Mathematical Programming 58 (1993) 1-32
North-Holland

Symmetric indefinite systems for interior point methods

Robert J. Vanderbei and Tamra J. Carpenter

Program in Statistics and Operations Research, Princeton University, Princeton, NJ, USA

Received 2 May 1991
Revised manuscript received 22 July 1992

We present a unified framework for solving linear and convex quadratic programs via interior point methods. At each iteration, this method solves an indefinite system whose matrix is $\left[\begin{smallmatrix} -D^{-2} & A^{\mathrm{T}} \\ A & 0 \end{smallmatrix}\right]$ instead of reducing to obtain the usual AD^2A^{T} system. This methodology affords two advantages: (1) it avoids the fill created by explicitly forming the product AD^2A^{T} when A has dense columns; and (2) it can easily be used to solve nonseparable quadratic programs since it requires only that D be symmetric. We also present a procedure for converting nonseparable quadratic programs to separable ones which yields computational savings when the matrix of quadratic coefficients is dense.

Key words: Interior point method, linear programming, quadratic programming.

1. Introduction

Since Karmarkar's fundamental paper [11] appeared in 1984, many interior point methods for linear programming have been proposed. These methods are all iterative algorithms that solve a system of the form

$$AD^2A^{\mathrm{T}}\Delta y = \gamma, \tag{1}$$

at each iteration. In all variants, D is a certain diagonal matrix (with strictly positive entries) that changes from one iteration to the next; γ is a known m-vector that also generally changes from one iteration to the next; and A is a fixed $m \times n$ constraint matrix that has full row rank.

The fact that the matrix in system (1) is positive definite and symmetric implies that one can choose the order of pivots at the beginning without considering numerical issues and then use this ordering in each subsequent iteration. This observation allows one to develop robust code that is also efficient because it does not have to reanalyze the matrix at every iteration.

While implementations based on this approach have proved to be quite efficient, difficulties arise if the constraint matrix A has dense columns or if one wants to generalize the algorithm to handle nonseparable quadratic programming problems.

Correspondence to: Prof. Robert J. Vanderbei, Statistics and Operations Research, Princeton University, Princeton, NJ 08544, USA.

In this paper we present a method to resolve these two difficulties by solving instead the following related system:

$$\begin{bmatrix} -D^{-2} & A^{\mathrm{T}} \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \sigma \\ \rho \end{bmatrix}. \tag{2}$$

Several variants of Karmarkar's algorithm have appeared in the literature, all of which can be described in terms of solutions to system (2). Table 1 shows how to choose D^2, σ and ρ to obtain the primal affine-scaling [5, 2, 27], dual affine-scaling [1, 17], one-phase primal-dual affine-scaling [14], and one-phase primal-dual path-following [13, 14, 24] methods. In this table, X denotes the diagonal matrix with the elements of x on the diagonal and Z denotes the diagonal matrix with z on the diagonal.

Table 1
Popular interior point methods

Name	D^2	σ	ρ
primal affine-scaling	X^2	c	0
dual affine-scaling	Z^{-2}	0	b
one-phase primal-dual affine-scaling	XZ^{-1}	$c - A^{\mathrm{T}}y$	$b - Ax$
one-phase primal-dual path-following	XZ^{-1}	$c - A^{\mathrm{T}}y - \mu X^{-1}e$	$b - Ax$

We focus on one variant — the one-phase primal-dual path-following algorithm (PDPF), which we implement in a code called LoQo (Linear or Quadratic Optimizer). LoQo's numerical "engine" is based on efficient routines for solving system (2), which explicitly take advantage of the inherent structure of the system and the fact that at each iteration only D^{-2}, ρ, and σ change. Our goal is to use LoQo to demonstrate the efficacy of the primal-dual interior point method based on system (2) for linear and quadratic programming.

We show that the strategy based on system (2) is actually just a generalization of the usual method involving AD^2A^{T} and provide computational comparisons with ALPO [24] — an analogous code for solving linear programs based on the AD^2A^{T} system. In most cases, LoQo automatically forms and factors the AD^2A^{T} system, so the two optimizers perform similarly. When A has dense columns, however, LoQo has the flexibility to delay handling dense columns, which yields significant computational savings.

In addition, LoQo does not explicitly require that the matrix D^{-2} negated in the upper left block in system (2) be diagonal; it must simply be positive definite and symmetric. This is precisely the structure of the system formed for convex quadratic programs. Thus, quadratic programs can also be addressed within this framework. We present computational results using LoQo to solve nonseparable quadratic programs and provide comparisons with MINOS [22] as a benchmark.

In the next section we give a brief description of the PDPF algorithm. In Section 3 we describe the ordering strategies used, and in Section 4 we give computational results for linear programs comparing LoQo to ALPO. Section 5 describes generalizing the PDPF algorithm to quadratic programming problems, with computational results included in Section 6. In Section 7 we describe separable formulations for nonseparable quadratic programs that yield significant computational savings for interior point methods. Extensions and future directions are discussed in the last section.

2. The primal–dual path-following algorithm

We are interested in solving the following *primal* linear programming problem:

$$\begin{aligned} \text{minimize} \quad & c^{\mathrm{T}}x \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{3}$$

where $A \in \mathbb{R}^{m \times n}$ has full row rank, $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$. Associated with this primal problem is the *dual* linear program:

$$\begin{aligned} \text{maximize} \quad & b^{\mathrm{T}}y \\ \text{subject to} \quad & A^{\mathrm{T}}y + z = c, \\ & z \geq 0, \end{aligned} \tag{4}$$

which we have written in equality form by introducing slack variables z (also called *reduced costs*).

The PDPF algorithm is motivated by applying the logarithmic barrier function to eliminate the inequality constraints in (3) and (4). The transformation of the primal yields the nonlinear program:

$$\begin{aligned} \text{minimize} \quad & c^{\mathrm{T}}x - \mu \sum_{j=1}^{n} \ln x_j \\ \text{subject to} \quad & Ax = b. \end{aligned} \tag{5}$$

Likewise, the barrier transformation of the dual can also be obtained.

If we fix $\mu > 0$ and apply the barrier transformation to the primal and dual linear programs at a point that satisfies $x > 0$ and $z > 0$, the requisite first order conditions for simultaneous optimality in the primal and dual barrier problems are:

$$Ax = b, \tag{6}$$

$$A^{\mathrm{T}}y + z = c, \tag{7}$$

$$XZe = \mu e, \tag{8}$$

where e denotes the n-vector of all ones. The first m equations (6) form part of the primal feasibility requirement, and the next n equations (7) form part of the dual feasibility requirement. The last n equations are called *μ-complementarity.*

Monteiro and Adler [20] provide a full theoretical development of the primal-dual path-following algorithm, including results which imply the following proposition:

Proposition 1. *If either the primal or the dual polytope is bounded and has a nonempty interior, then there exists a unique solution to* (6)-(8) *in the domain given by* $x \geq 0$, $z \geq 0$. □

Let (x_μ, y_μ, z_μ) be the solution to (6)-(8) provided by Proposition 1. The map $\mu \mapsto (x_\mu, y_\mu, z_\mu)$ is called the *central path.* As μ tends to zero, this path (x_μ, y_μ, z_μ) converges to (x^*, y^*, z^*) where x^* is optimal for the primal linear program, and (y^*, z^*) is optimal for its dual. (Assuming that the primal polytope is bounded and has nonempty interior, x_μ tends towards the "center" of the primal polytope and (y_μ, z_μ) tends toward infinity in the necessarily unbounded dual as μ tends to infinity.)

2.1. The algorithm

The primal-dual path-following algorithm can now be described quite simply. We start with any triple (x, y, z) satisfying $x > 0$ and $z > 0$ and with any $\mu > 0$. We next use one step of Newton's method to try to find a point closer to (x_μ, y_μ, z_μ). We then let this new point be our current (x, y, z), reduce μ appropriately, and start over. This iterative process is continued until primal and dual feasibility is attained and the duality gap is smaller than some predetermined tolerance.

Applying Newton's method to (6)-(8) yields

$$A\Delta x = \rho, \tag{9}$$

$$A^{\mathrm{T}}\Delta y + \Delta z = \sigma, \tag{10}$$

$$Z\Delta x + X\Delta z = \phi, \tag{11}$$

where

$$\rho = b - Ax, \qquad \sigma = c - A^{\mathrm{T}}y - z, \qquad \phi = \mu e - XZe.$$

Writing this in block matrix form, we get

$$\begin{bmatrix} X & Z & 0 \\ I & 0 & A^{\mathrm{T}} \\ 0 & A & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \phi \\ \sigma \\ \rho \end{bmatrix}. \tag{12}$$

The desired update is then

$$x \leftarrow x + \alpha_{\mathrm{p}}\Delta x, \qquad y \leftarrow y + \alpha_{\mathrm{d}}\Delta y, \qquad z \leftarrow z + \alpha_{\mathrm{d}}\Delta z, \tag{13}$$

where $0 < \alpha_{\mathrm{p}}, \alpha_{\mathrm{d}} \leq 1$ are chosen to keep $x > 0$ and $z > 0$. The steplengths α_{p} and α_d are chosen independently for the primal and the dual, and their inclusion makes the overall method a *damped* Newton method.

2.2. Alternative systems

The system of equations (12) is usually not attacked directly, but is first simplified algebraically as follows. First, we use the top set of equations to solve for Δz in terms of Δx, and then eliminate Δz from the system:

$$\Delta z = X^{-1}(\phi - Z\Delta x), \tag{14}$$

and

$$\begin{bmatrix} -ZX^{-1} & A^{\mathrm{T}} \\ A & 0 \end{bmatrix}\begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \sigma - X^{-1}\phi \\ \rho \end{bmatrix}. \tag{15}$$

Note that system (15) is symmetric whereas the original system (12) is not. Also, the reduction from (12) to (15) entails no off-diagonal fill-in in the system of equations. Hence, there is no reason to prefer system (12) to system (15).

