purpose SDP solver very expensive, unless p and q are small, and much more expensive than solving the corresponding least-squares approximation problem (*i.e.*, problem (1.33) with the Frobenius norm substituted for the nuclear norm).

A specialized interior-point method is described in (Liu and Vandenberghe, 2009a). The basic idea can be summarized as follows. The Newton equations for (1.34) are

$$\Delta Z_{11} = r_{V_1}, \qquad \Delta Z_{22} = r_{V_2}, \qquad \operatorname{Tr}(X_i^T \Delta Z_{12}) = r_{u_i}, \quad i = 1, \ldots, n$$

and

$$\left[\begin{array}{cc} \Delta V_1 & X(\Delta u) \\ X(\Delta u)^T & \Delta V_2 \end{array}\right] + T \left[\begin{array}{cc} \Delta Z_{11} & \Delta Z_{12} \\ \Delta Z_{12}^T & \Delta Z_{22} \end{array}\right] T = r_Z,$$

where $T = RR^T$. The variables ΔZ_{11}, ΔZ_{22}, ΔV_1, ΔV_2 are easily eliminated, and

$n = p = 2q$	time per iteration
100	0.30
200	2.33
300	8.93
400	23.9
500	52.4

Table 1.6: Time (seconds) per iteration of a customized interior-point method for the nuclear norm approximation problem.

the equations reduce to

$$\begin{aligned} X(\Delta u) + T_{11}\Delta Z_{12}T_{22} + T_{12}\Delta Z_{12}^T T_{12} &= r_{Z_{12}} \\ \mathrm{Tr}(X_i^T \Delta Z_{12}) &= r_{u_i}, \quad i = 1, \ldots, n, \end{aligned}$$

where T_{ij} are subblocks of T partitioned as the matrix in the constraint (1.34). The method of Liu and Vandenberghe (2009a) is based on applying a transformation that reduces T_{11} and T_{22} to identity matrices and T_{12} to a (rectangular) diagonal matrix, and then eliminating ΔZ_{12} from the first equation, to obtain a dense linear system in Δu. The cost of solving the Newton equations is $O(n^2pq)$ operations if $n \geq \max\{p, q\}$. For dense X_i this is comparable to the cost of solving the approximation problem in the least-squares (Frobenius-norm) sense.

Table 1.6 shows the time per iteration of a CVXOPT code for (1.34). The problems are randomly generated with $n = p = 2q$. Note that the SDP (1.34) has $n + p(p+1)/2 + q(q+1)/2$ variables and is very expensive to solve by general-purpose interior-point codes. CVX/SDPT3 applied to (1.33) takes 22 seconds per iteration for the first problem ($n = p = 100$, $q = 50$).

1.6 Conclusion

Interior-point algorithms for conic optimization are attractive in machine learning and other applications because they converge to a high accuracy in a small number of iterations and are quite robust with respect to data scaling. The main disadvantages are the high memory requirements and linear algebra complexity associated with the linear equations that are solved at each iteration. It is therefore critical to exploit problem structure when solving large problems. For linear and quadratic programming, sparse matrix techniques provide a general and effective approach to handling problem structure. For nonpolyhedral cone programs, and semidefinite programs in particular, the sparse approach is less effective, for two reasons. First, translating non-sparse problem structure into a sparse model may require introducing a very large number of auxiliary variables and constraints. Second, techniques for exploiting sparsity in SDPs are less well developed than for LPs. It is therefore

difficult to develop *general-purpose* techniques for exploiting problem structure in cone programs that are as scalable as sparse linear programming solvers. However, it is sometimes quite straightforward to find special-purpose techniques that exploit various types of problem structure. When this is the case, customized implementations can be developed that are orders of magnitudes more efficient than general-purpose interior-point implementations.

References

F. Alizadeh and D. Goldfarb. Second-order cone programming. *Mathematical Programming Series B*, 95:3–51, 2003.

E. D. Andersen. On primal and dual infeasibility certificates in a homogeneous model for convex optimization. *SIAM Journal on Optimization*, 11(2):380–388, 2000.

E. D. Andersen, C. Roos, and T. Terlaky. On implementing a primal-dual interior-point method for conic quadratic optimization. *Mathematical Programming*, 95 (2):249–277, 2003.

M. S. Andersen, J. Dahl, and L. Vandenberghe. Implementation of nonsymmetric interior-point methods for linear optimization over sparse matrix cones. *Mathematical Programming Computation*, 2010. URL `http://dx.doi.org/10.1007/s12532-010-0016-2`.

