NUMERICAL LINEAR ALGEBRA WITH APPLICATIONS
Numer. Linear Algebra Appl. 2001; **00**:1–6 *Prepared using* nlaauth.cls *[Version: 2000/03/22 v1.0]*

Preconditioning KKT systems §

John C. Haws[1]*and Carl D. Meyer

[1] *The Boeing Company, P.O. Box 3707 MC 7L-21, Seattle, Washington, USA 98103.*

SUMMARY

Standard preconditioners, such as ILU factorizations, often break down in construction for KKT matrices. We compare several approaches for preconditioning KKT systems. The first applies a constraint preconditioner by incorporating robust factored approximations of $(BB^T)^{-1}$ and BB^T, where B is the (2,1) block of the KKT matrix. In the second approach, we apply permutations and scalings that maximize the product of the magnitude of the diagonal entries of the matrix. This preprocessing improves the reliability of standard preconditioning techniques, and we experiment with a threshold-based ILU preconditioner. Our numerical experiments compare these approaches with exact constraint preconditioning and with a direct solver. Copyright © 2001 John Wiley & Sons, Ltd.

KEY WORDS: preconditioning; iterative methods; symmetric indefinite systems; sparse approximate inverses; sparse optimal control

1. INTRODUCTION

We consider the solution of real symmetric indefinite linear systems $\mathcal{K}x = b$, where

$$\mathcal{K} = \begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix}, \tag{1}$$

via preconditioned Krylov subspace methods [24, 42]. Matrices of the form (1) are called KKT matrices, in reference to the Karush-Kuhn-Tucker first-order necessary optimality conditions for the solution of general nonlinear programming problems. KKT matrices arise in equality and inequality constrained nonlinear programming, sparse optimal control, and mixed finite-element discretizations of partial differential equations. We assume $\mathcal{K}$ is nonsingular, $H \in \mathbb{R}^{n\times n}$ is nonsingular, symmetric and possibly indefinite, and $B \in \mathbb{R}^{m\times n}$, with $m \leq n$.

---

§revision 2.1, dated October 28, 2002.

*Correspondence to: John C. Haws, The Boeing Company, P.O. Box 3707 MC 7L-21 Seattle, Washington 98124-2207, USA (email: john.c.haws@boeing.com).

Contract/grant sponsor: This work was funded in part by the NSF; contract/grant number: CCR-ITR0113121, DMS9714811, and CCR9731856

This paper details two new preconditioners for KKT systems. The first preconditioner relies on the structure of the KKT matrix: we apply the constraint preconditioner

$$\mathcal{P} = \begin{bmatrix} I & B^T \\ B & 0 \end{bmatrix} \tag{2}$$

via its inverse

$$\mathcal{P}^{-1} = \begin{bmatrix} I & B^T \\ B & 0 \end{bmatrix}^{-1} = \begin{bmatrix} I & -B^T \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -(BB^T)^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -B & I \end{bmatrix}. \tag{3}$$

The application of $\mathcal{P}^{-1}$ via the factorization (3) requires a solve with BB^T, and our preconditioners utilize approximate factorizations of $(BB^T)^{-1}$ and BB^T.

When forming an incomplete factorization of BB^T, an intuitive approach is to form an incomplete QR factorization for B^T. For $QR \approx B^T$, then $R^TR \approx BB^T$. Strategies for forming incomplete QR factorizations include incomplete Gram-Schmidt methods [1, 40, 47], and methods based on Givens rotations [29, 50]; for an overview, see [13]. However, avoiding breakdowns for general symmetric positive definite matrices for these approaches is a non-trivial matter; see, for example, the appendix of James' PhD thesis [28].

We approximate $(BB^T)^{-1}$ with the stabilized approximate inverse (SAINV) of Benzi, Cullum, and Tuma [5], and we approximate BB^T with the related robust incomplete factorization RIF [10, 11]. The factorizations are guaranteed not to break down in exact arithmetic for symmetric positive definite matrices. The versions of RIF and SAINV we use involve only sparse matrix-vector products with the matrix B^T.

Our second approach to preconditioning KKT matrices arises out of efforts to improve robustness for general nonsymmetric, highly indefinite matrices [6]. Due to the structure of KKT systems, with the large zero (2,2) block, standard preconditioning techniques such as LU factorizations and factorized approximate inverses can breakdown due to zero pivots. For example, pivoting strategies to avoid breakdown were examined in [40, 49]. In [6], nonsymmetric permutations and scalings that place entries of large magnitude on the diagonal of a matrix were shown to improve the robustness and effectiveness of standard preconditioning techniques. In this paper, we experiment with these permutations combined with ILUT preconditioning [41] for KKT systems. Our experiments show that this is an effective technique, albeit one which destroys the symmetry of the original system.