The second reduction involves solving explicitly for Δx in terms of Δy using the first set of equations in (15) and then eliminating Δx from the system:

$$\Delta x = -XZ^{-1}(\sigma - X^{-1}\phi - A^{\mathrm{T}}\Delta y), \tag{16}$$

and

$$[AXZ^{-1}A^{\mathrm{T}}][\Delta y] = [\rho + AXZ^{-1}(\sigma - X^{-1}\phi)]. \tag{17}$$

The advantage of system (17) over (15) is that it is based on a symmetric positive definite matrix guaranteed to yield a factorization of the form $L\Lambda L^{\mathrm{T}}$, where L is lower triangular with unit diagonal and Λ is diagonal with strictly positive entries. Given the $L\Lambda L^{\mathrm{T}}$ factorization, Δy is easily obtained from (17) by solving two triangular systems. However, the disadvantage of system (17) is that it suffers a great deal of fill-in when $AXZ^{-1}A^{\mathrm{T}}$ is formed. This is especially true if A has one or more dense columns. Nonetheless, most implementations of interior point methods are based on system (17).

3. The symmetric indefinite system

Gill, Murray, Ponceleón and Saunders [9] and Fourer and Mehrotra [6] advocate solving the indefinite system (15) to prevent fill-in. Their approaches are based on performing a Bunch-Parlett [3] factorization of the indefinite matrix. When $D^2 = XZ^{-1}$ and

$$K = \begin{bmatrix} -D^{-2} & A^{\mathrm{T}} \\ A & 0 \end{bmatrix},$$

this factorization has the form $PKP^{\mathrm{T}} = LBL^{\mathrm{T}}$, where P is a permutation matrix and B is a symmetric block-diagonal matrix, with 1×1 or 2×2 blocks. Such a factorization is guaranteed to exist [10], but unlike the positive definite case, the pivot order cannot be computed symbolically because the permutation choice is based on numerical values. Thus, the major disadvantage of this approach is that the pivot

order cannot be fixed once in the analyze phase, but must be recomputed as X and Z are updated in each iteration.

Sometimes, however, a factorization $L\Lambda L^{\mathrm{T}}$ may also exist for indefinite matrices. When it does, Λ is no longer positive but remains diagonal and nonsingular. The factors $L\Lambda L^{\mathrm{T}}$ can be obtained by judiciously picking symmetric pivot rules while performing symmetric elimination. For example, suppose we stipulate that the first n columns be eliminated before the last m. Each of the first n pivots has the corresponding entry in $-D^{-2}$ as a pivot element when symmetric elimination is performed on K. Since D is positive, no zero pivots are encountered, and after the nth stage of symmetric elimination the partially reduced matrix has the form

$$K_n = \begin{bmatrix} -D^{-2} & 0 \\ 0 & M \end{bmatrix}.$$

Each stage of the elimination performs a rank-one update of the lower right $m \times m$ submatrix, and after n stages, AD^2A^{T} is explicitly formed. That is, the lower right $m \times m$ submatrix M is sequentially formed as

$$M = \sum_{j=1}^{n} d_j^2 a_j a_j^{\mathrm{T}} = AD^2A^{\mathrm{T}}.$$

At this point the remaining matrix is positive definite and the factorization $L_M D_M L_M^{\mathrm{T}}$ of M is guaranteed to exist. Moreover, the last m pivots can be performed in any order, using whatever efficient pivoting strategy is preferred. (For example, the minimum degree or the minimum local fill ordering heuristics might be used). When the symmetric elimination is complete, we have $K = L_K D_K L_K^{\mathrm{T}}$ where D_K is an $(n+m)\times(n+m)$ diagonal matrix of the form

$$D_K = \begin{bmatrix} -D^{-2} & 0 \\ 0 & D_M \end{bmatrix},$$

and

$$L_K = \begin{bmatrix} I & 0 \\ -AD^2 & L_M \end{bmatrix}.$$

Hence, the old method is imbedded in the new method as one particular pivot strategy. We call this pivot strategy the *conservative pivot strategy*. Solving (15) and solving (17) are roughly equivalent under the conservative strategy. The new method combines the formation and factorization of AD^2A^{T} into one process — simply factoring K. Also, the number of arithmetic operations required to implement this specific pivot rule is essentially equal to the number of arithmetic operations required to form and then factor AD^2A^{T}. Hence, applying this pivot strategy to system (15) should generally be about as efficient as working with system (17).

To guarantee that a factorization of the form $L\Lambda L^{\mathrm{T}}$ exists, it is not necessary to force all of the first n columns to be eliminated first as is done in the conservative strategy. All that is required is that enough columns from the first n are selected so that the corresponding columns of A span the entire column space of A. After a

spanning set of columns is eliminated in symmetric Gaussian elimination, the lower right submatrix of the partially reduced matrix is a *quasi-definite* matrix. We say that a symmetric indefinite matrix

$$\begin{bmatrix} -E & R^T \\ R & F \end{bmatrix}$$

is quasi-definite if both E and F are positive definite. Symmetric quasi-definite matrices form a class of symmetric indefinite matrices for which the factorization LAL^T always exists. But more importantly, Proposition 2, provided by Vanderbei [26], assures that all symmetric permutations of a quasi-definite matrix are factorable.

Proposition 2. *For every permutation matrix P, there exists a factorization such that*

$$P\begin{bmatrix} -E & R^T \\ R & F \end{bmatrix}P^T = LAL^T,$$

where L is lower triangular with unit diagonal and A is nonsingular and diagonal. □

Since the LAL^T factorization is guaranteed to exist for any symmetric reordering, a quasi-definite matrix can be symbolically analyzed to determine a pivot order. This property makes reducing our indefinite system (15) to a quasi-definite system as appealing as reducing it to a positive definite system. If we can determine a set of columns of rank m, we can reduce (15) to a quasi-definite system. To see this, partition A so that

$$A = [B|N]$$

where B has $k \geq m$ columns and $\operatorname{rank}(B) = m$ and then write K accordingly:

$$K = \left[\begin{array}{cc|c} -D_B^{-2} & & B^T \\ & -D_N^{-2} & N^T \\ \hline B & N & \end{array}\right].$$

Pivoting on the first k diagonal elements now yields a partially reduced matrix whose lower right submatrix is quasi-definite:

$$\left[\begin{array}{c|cc} -D_B^{-2} & & \\ \hline & -D_N^{-2} & N^T \\ & N & BD_B^2B^T \end{array}\right].$$

Choosing B remains an issue, but certainly letting B contain all of the columns of A is guaranteed to deliver a rank m set of columns. This choice is exactly the conservative pivot strategy. Another instance in which a spanning set of columns is readily available is when all constraints are inequality. In the inequality constrained case, the columns of slack (or surplus) variables appended to transform all constraints to equalities provide an immediate basis. This case is discussed further in Section 5.1.

Most of the time, however, we advocate using the conservative strategy. One exception is when A has dense columns. When the jth column of A is completely

dense, $a_j a_j^T$ is a fully dense $m \times m$ matrix. Consequently, AD^2A^T is also fully dense. This effect is illustrated in Figure 1. If all of the first n columns of K (including the dense columns) are pivoted out first, the matrix AD^2A^T, formed in the lower right hand corner, will be dense. So, requiring that dense columns be eliminated as part of the first n pivots can dramatically impair the efficiency of solving system (15).

If it is true that the *nondense* columns of A also span the column space of A, then fill-in can be reduced by grouping the dense columns with the last m pivots. This strategy forms a quasi-definite matrix in the lower right submatrix. Moreover, when the last group of pivots are arranged according to some efficient ordering scheme such as minimum degree or minimum local fill-in, the dense columns will be selected last even among this second group. Figure 2 illustrates that by eliminating the dense column last among the second group, no fill-in occurs from eliminating the dense column. This technique for delaying the elimination of dense columns is algebraically equivalent to the well-known Schur-complement technique for handling dense columns.

So, the method we propose for solving (15) consists, as usual, of an analyze phase which determines the order of the pivots and sets up data structures, and a numerical factorization phase in which the symmetrically reordered version of K is factored. Moreover, we suggest a very limited mixing *between* the two groups of columns during reordering. There is mixing only when dense columns are present. A recent implementation by Fourer and Mehrotra [6] solves a system equivalent to (15) but allows a full reordering of the matrix.

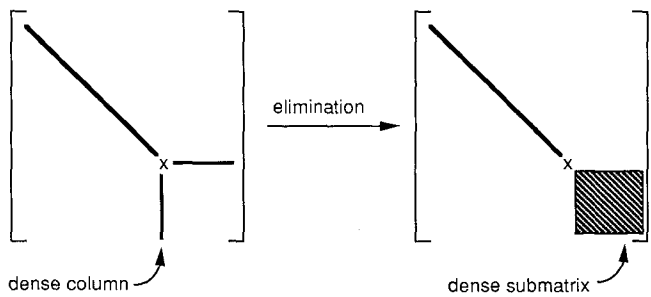

Fig. 1. Eliminating dense columns — conservative strategy.

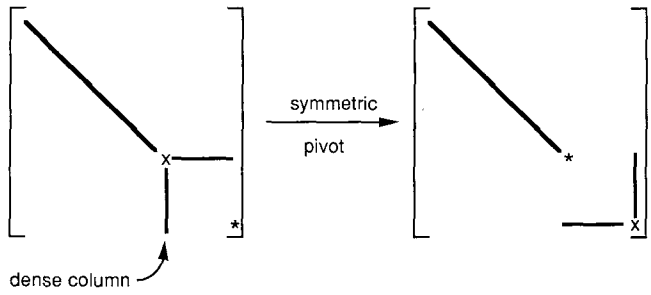

Fig. 2. Pivoting to reduce fill-in.

4. Computational testing—Linear programs

Herein, we compare LoQo which solves system (15) to ALPO — a similar code based on solving the usual AD^2A^T system. All testing is conducted on an IBM RISC System/6000. Both codes are written in C and compiled with AIX version 3.1 C compiler using the default optimization. We use the full NETLIB [7] test set in MPS standard format.

The requisite optimality criteria are the same for both solvers. Namely, a relative measure of the primal and dual infeasibility must be less than 1.0×10^{-6}, and the relative duality gap must be less than 1.0×10^{-8}. The relative primal and dual infeasibilities are measured by

$$\frac{\|Ax - b\|_2}{\|b\|_2 + 1},$$

and

$$\frac{\|c - A^T y - z\|_2}{\|c\|_2 + 1},$$

respectively, while the relative duality gap is

$$\frac{c^T x - b^T y}{|c^T x| + 1}.$$

ALPO and LoQo use the same heuristics for determining the starting point, the barrier parameter μ, and the steplengths α_p and α_d. Both solvers use a minimum degree ordering heuristic to preserve sparsity during their respective factorizations. LoQo does not, however, fully reorder the entire matrix K. When no dense columns are present, it orders the first n pivots and the last m pivots within their respective groups. Alternatively, when LoQo's internal heuristic detects dense columns, it can elect to move dense columns from the first group into the second before reordering. Likewise, ALPO may *split* dense columns or elect to solve the dual as the primal when dense columns are present (see [25] for details).