A. Ben-Tal and A. Nemirovski. *Lectures on Modern Convex Optimization. Analysis, Algorithms, and Engineering Applications*. SIAM, Philadelphia, PA, 2001.

S. J. Benson and Y. Ye. DSDP5: Software for semidefinite programming. Technical Report ANL/MCS-P1289-0905, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, September 2005. URL `http://www.mcs.anl.gov/hs/software/DSDP`.

S. J. Benson, Y. Ye, and X. Zhang. Solving large-scale sparse semidefinite programs for combinatorial optimization. *SIAM Journal on Optimization*, 10:443–461, 2000.

B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11(1):613–623, 1999.

S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004. `www.stanford.edu/~boyd/cvxbook`.

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

S. Burer. Semidefinite programming in the space of partial positive semidefinite matrices. *SIAM Journal on Optimization*, 14(1):139–172, 2003.

E. J. Candès and Y. Plan. Matrix completion with noise. *Proceedings of the IEEE*,

98(6):925–936, 2010.

K. Crammer and Y. Singer. On the algorithmic implementation of the multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

J. Dahl and L. Vandenberghe. *CVXOPT: A Python Package for Convex Optimization*. `abel.ee.ucla.edu/cvxopt`, 2009.

M. Fazel. *Matrix Rank Minimization with Applications*. PhD thesis, Stanford University, 2002.

M. Fazel, H. Hindi, and S. Boyd. A rank minimization heuristic with application to minimum order system approximation. In *Proceedings of the American Control Conference*, pages 4734–4739, 2001.

M. Fazel, H. Hindi, and S. Boyd. Rank minimization and applications in system theory. In *Proceedings of American Control Conference*, pages 3273–3278, 2004.

M. C. Ferris and T. S. Munson. Interior-point methods for massive support vector machines. *SIAM Journal on Optimization*, 13(3):783–804, 2003.

S. Fine and K. Scheinberg. Efficient SVM training using low-rank kernel representations. *Journal of Machine Learning Research*, 2:243–264, 2002.

M. Fukuda, M. Kojima, K. Murota, and K. Nakata. Exploiting sparsity in semidefinite programming via matrix completion I: general framework. *SIAM Journal on Optimization*, 11:647–674, 2000.

P. Gahinet and A. Nemirovski. The projective method for solving linear matrix inequalities. *Mathematical Programming*, 77(2):163–190, May 1997.

P. Gahinet, A. Nemirovski, A. J. Laub, and M. Chilali. *LMI Control Toolbox*. The MathWorks, Inc., 1995.

E. M. Gertz and S. J. Wright. Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29(1):81, 2003.

J. Gillberg and A. Hansson. Polynomial complexity for a Nesterov-Todd potential-reduction method with inexact search directions. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, volume 3, pages 3824–3829, 2003.

D. Goldfarb and K. Scheinberg. Product-form Cholesky factorization in interior point methods for second-order cone programming. *Mathematical Programming Series A*, 103:153–179, 2005.

G. H. Golub and C. F. Van Loan. *Matrix Computations*. John Hopkins University Press, 3rd edition, 1996.

M. Grant and S. Boyd. *CVX: Matlab software for disciplined convex programming (web page and software)*. `http://stanford.edu/~boyd/cvx`, 2007.

M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control (a tribute to M. Vidyasagar)*. Springer, 2008.

C. Helmberg, F. Rendl, R. J. Vanderbei, and H. Wolkowicz. An interior-point

method for semidefinite programming. *SIAM Journal on Optimization*, 6(2): 342–361, 1996.

S. Joshi and S. Boyd. An efficient method for large-scale gate sizing. *IEEE Transactions on Circuits and Systems I*, 55(9), 2008. To appear.

J. Kandola, T. Graepel, and J. Shawe-Taylor. Reducing kernel matrix diagonal dominance using semi-definite programming. In B. Schölkopf and M. H. Warmuth, editors, *Learning theory and Kernel machines, 16th Annual Conference on Learning Theory and 7th Kernel Workshop*, pages 288–302, 2003.

S.-J. Kim, K. Koh, M. Lustig, S. Boyd, and D. Gorinevsky. An interior-point method for large-scale ℓ_1-regularized least squares. *IEEE Journal on Selected Topics in Signal Processing*, 1(4):606–617, 2007.

K. Koh, S.-J. Kim, and S. Boyd. An interior-point method for large-scale ℓ_1-regularized logistic regression. *Journal of Machine Learning Research*, 8:1519–1555, 2007.

M. Kojima, S. Shindoh, and S. Hara. Interior-point methods for the monotone linear complementarity problem in symmetric matrices. *SIAM Journal on Optimization*, 7:86–125, February 1997.