Table 2 provides a computational comparison between LoQo and ALPO on the NETLIB test set. The main points of comparison are the number of arithmetic operations needed to perform the required factorization and the total solve time obtained using the time command from the UNIX command line. For problems that are not solved, the maximum number of digits of agreement between the primal and dual objectives is given in parentheses in place of the solve time.

The *dense column threshold* given in Table 2 indicates the level at which LoQo declares a column of K to be dense. The heuristic employed to detect dense columns is simple and conservative. Given K, it first selects the m sparsest columns from the first n and places them in the first pivot group. Then, it declares dense any column from the first n that has more than ten times the number of nonzeros as the densest of these first m columns. Dense columns are placed in the second group

Table 2
Statistics for netlib problems

Problem name	Arithmetic operations		Solution time		Iterations		Maximum column density	Dense column threshold
	AD^2A^T	K	AD^2A^T	K	AD^2A^T	K		
25fv47	2760211	2780905	23.99	25.80	40	40	21	
80bau3b	2945058	2482245	83.32	98.58	95	95	12	
adlittle	6776	7338	0.38	0.42	26	26	11	
afiro	1003	1156	0.14	0.18	13	13	4	
agg	230528	506107	4.47	8.99	60	65	43	30
agg2	1193785	870064	10.99	8.42	38	37	43	30
agg3	1127453	857144	11.45	8.85	41	40	43	30
bandm	131780	134115	2.34	2.37	29	29	22	
beaconfd	192992	152646	2.50	1.86	21	20	27	
blend	22846	23482	0.38	0.47	19	19	16	
bnl1	563765	581259	13.92	16.52	83	83	8	
bnl2	13690320	13358314	125.55	128.37	57	57	8	
boeing1	265999	226677	4.70	(6)	38	*	32	30
boeing 2	80846	81357	1.44	1.61	34	35	23	
bore3d	93047	91175	1.87	1.91	38	38	28	
brandy	162037	145041	2.26	2.00	30	30	29	
capri	259466	244917	3.45	3.54	41	41	25	
cycle	6139656	9344502	(7)	106.81	*	60	28	
czprob	167559	181829	14.63	20.21	78	78	4	
d2q06c	26344173	33611638	249.75	311.53	58	57	34	30
degen2	970554	1016637	6.59	7.38	26	27	22	
degen3	19014469	16184767	181.44	127.90	53	42	49	
e226	124091	128163	2.41	2.34	33	32	21	
etamacro	1079849	1158886	8.94	10.19	41	41	8	
fffff800	1097108	1058556	15.20	14.60	51	53	50	
finnis	162763	163461	3.58	4.20	39	39	14	
fit1d	210077	224553	6.57	7.28	26	26	18	
fit1p	588603	148220	15.50	5.83	27	25	610	20
fit2d	1891633	2031199	72.78	217.49	33	33	17	
fit2p	4442053	649721	227.28	47.46	32	28	2971	20
forplan	231300	218645	4.50	4.20	42	42	35	
ganges	1577518	1753923	12.64	14.10	30	30	13	
gfrdpnc	20385	23916	2.17	2.67	27	27	3	
greenbea	5834350	6149368	(3)	(5)	*	*	24	
greenbeb	5834350	6149368	225.10	257.78	175	177	24	
grow15	244210	235265	4.18	3.74	25	25	20	
grow22	361096	347986	(8)	11.25	*	68	20	
grow7	110626	106441	1.87	1.60	23	22	20	
isreal	719719	105436	6.14	2.80	38	40	136	20
kb2	10608	10989	0.35	0.42	28	28	14	
lotfi	41241	43187	1.30	1.30	31	31	10	
maros	1708722	1677094	19.15	19.63	44	44	20	
nesm	1597438	1980353	41.35	49.10	95	95	10	
perold	1983413	2416657	29.40	35.75	74	74	16	
pilot4	1168188	1078103	16.94	16.20	64	63	27	
pilot87	211406218	232401792	2375.41	2362.62	79	73	96	30
piloja	6812674	6091671	85.70	75.80	68	66	55	40
pilotnov	5866830	5821645	50.60	50.62	45	45	40	

Table 2—continued

Problem name	Arithmetic operations		Solution time		Iterations		Maximum column density	Dense column threshold
	AD^2A^T	K	AD^2A^T	K	AD^2A^T	K		
pilots	53044741	45723391	519.56	428.70	63	59	121	30
pilotwe	1010753	1017334	39.11	46.26	145	145	12	
recipe	19096	19987	0.49	0.61	17	17	10	
sc105	5455	6228	0.36	0.43	21	21	5	
sc205	11974	12594	0.62	0.76	23	23	5	
sc50a	2196	2292	0.20	0.25	18	18	5	
sc50b	2138	2276	0.20	0.25	16	16	4	
scargr25	34868	37250	1.76	2.17	33	33	9	
scagr7	8718	9414	0.56	0.63	29	29	9	
scfxm1	132268	134388	2.78	2.76	36	35	20	
scfxm2	276635	276338	5.98	6.25	40	39	20	
scfxm3	421000	417348	9.11	9.70	40	40	20	
scorpion	33676	35210	1.32	1.64	27	27	7	
scrs8	184874	190685	3.65	4.55	35	35	8	
scsd1	43153	46301	1.19	1.48	17	17	4	
scsd6	73613	79279	2.32	2.81	19	19	4	
scsd8	142703	154037	4.30	5.41	18	18	4	
sctap1	45902	47158	1.76	2.21	38	38	6	
sctap2	631744	671208	6.83	8.10	24	24	6	
sctap3	677510	689712	8.90	10.97	25	25	6	
seba	3009683	67625	24.48	4.38	44	45	230	20
share1b	28678	35300	1.16	1.38	45	45	10	
share2b	21849	22466	0.52	0.62	21	21	12	
shell	86456	95187	3.84	4.83	35	35	3	
ship04l	104554	113100	4.80	5.80	25	25	6	
ship04s	74998	80904	2.77	3.45	23	23	6	
ship08l	212626	230111	9.45	12.45	34	34	6	
ship08s	124698	134111	5.20	6.64	30	30	6	
ship12l	263961	285946	11.4	14.52	29	29	6	
ship12s	137593	152618	5.85	7.57	28	28	6	
sierra	355122	366160	7.50	9.22	28	28	4	
stair	1292003	1077366	8.80	7.47	30	29	34	
standata	73155	77659	2.50	3.35	30	30	10	
standmps	136351	140057	3.64	4.54	36	36	10	
stocfor1	13458	13884	0.40	0.53	19	19	6	
stocfor2	612126	622502	14.54	17.30	45	45	10	
tuff	427430	431047	6.28	6.55	45	45	25	
vtpbase	66735	68132	1.87	2.7	54	54	12	
wood1p	5197822	3738081	82.97	57.97	28	30	28	
woodw	3582687	3595218	48.4	57.27	44	41	21	

while the remaining columns are added to the first group. This strategy guarantees that there are at least m columns in the first group, and often it places all n columns in the first group to yield the basic conservative strategy.

In most cases, the timings of the two solvers are comparable although ALPO is faster on 62 of the 84 problems that are solved by both. Usually, widely differing performance is the result of dense column treatment, and LoQo tends to do better

on these problems. Below we summarize observations on the solution of problems which yield wide performance differences between the two solvers.

- On problem agg, ALPO chooses to solve the dual problem, which clearly gives it an edge over LoQo even though LoQo does declare some columns as dense.
- On problems agg2 and agg3, ALPO elects to solve the primal problem, and the fact that LoQo declares some columns as dense gives LoQo the edge on these problems.
- On problems fit1p and fit2p, ALPO gains a substantial advantage by solving the dual problem, but it is not enough to match the advantage LoQo gets by declaring all columns with 20 or more nonzeros as dense.
- On problem israel, LoQo declares all columns with 20 or more nonzeros as dense. ALPO chooses to solve the primal problem (since the problem has dense rows as well as dense columns) and only gets a slight improvement from splitting columns with 20 or more nonzeros.
- On problem seba, ALPO solves the primal (since as currently implemented, it can't solve the dual for problems with ranges) and gains some advantage by splitting colums with 20 or more nonzeros. LoQo, however, gains a much bigger advantage by declaring as dense all columns with 20 or more nonzeros.
- On problem wood1p, 2556 of the 2595 columns have 20 or more nonzeros and so the problem that ALPO solves after splitting is much larger than the original problem. On this problem, it would have been better if ALPO had decided not to split columns. ALPO solves wood1p in 62.39 seconds when splitting is turned off, which is still slightly slower than the LoQo time.
- LoQo fails to meet the termination criteria of boeing1 and greenbea. boeing1 attains both primal and dual feasibility with 6 digits of agreement between the primal and dual objective values. greenbea, however, is never primal feasible.
- ALPO fails to satisfy the termination criteria of cycle, greenbea, and grow22. Although grow22 gets the necessary agreement in primal and dual objective values, it nevers satisfies the primal feasibility requirement.

5. Quadratic programs

We now consider the role of the symmetric indefinite system in quadratic programming. The standard form quadratic program is

$$\begin{aligned} \text{minimize} \quad & c^{\mathrm{T}}x + \tfrac{1}{2}x^{\mathrm{T}}Qx \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0, \end{aligned} \tag{18}$$

where Q is an $n \times n$ positive semidefinite matrix. It has the associated dual quadratic program

$$\begin{aligned} &\text{maximize} && b^{\mathrm{T}}y - \tfrac{1}{2}x^{\mathrm{T}}Qx \\ &\text{subject to} && A^{\mathrm{T}}y + z - Qx = c, \\ & && z \geqslant 0. \end{aligned} \tag{19}$$

Once again, we can apply the logarithmic barrier function to eliminate the inequality constraints in (18) and (19). Simultaneous optimality in the barrier transformed problems requires

$$Ax = b, \qquad A^{\mathrm{T}}y + z - Qx = c, \qquad XZe = \mu e. \tag{20}$$

We can now apply Newton's method to the requisite conditions (20) to obtain a step direction. This is analogous to the linear case in Section 2 and is described in Monteiro and Adler [21]. The step direction for the quadratic program is the solution $(\Delta x, \Delta y, \Delta z)$ of the system

$$\begin{bmatrix} X & Z & 0 \\ I & -Q & A^{\mathrm{T}} \\ 0 & A & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \phi \\ \sigma_{\mathrm{q}} \\ \rho \end{bmatrix}, \tag{21}$$

where ρ, σ_{q}, and ϕ are respectively the primal infeasibility, dual infeasibility, and noncomplementarity at the current point. The previous definitions for ρ and ϕ still apply, but the dual infeasibility now also depends on the primal variables and is defined as

$$\sigma_{\mathrm{q}} = c + Qx - A^{\mathrm{T}}y - z.$$

Having obtained Δx, Δy, and Δz, the new estimate is

$$x \leftarrow x + \alpha \Delta x, \qquad y \leftarrow y + \alpha \Delta y, \qquad z \leftarrow z + \alpha \Delta z, \tag{22}$$

where α is a steplength chosen to ensure that x and z are sufficiently nonnegative.

As in the linear case, we can use the first set of equations in (21) to eliminate Δz and get

$$\begin{bmatrix} -(Q + ZX^{-1}) & A^{\mathrm{T}} \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = \begin{bmatrix} \sigma_{\mathrm{q}} - X^{-1}\phi \\ \rho \end{bmatrix}. \tag{23}$$

Let K_{q} denote the matrix in system (23). This matrix differs from K considered previously in that $(Q + ZX^{-1})$ appears in the upper left where the diagonal matrix ZX^{-1} had been formerly. Since Q is symmetric, K_{q} is also symmetric and indefinite. Once again, it is valid to eliminate Δx from (23) using the first set of equations and solve

$$A\Theta^{-1}A^{\mathrm{T}}\Delta y = \rho + A\Theta^{-1}(\sigma_{\mathrm{q}} - X^{-1}\phi) \tag{24}$$

for Δy, where $\Theta = (Q + ZX^{-1})$. The matrix in the left-hand side of system (24) is positive definite, so its $L\Lambda L^T$ factorization can be computed. Unfortunately, this requires computing Θ^{-1} in the process. Since Q is at least positive semidefinite and we add the positive diagonal matrix ZX^{-1} this matrix is clearly nonsingular. In fact, when Q is diagonal, the analysis of Section 3 applies directly. We simply let D^{-2} in K be the diagonal matrix $(Q + ZX^{-1})$ to form K_q. Since the upper left matrix is positive and diagonal, elimination in K_q is analogous to elimination in K, discussed for the linear case.

When Q is not diagonal, however, Θ^{-1} may be difficult to compute and fully dense. In addition, the fill-in that occurs in (24) results not only from the structure of A, but is exacerbated by the appearance of Θ^{-1}. Alternatively, we can factor Θ so that $\Theta = L_\Theta \Lambda_\Theta L_\Theta^T$, and system (24) becomes

$$AL_\Theta^{-T}\Lambda_\Theta^{-1}L_\Theta^{-1}A^T\Delta y = \rho + AL_\Theta^{-T}\Lambda_\Theta^{-1}L_\Theta^{-1}(\sigma_q - X^{-1}\phi). \tag{25}$$

This is effectively equivalent to applying the conservative strategy to solve (23).

In the remainder of this section, we explore symmetric elimination in K_q when Q is not necessarily diagonal. Since Q is symmetric, K_q is symmetric, but a factorization of the form $K_q = L\Lambda L^T$ is not guaranteed to exist because it is indefinite. Still, we can fix an ordering to assure that this factorization is computable. Once again, we adopt a conservative pivot strategy in which we require that the first n columns be eliminated first. The positive definiteness of Θ in the matrix

$$K_q = \begin{bmatrix} -\Theta & A^T \\ A & 0 \end{bmatrix}$$

assures that no zero diagonal elements are encountered in the first n eliminations. Therefore, after n stages of elimination, the partially reduced matrix has the form

$$\begin{bmatrix} -\Lambda_\Theta & 0 \\ 0 & M \end{bmatrix}. \tag{26}$$

Analogous to the linear case, the matrix $M = AL_\Theta^{-T}\Lambda_\Theta^{-1}L_\Theta^{-1}A^T$ — the matrix which appears in (25). The positive definiteness of Θ (and hence its factors) guarantees the positive definiteness of $AL_\Theta^{-T}\Lambda_\Theta^{-1}L_\Theta^{-1}A^T$ when A has full row rank and assures that the remainder of K_q can now be eliminated in the usual way to yield a factorization $K_q = L\Lambda L^T$. Further, the work required to obtain the factors of K_q is essentially equivalent to the work required to first factor Θ and then form and factor $AL_\Theta^{-T}\Lambda_\Theta^{-1}L_\Theta^{-1}A^T$ in order to solve for the Newton direction via the smaller system (24).

Fill-in of the lower $m \times m$ submatrix formed during elimination occurs as a result of either nonzeros in A or nonzeros in L_Θ. A dense column and row (by symmetry) in Q yields a dense row in L_Θ and a dense column in L_Θ^T — even when the factorization is performed to minimize fill. Likewise, L_Θ^{-1} and L_Θ^{-T} have a dense row and column, respectively. Whenever L_Θ^{-T} has a dense column, the product AL_Θ^{-T} has a dense column *regardless* of the structure of A. Dense columns in AL_Θ^{-T} therefore occur when there are dense columns in either A or Q. Indeed, if there are few

quadratic variables with many cross terms, the effect should be very similar to when there are dense columns in A. Once again, it may be beneficial to group particularly dense columns in K_q with the last m columns during elimination to avoid creating fill. This is possible as long as the nondense portion of AL_Θ^{-T} has rank m. By doing this, pivot rules are exploited throughout the solve (not just in the last m steps) to maintain sparsity. But as in the linear case, the reordering is mainly within the two groups but allows dense columns to be placed in the second group. This is in contrast to the method proposed by Ponceleón [23] which advocates a full reorder of the indefinite system and requires a specialized procedure for factoring indefinite matrices.

5.1. *The inequality constrained quadratic program*

One case in which solving the indefinite system is both robust and efficient is the *inequality* constrained quadratic program:

$$\begin{aligned} &\text{minimize} && c^T x + \tfrac{1}{2} x^T Q x \\ &\text{subject to} && Ax \leq b, \\ & && x \geq 0. \end{aligned} \tag{27}$$

The quadratic program (27) can always be written in standard form (18) by adding slack variables. We get

$$\begin{aligned} &\text{minimize} && c^T x + \tfrac{1}{2} x^T Q x \\ &\text{subject to} && Ax + s = b, \\ & && x, s \geq 0, \end{aligned} \tag{28}$$

where $s \in \mathbb{R}^m$. The primal quadratic program has $n+m$ variables and the dual has $n+m$ corresponding constraints:

$$A^T y + z_x - Qx = c, \tag{29}$$

$$y + z_s = 0. \tag{30}$$

We partition the dual slack variables so that $z = [z_x \mid z_s]$ where z_x is the first n and z_s the last m slack variables. The Newton equations based on (28) are now

$$\begin{bmatrix} X & & Z_x & & \\ & S & & Z_s & \\ I & & -Q & & A^T \\ & I & & & I \\ & & A & I & \end{bmatrix} \begin{bmatrix} \Delta z_x \\ \Delta z_s \\ \Delta x \\ \Delta s \\ \Delta y \end{bmatrix} = r. \tag{31}$$

To simplify exposition, we allow r to be a known, conformable right-hand side vector throughout the remainder of this section.

Now we can proceed in the usual way by using the first $m+n$ equations to eliminate Δz_x and Δz_s and obtain

$$\begin{bmatrix} -(Q+Z_xX^{-1}) & & A^T \\ & -S^{-1}Z_s & I \\ A & I & \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta s \\ \Delta y \end{bmatrix} = r. \tag{32}$$

System (32) corresponds to system (23) for the original standard form quadratic program where the constraint matrix is $\bar{A}=[A\,|\,I]$. The columns in $\bar{A}$ associated with the slack variables are of rank m. Thus, if we order to eliminate in these columns first, Proposition 2 assures that the remaining columns can be eliminated in *any order* to best preserve sparsity. Because each of the slack variable columns has only one off-diagonal nonzero, LoQo *automatically* places them among the first to be eliminated. Consequently, the ordering selected by LoQo yields a factorization of the form $L\Lambda L^T$ whenever the constraints are in inequality form. This factorization is also guaranteed for inequality constrained linear programs, but we obtain an extra bonus in the quadratic case.

Having eliminated the Δs variables in the usual way, we have

$$\begin{bmatrix} -(Q+Z_xX^{-1}) & A^T \\ A & SZ_s^{-1} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = r. \tag{33}$$

The matrix in (33) is quasi-definite and can be symmetrically permuted to minimize fill-in. If we prohibit mixing between the two blocks, however, there are two ways to proceed. We can use the top set of equations to solve for Δx in terms of Δy and solve the system

$$[A(Q+Z_xX^{-1})^{-1}A^T+SZ_s^{-1}]\Delta y = r \tag{34}$$

for Δy in the usual way. Alternatively, we can use the *bottom* set of equations to solve for Δy in terms of Δx and then solve

$$-[(Q+Z_xX^{-1})+A^TS^{-1}Z_sA]\Delta x = r \tag{35}$$

for Δx. This second approach requires factoring one $n\times n$ matrix of known sparsity pattern. The first approach entails first factoring the $n\times n$ matrix $(Q+Z_xX^{-1})$ and then forming and factoring the $m\times m$ matrix that appears in (34). Solving the system (35) offers a clear advantage — especially when Q is dense. If Q is dense, no fill-in occurs from reducing (33) to (35), whereas the reduction to (34) does entail fill-in. The dense column heuristic in LoQo will likely delay the columns involving a dense Q until last, thereby executing the preferred strategy which forms (35).

5.2. A note on stability

So far, we have not explicitly considered the issue of stability beyond saying that, when the $L\Lambda L^T$ factorization is guaranteed to exist, our method should be about as stable as the method based on the $A\Theta^{-1}A^T$ system. In the remainder of this section, we attempt to formalize our consideration of stability. Before proceeding,

however, we note that the stability of interior point methods (even those based on the $A\Theta^{-1}A^T$ system) needs to be better understood.

The overall stability of the method based on the symmetric indefinite system depends on two issues. The first is whether or not the symmetric reordering places a set of columns of rank m in the first group of columns to be eliminated. If it does, then eliminating these columns yields a reduced system whose matrix is symmetric quasi-definite and guarantees that an $L\Lambda L^T$ factorization exists for this ordering.