Y. LeCun and C. Cortes. The MNIST database of handwritten digits. Available at `http://yann.lecun.com/exdb/mnist/`, 1998.

Z. Liu and L. Vandenberghe. Low-rank structure in semidefinite programs derived from the KYP lemma. In *Proceedings of the 46th IEEE Conference on Decision and Control*, pages 5652–5659, 2007.

Z. Liu and L. Vandenberghe. Interior-point method for nuclear norm approximation with application to system identification. *SIAM Journal on Matrix Analysis and Applications*, 31:1235–1256, 2009a.

Z. Liu and L. Vandenberghe. Semidefinite programming methods for system realization and identification. In *Proceedings of the 48th IEEE Conference on Decision and Control*, pages 4676–4681, 2009b.

J. Löfberg. YALMIP : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

J. Löfberg and P. A. Parrilo. From coefficients to samples: a new approach to SOS optimization. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, pages 3154–3159, 2004.

R. D. C. Monteiro and Y. Zhang. A unified analysis for a class of long-step primal-dual path-following interior-point algorithms for semidefinite programming. *Mathematical Programming*, 81:281–299, 1998.

MOSEK ApS. *The MOSEK Optimization Tools Manual. Version 6.0.*, 2010. Available from `www.mosek.com`.

K. Nakata, K. Fujitsawa, M. Fukuda, M. Kojima, and K. Murota. Exploiting sparsity in semidefinite programming via matrix completion II: implementation and numerical details. *Mathematical Programming Series B*, 95:303–327, 2003.

Yu. Nesterov and A. Nemirovskii. *Interior-point polynomial methods in convex programming*, volume 13 of *Studies in Applied Mathematics*. SIAM, Philadelphia, PA, 1994.

Yu. Nesterov, M. J. Todd, and Y. Ye. Infeasible-start primal-dual methods and infeaibility detectors for nolinear programming problems. *Mathematical Programming*, 84(2):227–267, 1999.

Yu. E. Nesterov and M. J. Todd. Self-scaled barriers and interior-point methods for convex programming. *Mathematics of Operations Research*, 22(1):1–42, 1997.

Yu. E. Nesterov and M. J. Todd. Primal-dual interior-point methods for self-scaled cones. *SIAM Journal on Optimization*, 8(2):324–364, May 1998.

M. Nouralishahi, C. Wu, and L. Vandenberghe. Model calibration for optical lithography via semidefinite programming. *Optimization and Engineering*, 9: 19–35, 2008.

B. Recht, M. Fazel, and P. A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. *SIAM Review*, 52(3):471–501, 2010.

T. Roh and L. Vandenberghe. Discrete transforms, semidefinite programming, and sum-of-squares representations of nonnegative polynomials. *SIAM Journal on Optimization*, 16(4):939–964, 2006.

T. Roh, B. Dumitrescu, and L. Vandenberghe. Multidimensional FIR filter design via trigonometric sum-of-squares optimization. *IEEE Journal of Selected Topics in Signal Processing*, 1:641–650, 2007.

P. K. Shivaswamy, C. Bhattacharyya, and A. J. Smola. Second order cone programming approaches for handling missing and uncertain data. *Journal of Machine Learning Research*, 7:1283–1314, 2006.

J. F. Sturm. Using SEDUMI 1.02, a Matlab toolbox for optimization over symmetric cones. *Optimization Methods and Software*, 11-12:625–653, 1999.

J. F. Sturm. Implementation of interior point methods for mixed semidefinite and second order cone optimization problems. *Optimization Methods and Software*, 17(6):1105–1154, 2002.

K. C. Toh, R. H. Tütüncü, and M. J. Todd. Inexact primal-dual path-following algorithms for a special class of convex quadratic SDP and related problems. *Pacific Journal of Optimization*, 3, 2007.

R. H. Tütüncü, K. C. Toh, and M. J. Todd. Solving semidefinite-quadratic-linear programs using SDPT3. *Mathematical Programming Series B*, 95:189–217, 2003.

R. Wallin, A. Hansson, and J. H. Johansson. A structure exploiting preprocessor for semidefinite programs derived from the Kalman-Yakubovich-Popov lemma. *IEEE Transactions on Automatic Control*, 54(4):697–704, 2009.

M. Yamashita, K. Fujisawa, and M. Kojima. Implementation and evaluation of SDPA 6.0 (Semidefinite Programming Algorithm 6.0). *Optimization Methods and Software*, 18(4):491–505, 2003.