Given an ordering for which the $L\Lambda L^T$ factorization exists, whether or not it can be computed in a numerically stable fashion is the second issue. In [26], Vanderbei discusses the stability of factoring the matrix that appears in (33) when Q is zero. This matrix becomes increasingly poorly conditioned as the method progresses. On the last iteration, the ratio of its largest to smallest diagonal elements (in absolute value) ranges from 1.0e+18 to 1.0e+57, as given in [26]. In this light, it is somewhat surprising that LoQo solves almost all of the NETLIB problems. Furthermore, those problems that it does not solve do not necessarily have the largest ratios. Vanderbei [26] notes that for many of the NETLIB problems the matrix in (33) is actually converging to a singular matrix. When this is the case, *any* symmetric reordering is unstable — including the conservative strategy which reduces to the $A\Theta^{-1}A^T$ system.

6. Computationl testing — Quadratic programs

The general scheme of solving the indefinite system (23) for quadratic programs was suggested by Ponceleón [23] but was not computationally tested for nonseparable problems. Herein, we explore the efficacy of this approach for solving nonseparable problems.

6.1. Test problem description

We are not currently aware of a publicly available set of positive semidefinite quadratic programs for testing. Therefore, we employ the NETLIB [7] set of linear programs to generate quadratic test cases. We create quadratic objective terms as a product based on a subset of the rows of A. We let A_r denote a 0-1 mask of the first r% of the rows of A; an element of A_r is 1 whenever the corresponding element of A is nonzero and 0 otherwise. We set $Q = A_r^T A_r$ so that the quadratic test problems are simply the NETLIB problems with this additional quadratic term. The current set of test problems is formed with $r = 5\%$. So formed, Q is typically of low rank and may be quite dense.

Our test set consists of 74 of the 89 total NETLIB problems. We omit any problem with: more than 2000 rows; more than 10,000 columns; or more than 500,000 nonzeros in the Q matrix. Thus, 80bau3b, bnl2, d2q06c, dfl001, fit2d, fit2p, greenbea, greenbeb, pilot87, and stocfor2 are removed based on the size of the constraint matrix. Czprob, fit1d, and woodlp are omitted because of the density of Q.

Table 3
Test problem characteristics

Problem	Rows	Columns	Nonzeros in Q	Problem	Rows	Columns	Nonzeros in Q
25fv47	821	1571	116888	pilotnov	975	2172	873
adlittle	56	97	146	pilots	1441	3652	14456
afiro	27	32	9	pilotwe	722	2789	6367
agg	488	163	102	recipe	91	180	64
agg2	516	302	511	sc105	105	103	16
agg3	516	302	511	sc205	205	203	31
bandm	305	472	57	sc50a	50	48	9
beaconfd	173	262	32	sc50b	50	48	9
blend	74	83	22	scagr25	471	500	228
bnl1	643	1175	1990	scagr7	129	140	42
boeing1	351	384	123654	scfxm1	330	457	1409
boeing2	166	143	10705	scfxm2	660	914	2188
bore3d	233	315	127	scfxm3	990	1371	2353
brandy	220	249	114	scorpion	388	358	58
capri	271	353	1727	scrs8	490	1169	209
cycle	1903	2857	1915	scsd1	77	760	1292
degen2	444	534	18275	scsd6	147	1350	2712
degen3	1503	1818	120164	scsd8	397	2750	4740
e226	223	282	1861	sctap1	300	480	270
etamacro	400	688	8516	sctap2	1090	1880	1413
fffff800	524	854	3554	sctap3	1480	2480	1908
finnis	497	614	3807	seba	515	1028	1156
fit1p	627	1677	3176	share1b	117	225	44
forplan	161	421	1128	share2b	96	79	81
ganges	1309	1681	1272	shell	536	1775	69168
gfrdpnc	616	1092	267	ship04l	402	2118	98
grow15	300	645	962	ship04s	402	1458	98
grow22	440	946	1639	ship08s	778	2387	21216
grow7	140	301	684	ship12s	1151	2763	33280
israel	174	142	1354	sierra	1227	2036	244
kb2	43	41	32	stair	356	467	1925
lotfi	153	308	286	standata	359	1075	1275
maros	846	1443	896	standmps	467	1075	2445
nesm	662	2923	20625	stocfor1	117	111	20
perold	625	1376	727	tuff	333	587	196
pilot4	410	1000	453	vtpbase	198	203	5050
pilotja	940	1988	870	woodw	1098	8405	3312

Descriptive statistics for the remaining 74 problems are provided in Table 3. The number of constraints provided in Table 3 is for the MPS standard form.

6.2. *Numerical experiments*

In order to obtain a benchmark for quadratic LoQo, each of the test problems is attempted with both MINOS 5.3 (June, 1989) and LoQo. MINOS [22] is a well-known reduced gradient based solver for linear and nonlinear programs. All compu-

tational tests are performed on an IBM RISC System/6000 workstation running the UNIX operating system.

MINOS is a FORTRAN code compiled with the xlf compiler under the default optimization. In order to solve nonlinear programs, MINOS requires a user provided subroutine for evaluating the objective function and its gradient. This subroutine reads the matrix Q (in only the first call) from an external file and stores it in sparse matrix format. The sparse matrix representation allows us to assume that all variables are nonlinear without degrading performance. The external data file contains the lower portion of the Q matrix in format free input where each line provides: row number i; column number j; and associated value Q_{ij} of Q.

Whenever possible, MINOS is run with the default option. In each case we declare all variables to be nonlinear and suppress printing a solution by specifying the SOLUTION NO option. By default, optimality requires that the reduced gradient and the primal infeasibility norms are less than 1.0×10^{-6}.

LoQo is written in C and compiled with the AIX C compiler under the default optimization. The nonzeros in the lower part of Q are included in a special section of the MPS file (called QUADS). This section has the same format as the COLUMNS section but the row names now correspond to rows of Q and therefore are variable names themselves.

The test problems are run with all default options in LoQo except that the minimum degree ordering heuristic is invoked and a parameter specifying the minimum number of nonzeros in a dense column is provided. The parameter is high enough that no columns can be declared dense, and LoQo is forced to execute the *conservative strategy* in all cases. In this first set of tests, our goal is to demonstrate the viability of the simplest possible method. The next section provides results for problems with dense columns when this dense column threshold is not given.

Optimality requires primal and dual infeasibility to be less than 1.0×10^{-6}, and the relative duality gap to be reduced below 1.0×10^{-8}. The primal infeasibility is measured by

$$\frac{\|Ax - b\|_2}{\|b\|_2 + 1};$$

the dual infeasibility is

$$\frac{\|c + Qx - A^{\mathrm{T}}y - z\|_2}{\|c\|_2 + 1};$$

and the relative duality gap is

$$\frac{c^{\mathrm{T}}x - b^{\mathrm{T}}y + x^{\mathrm{T}}Qx}{|c^{\mathrm{T}}x + \frac{1}{2}x^{\mathrm{T}}Qx| + 1}.$$

Table 4 summarizes the times required by LoQo and MINOS to solve the test problems. The MINOS time is the internal *Mean time for entire program* reported in the output file, whereas the LoQo time is measured externally on the UNIX

Table 4
Total time in seconds

Problem	MINOS time	LoQo time	Problem	MINOS time	LoQo time
25fv47	540.29	(0)	pilotnov	401.84	142.19
adlittle	0.50	0.50	pilots	1695.73	(4)
afiro	0.09	0.16	pilotwe	424.84	95.10
agg	2.40	27.82	recipe	0.52	0.64
agg2	5.02	45.35	sc105	0.26	0.40
agg3	5.18	47.51	sc205	0.50	0.75
bandm	3.78	3.70	sc50a	0.14	0.23
beaconfd	1.77	2.38	sc50b	0.13	0.23
blend	0.47	0.53	scagr25	3.97	8.20
bnl1	68.80	21.34	scagr7	0.63	1.50
boeing1	50.44	(2)	scfxm1	4.11	(7)
boeing2	4.58	22.27	scfxm2	14.08	(7)
bore3d	1.26	2.67	scfxm3	27.57	(7)
brandy	2.82	2.41	scorpion	2.34	1.83
capri	3.65	38.17	scrs8	11.71	6.17
cycle	142.91	60.54	scsd1	3.43	1.82
degen2	24.18	88.54	scsd6	9.49	3.55
degen3	384.34	5438.90	scsd8	31.46	7.47
e226	6.70	3.38	sctap1	2.61	2.56
etamacro	19.59	58.30	sctap2	19.68	9.82
fffff800	25.14	107.11	sctap3	29.27	13.10
finnis	7.87	(7)	seba	7.19	287.96
fit1p	18.35	409.50	share1b	1.86	1.72
forplan	2.89	6.10	share2b	0.55	0.73
ganges	15.96	(4)	shell	29.11	269.53
gfrdpnc	11.56	3.50	ship04l	8.16	5.48
grow15	13.32	6.57	ship04s	4.74	3.68
grow22	34.19	11.70	ship08s	16.87	26.22
grow7	4.22	2.83	ship12s	29.85	32.25
israel	2.78	15.24	sierra	17.49	(1)
kb2	0.33	0.45	stair	6.00	45.58
lotfi	1.18	1.50	standata	2.54	3.90
maros	79.00	(1)	standmps	4.79	7.23
nesm	145.52	116.75	stocfor1	0.45	0.54
perold	246.80	136.70	tuff	11.59	7.83
pilot4	27.32	26.39	vtpbase	2.53	(4)
pilotja	590.82	304.50	woodw	111.33	76.17

command line by the time command. Once again, the maximum number of significant digits of agreement between the primal and dual objectives is given for problems that LoQo cannot solve. In all, LoQo solves 63 problems while MINOS solves all 74. Of the problems that are solved by both, MINOS solves 34 faster, LoQo solves 28 faster, and one takes the same amount of time on either solver. There are 11 problems that both solvers finish in under one second; MINOS outperforms LoQo on 10 of 11 of these, and the last is a tie. Eliminating these, MINOS is faster on 24 and LoQo on 28. The timings demonstrate that the interior

point method is certainly competitive, but is hampered by dense columns in A and density in Q. The problems with dense columns (fit1p, israel, and seba) are all solved much faster under LoQo's default pivot strategy in the following section. In addition, we suggest a formulaic strategy that prevents explicitly forming a dense Q in the following section. In particular, boeing2, degen2, degen3, fffff800, shell, ship08s, and ship12s are dramatically improved.

7. Separable equivalents

Each of the problems in the preceding section is considered along with a *separable equivalent.* We note that any positive definite quadratic program has a separable analog based on the Cholesky factorization of Q. That is, given a quadratic program in standard form

$$\begin{aligned} \text{minimize} \quad & c^{\mathrm{T}}x + \tfrac{1}{2}x^{\mathrm{T}}Qx \\ \text{subject to} \quad & Ax = b, \\ & x \geq 0, \end{aligned}$$

where Q is positive definite with Cholesky factor $\bar{L}$, we can introduce a set of variables $w = \bar{L}^{\mathrm{T}}x$ and obtain an equivalent separable quadratic program

$$\begin{aligned} \text{minimize} \quad & c^{\mathrm{T}}x + \tfrac{1}{2}w^{\mathrm{T}}w \\ \text{subject to} \quad & Ax = b, \\ & \bar{L}^{\mathrm{T}}x - w = 0, \\ & x \geq 0, \\ & w \text{ free}. \end{aligned} \tag{36}$$

The quadratic program (36) based on the Cholesky factorization has a number of drawbacks: it has n more constraints; it has n more variables — all of which are free; and it requires the Cholesky factorization of Q. In compensation for these costs, we obtain a separable quadratic program whose iterations are not much more difficult than those for a linear program with the same constraints [4]. It is often the case, however, that a positive semidefinite Q has already been formed naturally from some matrix S as $S^{\mathrm{T}}S$. When this is true, it may be advantageous not to actually form Q but to solve the following quadratic program instead:

$$\begin{aligned} \text{minimize} \quad & c^{\mathrm{T}}x + \tfrac{1}{2}w^{\mathrm{T}}w \\ \text{subject to} \quad & Ax = b, \\ & Sx - w = 0, \\ & x \geq 0, \\ & w \text{ free}. \end{aligned} \tag{37}$$

In addition to being *free*, the matrix S may not require n rows.

7.1. An example: The Markowitz model

The Markowitz [16] model for asset allocation is a widely solved nonseparable quadratic program for which there is a natural S used to create Q. The form of the model is generally

$$\begin{aligned} \text{minimize} \quad & x^{\mathrm{T}}Qx \\ \text{subject to} \quad & e^{\mathrm{T}}x = 1, \\ & \bar{r}^{\mathrm{T}}x \geq r_{\min}, \\ & x \geq 0, \end{aligned} \tag{38}$$

where the solution x is an investment decision that allocates funds among n different asset types to minimize the *risk* represented by the portfolio variance. In (38), e is an n vector of ones; $\bar{r}$ is the mean return for each asset category; and $r_{\min}$ is the minimum acceptable portfolio return. Typically, Markowitz models require few constraints. The constraints in (38) require all funds to be invested and demand that the expected portfolio return exceeds some minimum. In addition, there may be individual policy constraints. The model (38) is a basic Markowitz model and involves two constraints. The difficulty in solving such a model derives entirely from Q. The matrix Q is a sample covariance matrix. As such, it is positive semidefinite and *fully dense*. Its size is $n \times n$ where n is the number of asset categories. This may be massive. We note, however, that since Q is a sample covariance matrix,

$$Q = \frac{1}{n-1}(R - \bar{R})^{\mathrm{T}}(R - \bar{R}),$$

where R is a $t \times n$ matrix for which t is the number of time periods in which returns are observed and R_{ij} is the return for asset j in period i. $\bar{R}$ is a $t \times n$ matrix of mean returns where each of its t rows is simply $\bar{r}^{\mathrm{T}}$. Thus, we can define $Q = S^{\mathrm{T}}S$ where

$$S = \frac{1}{\sqrt{n-1}}(R - \bar{R}).$$

The Markowitz model (38) can, therefore, be rewritten as the following equivalent separable quadratic program:

$$\begin{aligned} \text{minimize} \quad & w^{\mathrm{T}}w \\ \text{subject to} \quad & e^{\mathrm{T}}x = 1, \\ & \bar{r}^{\mathrm{T}}x \geq r_{\min}, \\ & Sx - w = 0, \\ & x \geq 0, \\ & w \text{ free}. \end{aligned} \tag{39}$$

Recently, Konno and Suzuki [12] independently proposed (39) as a preferable formulation of the Markowitz model.

The quadratic program (39) has t new constraints. Often, however, it is the case that return data is compiled only monthly or quarterly, so the number of observations t may be small relative to the number of asset categories. In this case, the separable quadratic program (37) or (39) is much easier to solve than its nonseparable counterpart (18) or (38).

In practice, we feel that many nonseparable quadratic programs arise naturally based on a Q of the form S^TS. When S is available and has few rows relative to columns, the separable form (37) may be the appropriate form to consider. We illustrate the potential computational advantage by generating and solving separable analogs of our test problems. The size increase for the separable equivalents when $Q = A_r^T A_r$ varies depending on the size of A.

7.2. Numerical results

Since the Q matrix that appears in the test problems is derived based on a fraction r of the rows of A (in this case 5%), the separable equivalents are of the form

$$\begin{aligned} \text{minimize} \quad & c^T + \tfrac{1}{2} w^T w \\ \text{subject to} \quad & Ax = b, \\ & A_r x - w = 0, \\ & x \geqslant 0, \\ & w \text{ free.} \end{aligned} \tag{40}$$

The problem (40) has two salient features. First, A_r is constructed from the first r% of the rows of A, so the sparsity pattern in A_r is the same as that already present in A. Consequently, these rows contribute no new fill in the factorization. In general, this is not the case, and the additional rows can produce fill. Second, the number of additional rows ($0.05 \times m$) is small relative to the overall size in each of the test problems. Thus, even if Q is quite dense when explicitly formed, the indefinite system based on the separable equivalent is not much larger (or denser) than that of the original linear program. The Markowitz model is an application that also exhibits this feature.

The time required to solve each of the separable equivalent problems is provided in Table 5. The problems maros and sierra are not solved by either solver and are therefore omitted. Here again, the maximum number of digits of agreement between the primal and dual objective values is provided for problems that LoQo cannot solve. On the problem forplan, LoQo satisfies the complementarity requirement but is not dual feasible. MINOS declares that both problems it does not solve are infeasible. Once again, we force LoQo to use the conservative strategy by setting a parameter that specifies the minimum number of nonzeros in a column considered dense.

Table 5
Solving the separable equivalent - Total time in seconds

Problem	Nonzeros in Q	MINOS time	LoQo time	Problem	Nonzeros in Q	MINOS time	LoQo time
25fv47	41	205.93	100.86	pilotnov	48	387.86	149.20
aslittle	2	0.58	0.46	pilots	72	1681.37	(0)
afiro	1	0.12	0.16	pilotwe	36	144.76	59.00
agg	24	3.18	20.94	recipe	4	0.71	0.68
agg2	25	5.92	31.95	sc105	5	0.32	0.42
agg3	25	6.67	31.33	sc205	10	0.68	0.81
bandm	15	4.73	3.11	sc50a	2	0.15	0.25
beaconfd	8	2.75	2.41	sc50b	2	0.18	0.19
blend	3	0.62	0.51	scagr25	23	4.86	4.64
bnl1	32	75.45	19.41	scagr7	6	0.81	0.93
boeing1	17	12.80	26.96	scfxm1	16	4.71	(6)
boeing 2	8	2.32	3.71	scfxm2	33	15.58	(4)
bore3d	11	1.87	2.67	scfxm3	49	30.99	(6)
brandy	11	3.47	2.37	scorpion	19	2.89	1.96
capri	13	3.77	(7)	scrs8	24	13.43	5.81
cycle	95	135.98	68.30	scsd1	3	3.93	1.53
degen2	22	15.75	10.30	scsd6	7	10.61	2.98
degen3	75	309.10	175.71	scsd8	19	33.57	6.36
e226	11	6.77	2.71	sctap1	15	2.98	2.56
etamacro	20	14.58	45.48	sctap2	54	21.70	9.48
fffff800	26	20.61	30.59	sctap3	74	31.90	12.40
finnis	24	8.65	8.36	seba	25	8.63	196.48
fit1p	31	33.08	532.08	share1b	5	2.19	3.60
forplan	8	4.15	(14)	share2b	4	0.78	0.70
ganges	65	18.95	108.80	shell	26	8.06	11.22
gfrdpnc	30	13.34	3.63	ship04l	20	11.21	5.67
grow15	15	14.33	6.76	ship04s	20	6.67	3.90
grow22	22	31.19	10.99	ship08s	38	13.49	8.23
grow7	7	3.93	2.73	ship12s	57	23.48	9.51
israel	8	2.61	13.51	stair	17	7.46	36.63
kb2	2	0.43	0.40	standata	17	3.09	3.81
lotfi	7	1.29	1.50	standmps	23	5.28	5.14
nesm	33	130.52	86.59	stocfor1	5	0.63	0.49
perold	31	236.93	100.30	tuff	16	14.34	7.99
pilot4	20	28.99	24.26	vtpbase	9	1.52	(0)
pilotja	47	633.32	288.44	woodw	54	114.38	82.7

The size increase stated in Table 5 is the number of additional rows and columns that are included to create the separable formulation. In addition, it provides the number of variables appearing quadratically. MINOS is able to solve both forms of 72 problems. Of these, it solves 52 faster in the original nonseparable form; however, the total time for all test problems is smaller with the separable form. Hence, it is not clear which format is preferrable for MINOS. LoQo, on the other hand, is impeded by fill-in during the factorization. Density in Q contributes fill in the factorization, so LoQo benefits from considering the separable form. LoQo

solves both formulations of 61 of the test problems. Of these, only 16 are solved faster in the nonseparable form, while 41 are faster when posed in the separable form. (The solution time is unchanged for four.) Figure 3 illustrates the percent change in LoQo's solution time that results from solving the separable equivalent instead of the nonseparable formulation.

A total of 65 of the separable problems are solved by both MINOS and LoQo. LoQo solves 42 faster while MINOS is faster on 23. It is apparent that the performance of a particular solver is affected by the formulation it is presented. In the case of interior point methods, it is often advantageous to sacrifice size for sparsity. The separable equivalent formulation, though larger, can preserve sparsity. This is illustrated in Table 6 which provides the number of nonzeros in the factor L of K_q for both the separable and nonseparable formulations.

Further, the solution time for problems with dense columns is significantly reduced when no dense column threshold is supplied. In our test set, the problems fit1p, israel, and seba are characterized by dense columns. Allowing LoQo to automatically detect and delay eliminating dense columns, reduces computation time by factors of 4 to 67. This savings is largely due to the preservation of sparsity during

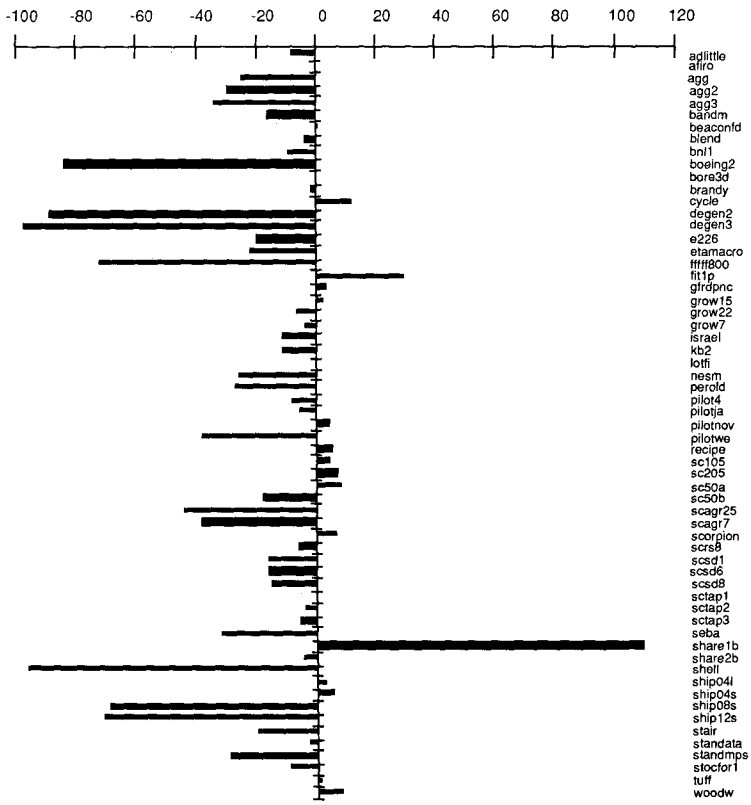

Fig. 3. Percent change in solution time arising from separable formulation.

Table 6
Nonzeros in factors

Problem	Nonzeros in L		Problem	Nonzeros in L	
	Separable	Nonseparable		Separable	Nonseparable
25fv47	49898	190128	pilotnov	72244	76782
adlittle	828	939	pilots	252407	479511
afiro	194	194	pilotwe	27787	36209
agg	20032	24245	recipe	1532	1485
agg2	27439	35219	sc105	856	848
agg3	27455	36445	sc205	1788	1794
bandm	7227	7009	sc50a	364	350
beaconfd	6175	6143	sc50b	342	337
blend	1566	1600	scagr25	4551	4634
bnl1	18420	19983	scagr7	1169	1126
boeing1	16846	192864	scfxm1	8125	11374
boeing2	5542	21875	scfxm2	16171	20044
bore3d	4473	4501	scfxm3	24193	27627
brandy	5601	5655	scorpion	3818	3672
capri	7967	12321	scrs8	10000	11106
cycle	97320	98116	scsd1	3848	4755
degen2	21394	82369	scsd6	7048	8984
degen3	152963	820680	scsd8	14945	19732
e226	6680	8281	sctap1	4446	4625
etamacro	23951	32553	sctap2	22546	23677
fffff800	28631	50692	sctap3	28799	31645
finnis	10062	19485	seba	60738	69125
fit1p	226547	192518	share1b	2667	2898
forplan	8946	9184	share2b	1909	1851
ganges	36629	36700	shell	8881	117468
gfrdpnc	4334	5112	ship04l	11054	11098
grow15	12204	12175	ship04s	7910	7944
grow22	18116	18225	ship08s	14321	46467
grow7	5509	5619	ship12s	17472	65559
israel	14353	15994	sierra	20025	19737
kb2	856	965	stair	21592	21790
lotfi	3022	3136	standata	6492	6998
maros	36972	45392	standmps	9138	12790
nesm	39278	52364	stocfor1	1408	1370
perold	33793	40852	tuff	13700	13571
pilot4	21034	23258	vtpbase	4676	18987
pilotja	70185	77700	woodw	85535	88150

factorization. Table 7 summarizes the timings obtained from solving problems with dense columns under several solution strategies, while Table 8 shows that the number of nonzeros in L is drastically reduced by delayed elimination of dense columns.

7.3. Free variables

Since the separable equivalent format can require the inclusion of many free variables, developing robust procedures for handling free variables is important.

Table 7

Solution time in seconds - Problems with dense columns

Problem	Conservative strategy		Default strategy		MINOS
	Nonseparable	Separable	Nonseparable	Separable	Nonseparable
fit1p	409.50	532.50	8.16	7.96	18.35
israel	15.24	13.51	3.57	3.34	2.78
seba	287.96	196.48	10.20	9.34	7.19

Table 8

Nonzeros in factors - Problems with dense columns

Problem	Conservative strategy		Default strategy	
	Nonseparable	Separable	Nonseparable	Separable
fit1p	192518	226547	11445	10703
israel	15994	14353	4675	4552
seba	69125	60738	6046	6104

Herein, we describe several options for handling quadratic free variables in the primal-dual interior point method. In particular, we consider three general free variable techniques: (1) the simplex method "trick" of splitting free variables into a difference of two nonnegative variables; (2) ignoring free variables in the primal while forcing equality in the associated dual constraints; and (3) allowing free variables to *flip signs*. The last method is incorporated in LoQo.

Perhaps the most widely used method for incorporating free variables is to write them as a difference of two nonnegative variables:

$$x_j = x_j^+ - x_j^-, \tag{41}$$

where x_j is free; $x_j^+ \geq 0$; and $x_j^- \geq 0$. Lustig, Marsten and Shanno [14] have incorporated this technique in OB1 — a primal-dual interior point method solver for linear programs. While it performs well on the NETLIB set, the basic method may become numerically unstable when there are a large number of free variables. Vanderbei [24] suggests that the reason for this stems from the fact that the dual constraint associated with a free variable must hold with equality. When x_j is rewritten using (41), the associated dual has two inequality constraints with slack variables z^+ and z^-, respectively. At an optimal solution, both z^+ and z^- must be zero. Thus, in the diagonal block D of K, at least one of the terms x^+/z^+ and x^-/z^- tends to infinity. In fact, both terms may go to infinity together. When free variables appear quadratically, this situation may be somewhat alleviated because $(Q + ZX^{-1})$ appears in K_q instead of just ZX^{-1}. Further, for separable quadratic programs. Carpenter, Lustig, Mulvey, and Shanno [4] show that splitting quadratic free variables yields a solution in which at least one of x^+ and x^- is zero.

Variable splitting is also applicable in the nonseparable case. If we consider the quadratic program (18) in which some of the variables are free, x, c, and A can be partitioned into bounded and free parts as follows:

$$x = [x_b | x_f], \qquad c = [c_b | c_f], \qquad A = [A_b | A_f]. \tag{42}$$

Likewise, Q can be written

$$Q = \begin{bmatrix} Q_b & Q_{bf}^T \\ Q_{bf} & Q_f \end{bmatrix}. \tag{43}$$

Now, the quadratic program (18) with free variables split is

$$\begin{aligned} \text{minimize} \quad & c_b^T x_b + c_f^T (x^+ - x^-) + \tfrac{1}{2}(x_b, x^+ - x^-)^T Q (x_b, x^+ - x^-) \\ \text{subject to} \quad & A_b x_b + A_f (x^+ - x^-) = b, \\ & x_b, x^+, x^- \geq 0. \end{aligned} \tag{44}$$

Letting $\hat{x} = (x_b, x^+, x^-)$ and

$$\hat{Q} = \begin{bmatrix} Q_b & Q_{bf}^T & -Q_{bf}^T \\ Q_{bf} & Q_f & -Q_f \\ -Q_{bf} & -Q_f & Q_f \end{bmatrix},$$

solving (44) is equivalent to solving

$$\begin{aligned} \text{minimize} \quad & c_b^T x_b + c_f^T (x^+ - x^-) + \tfrac{1}{2} \hat{x}^T \hat{Q} \hat{x} \\ \text{subject to} \quad & A_b x_b + A_f (x^+ - x^-) = b, \\ & x_b, x^+, x^- \geq 0. \end{aligned} \tag{45}$$

The positive semidefiniteness of Q guarantees that

$$(x_b, x^+ - x^-)^T Q (x_b, x^+ - x^-) \geq 0$$

for any x_b, x^+, x^-. And since

$$\hat{x}^T \hat{Q} \hat{x} = (x_b, x^+ - x^-)^T Q (x_b, x^+ - x^-)$$

we have that

$$\hat{x}^T \hat{Q} \hat{x} \geq 0$$

for any $\hat{x}$. Thus, (45) has a positive semidefinite objective and can be solved by the basic primal-dual interior point procedure. Unfortunately, because the quadratic program with split variables (45) is only positive semidefinite, x^+ and x^- may both tend to get large together, causing the same type of instability observed in the linear case.

A second approach for handling free variables is to simply write the dual constraint associated with a free variable as an *equality* constraint. The slack variables are therefore not included for these dual constraints. When there are n_b bounded primal variables x_b and n_f free variables x_f, the Newton system becomes

$$\begin{bmatrix} X_b & Z & 0 & 0 \\ I & -Q_b & -Q_{bf}^T & A_b^T \\ 0 & -Q_{bf} & -Q_f & A_f^T \\ 0 & A_b & A_f & 0 \end{bmatrix} \begin{bmatrix} \Delta z \\ \Delta x_b \\ \Delta x_f \\ \Delta y \end{bmatrix} = \begin{bmatrix} \phi \\ \sigma_{q_b} \\ \sigma_{q_f} \\ \rho \end{bmatrix}, \tag{46}$$

where σ_{q_b} is the first n_b and σ_{q_f} the remaining n_f elements of σ_q. Using the first set of equations to eliminate Δz yields

$$\underbrace{\begin{bmatrix} (X_b^{-1}Z + Q_b) & -Q_{bf}^T & A_b^T \\ -Q_{bf} & -Q_f & A_f^T \\ A_b & A_f & 0 \end{bmatrix}}_{K_q} \begin{bmatrix} \Delta x_b \\ \Delta x_f \\ \Delta y \end{bmatrix} = \begin{bmatrix} \sigma_{q_b} - X_b^{-1}\phi \\ \sigma_{q_f} \\ \rho \end{bmatrix}. \tag{47}$$

Ponceleón [23] noted that (47) can be solved as long as the pivot order in factoring K_q is appropriately chosen. Finding such a pivot strategy for K_q may be *easier* than for its linear programming counterpart K in which all Q blocks are zero. When there are few free variables, simply eliminating them after the bounded variables should perform well. This method has recently been implemented by Mehrotra [19] for linear programs and appears to avoid the numerical instability introduced by variable splitting.

Free variables are handled in LoQo via a variable transformation. This avoids creating extra variables as in splitting and mitigates numerical instability that can result from introducing zero in the leading diagonal elements of K (or K_q). Free variables in LoQo are ignored in the calculation of the primal steplength α_p. Thus, they are allowed to become negative, but after each Newton step, free variables are checked. Those that have become small are *rotated* in such a way that they are sufficiently positive in a transformed problem. Specifically, when a free variable x_j falls below 1.0 we consider a problem based on the new variable $x_j' = 2 - x_j$ and substitute $2 - x_j'$ for x_j in the original quadratic program. This necessitates change in the objective function, the right-hand side, and requires negation of a column of the constraint matrix. Applying this change of variables an even number of times returns the original quadratic program.

The relationship between the original and the transformed variables is shown in Figure 4. Whenever a free variable stays above 1.0 it is left alone; as soon as it falls below 1.0 it is rotated as shown. Figure 4 illustrates that a free variable must be greater than or equal to 1.0 in either the original or the transformed problem. In particular, a free variable never assumes the value zero (until the actual solution is recaptured from the transformed problem) which prevents converging to a point

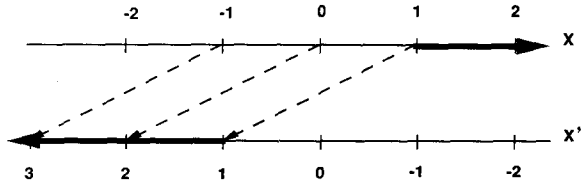

Fig. 4. Free variable transformation.

that does not satisfy $z_j = 0$. This approach was implemented in the ALPO linear program solver [24] and was easily extended to quadratic objectives for LoQo.

8. Conclusions and future directions

The numerical results in Sections 4, 6, and 7 demonstrate that a primal-dual interior point method based on solving the indefinite system is viable for solving linear and convex quadratic programs. Moreover, it eliminates the need to treat dense columns in the constraint matrix and nonseparable objectives as special cases. We believe that this additional level of generality will continue to yield computational benefits.

Still, LoQo is a development code. As such, we expect improvements both in terms of efficiency and robustness. First, we note that both ALPO and LoQo are intended to be *computational workbenches* designed for simplicity and ease of use. Thus, we feel that these ideas can certainly be incorporated in faster codes with more painstaking linear algebra. Also, we handle indefiniteness simply by choosing relatively conservative elimination strategies. In addition to this, the factorization can be made more robust by allowing *block* eliminations as a safeguard measure (see [23, 10, 6]).

While there is considerable computational experience documenting the effect of parameter choices in linear interior point methods, this is not true in quadratic programming. In LoQo, we use the same starting point for linear and quadratic programs. Interior point methods have been shown to be quite sensitive to starting point; therefore, a starting point tuned for quadratic programs should be beneficial. Also, the current implementation *always* takes equal primal and dual steps. Iterations were significantly reduced in linear programs by allowing independent choice of steplengths. Allowing independent steplengths in the primal and dual quadratic programs appeared to be less stable than the single step variant because the primal variables affect dual feasibility. Nevertheless, it may be advantageous to employ a combined strategy that enforces the single steplength requirement more stringently as optimality is approached.

Finally, the predictor-corrector method suggested by Mehrotra [18] has significantly reduced the time for methods based on the positive definite system to solve linear programs. In every iteration, the predictor-corrector variant performs two backsolves with the computed factorization. Thus, each iteration is more

expensive, but the method often performs fewer iterations and therefore fewer factorizations in all. The results of Mehrotra [18] and of Lustig, Marsten, and Shanno [15] demonstrate that the ensuing savings can be substantial. A predictor-corrector variant can also be implemented based on the indefinite system. In fact, Fourer and Mehrotra [6] include the predictor-corrector in their linear programming implementation. We believe that the quadratic solver may benefit even more than the linear solver because the required factorizations are inherently more difficult. We have begun experimentation with a predictor-corrector variant of LoQo, and the preliminary results are promising. The method solves 60 of the 74 nonseparable problems and yields improved solution times in all but three cases. In fact, the speedup is over two on 14 of the 54 problems that are solved by both versions of LoQo, and it exceeds seven in one case. These results, however, are only preliminary because the predictor-corrector implementation is still in the early stages of development.

Although further experimentation is required to tune both the overall method and the particular implementation, we feel that the primal-dual interior point method based on solving the indefinite system has clear promise. Computationally, it opens the class of convex quadratic programs for solution by a general purpose interior point solver.

Acknowledgement

We thank Irv Lustig and Dave Shanno for several useful suggestions pertaining to this work and Leslie Hall for a lively discussion of free variables. We also thank John Mulvey for teaching us about the Markowitz model and introducing us to [12]. We are also grateful to the referees for their careful consideration of this manuscript.

References

[1] I. Adler, N.K. Karmarkar, M.G.C. Resende and G. Veiga, "An implementation of Karmarkar's algorithm for linear programming," *Mathematical Programming* 44 (1989) 297-335.

[2] E.R. Barnes, "A variation on Karmarkar's algorithm for solving linear programming problems," *Mathematical Programming* 36 (1986) 174-182.

[3] J.R. Bunch and B.N. Parlett, "Direct methods for solving symmetric indefinite systems of linear equations," *SIAM Journal on Numerical Analysis* 8 (1971) 639-655.

[4] T. J. Carpenter, I.J. Lustig, J.M. Mulvey and D.F. Shanno, "Separable quadratic programming via a primal-dual interior point method and its use in a sequential procedure," to appear in: *ORSA Journal on Computing.*

[5] I.I. Dikin, "Iterative solution of problems of linear and quadratic programming," *Soviet Mathematics Doklady* 8 (1967) 674-675.

[6] R. Fourer and S. Mehrotra, "Performance of an augmented system approach for solving least-squares problems in an interior point method for linear programming," *Mathematical Programming Society COAL Newsletter* 19 (1991) 26-31.

[7] D.M. Gay, "Electronic mail distribution of linear programming test problems," *Mathematical Programming Society COAL Newsletter* 13 (1985) 10-12.

[8] A. George and J. Liu, *Computer Solution of Large Sparse Positive Definite Systems* (Prentice-Hall, Englewood Cliffs, NJ, 1981).

[9] P.E. Gill, W. Murray, D.B. Ponceleón and M.A. Saunders, "Preconditioners for indefinite systems arising in optimization," Technical Report SOL 90-8, Systems Optimization Laboratory, Stanford University (Stanford, CA, 1990).

[10] P.E. Gill, W. Murray and M.H. Wright, *Numerical Linear Algebra and Optimization, Vol. 1* (Addison-Wesley, Redwood City, CA, 1991).

[11] N.K. Karmarkar, "A new polynomial time algorithm for linear programming," *Combinatorica* 4 (1984) 373-395.

[12] H. Konno and K. Suzuki, "A fast algorithm for solving large scale mean-variance models by compact factorization of covariance matrices," Report IHSS 91-32, Institute of Human and Social Sciences, Tokyo Institute of Technology (Tokyo, 1991).

[13] I.J. Lustig, "Feasibility issues in a primal-dual interior point method for linear programming," *Mathematical Programming* 49 (1991) 145-162.

[14] I.J. Lustig, R. Marsten and D.F. Shanno, "Computational experience with a primal-dual interior point method for linear programming," *Linear Algebra and its Applications* 152 (1991) 191-222.

[15] I.J. Lustig, R.E. Marsten and D.F. Shanno, "On implementing Mehrotra's predictor-corrector interior point method for linear programming," *SIAM Journal on Optimization* 2 (1992) 435-449.

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

[17] R.E. Marsten, M.J. Saltzman, D.F. Shanno, G.S. Pierce and J.F. Ballintijn, "Implementation of a dual affine interior point algorithm for linear programming," *ORSA Journal on Computing* 1 (1989) 287-297.

[18] S. Mehrotra, "On the implementation of a (primal-dual) interior point method," Technical Report 90-03, Department of Industrial Engineering and Management Sciences, Northwestern University (Evanston, IL, 1990).

[19] S. Mehrotra, "Handling free variables in interior point methods," Technical Report 91-06, Department of Industrial Engineering and Management Sciences, Northwestern University (Evanston, IL, 1991).

[20] R.D.C. Monteiro and I. Adler, "Interior path following primal-dual algorithms. Part I: Linear programming," *Mathematical Programming* 44 (1989) 27-42.

[21] R.D.C. Monteiro and I. Adler, "Interior path following primal-dual algorithms. Part II: Convex quadratic programming," *Mathematical Programming* 44 (1989) 43-66.

[22] B.A. Murtagh and M.A. Saunders, "MINOS 5.1 user's guide," Technical Report SOL 83-20R, Systems Optimization Laboratory, Stanford University (Stanford, CA, 1987).

[23] D.B. Ponceléon, "Barrier methods for large-scale quadratic programming," Ph.D. Thesis, Stanford University (Stanford, CA, 1990).

[24] R.J. Vanderbei, "ALPO: Another linear program optimizer," Technical Report, AT&T Bell Laboratories (Murray Hill, NJ, 1990).

[25] R.J. Vanderbei, "A brief description of ALPO," Technical Report, AT&T Bell Laboratories (Murray Hill, NJ, 1990).

[26] R.J. Vanderbei, "Symmetric quasi-definite matrices," Technical Report SOR-91-10, Department of Civil Engineering and Operations Research, Princeton University (Princeton, NJ, 1991).

[27] R.J. Vanderbei, M.S. Meketon and B.F. Freedman, "A modification of Karmarkar's linear programming algorithm," *Algorithmica* 1 (1986) 395-407.