When the $(1,1)$ block H is symmetric positive definite (SPD) in the nullspace of the constraint matrix B^T (i.e. when X^THX is SPD with the columns of X forming a basis for the nullspace of B^T), the linear system $\mathcal{K}x = b$ can be solved with the reduced conjugate gradient method [22, 23, 30]. This is an extremely effective application of constraint preconditioning, and the approach can be competitive with a direct solver [30]. However, for the optimal control problems, we have no guarantee that H is SPD in the nullspace of the constraints; see Section 5 and our discussion of the optimal control problems. The use of reduced conjugate gradients in this context is not straightforward, and we do not address it in this paper.

The paper is organized as follows: in Section 2 we describe constraint preconditioners, we discuss scalings, and we describe how to approximate constraint preconditioners using factorizations of $(BB^T)^{-1}$ and BB^T; in Section 3 we describe the algorithms SAINV and RIF and the variations that lead to the approximations of $(BB^T)^{-1}$ and BB^T; Section 4 gives a brief description of algorithms for maximizing the product of entries on the diagonal of a matrix; we describe the test matrices in Section 5; in Section 6, we present experiments

comparing exact constraint preconditioning to our approximate constraint preconditioners, to standard ILU preconditioning combined with the permutations of Section 4, and to a direct method based on a symmetric indefinite factorization. We conclude in Section 7.

2. THE CONSTRAINT PRECONDITIONERS AND SCALINGS

In this section, we describe the scalings and constraint preconditioners applied to the KKT matrix. Our preconditioners have three features: we scale the (1,1) block H to have unit diagonal (when H has all positive diagonal elements), we scale the (2,1) and (1,2) blocks so that BB^T has unit diagonal, and we approximate $(BB^T)^{-1}$ or BB^T.

In some cases, an efficient factorization of the preconditioner $\mathcal{P}$ may be obtained, but this is not the case in general. The factorization (3) shows that solves with BB^T, often denser than B, are the most computationally expensive aspects of applying the preconditioner. As in [38], we exploit an approximation to ease computational complexity. One approach hinges on the use of SAINV to approximate $(BB^T)^{-1}$; that is, the preconditioner may be written

$$\tilde{\mathcal{P}}^{-1}_{\text{SAINV}} = \begin{bmatrix} I & -B^T \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -ZD^{-1}Z^T \end{bmatrix} \begin{bmatrix} I & 0 \\ -B & I \end{bmatrix}, \tag{4}$$

where $ZD^{-1}Z^T \approx (BB^T)^{-1}$. As an alternative, we also use RIF to approximate BB^T, and in that case, the preconditioner may be written

$$\tilde{\mathcal{P}}_{\text{RIF}} = \begin{bmatrix} I & B^T \\ B & 0 \end{bmatrix} = \begin{bmatrix} I & 0 \\ B & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & -LDL^T \end{bmatrix} \begin{bmatrix} I & B^T \\ 0 & I \end{bmatrix}, \tag{5}$$

where $LDL^T \approx BB^T$.

2.1. DIAGONAL SCALINGS

When the (1,1) block of the KKT matrix is positive definite, the identity matrix in the (1,1) block of the constraint preconditioner (2) may be replaced with $G = \text{diag}(H)$. This approach provides at least as good a preconditioner as the unscaled version with identity matrix in the (1,1) block (see Section 2.2 below). We describe an equivalent approach in which, when H has all positive diagonal entries, we diagonally scale the original matrix so that the scaled matrix has unit diagonal in the (1,1) block.

We also scale the (2,1) and (1,2) blocks of the KKT matrix. In [5], diagonal scaling improves the quality of the SAINV factorization for general symmetric positive definite matrices. We employ a similar scaling and guard against the impact on the B matrices of the scaling from the (1,1) block. We scale $\mathcal{K}$ so that BB^T also has unit diagonal, accounting for any scaling of the (1,1) block.

To describe the scalings in detail, let b_i denote the ith row of the matrix B. The first block row and column of $\mathcal{K}$ are scaled by the $n \times n$ diagonal matrix

$$D_H = \begin{cases} (\sqrt{\text{diag}(H)})^{-1} & \text{if } \text{diag}(H) > 0, \\ I & \text{otherwise.} \end{cases}$$

Let $\bar{B}^T = D_H B^T = [\bar{b}_1, \ldots, \bar{b}_m]$, and define the $m \times m$ diagonal matrix D_B by

$$(D_B)_{ii} = \|\bar{b}_i\|_2^{-1}.$$

The second block row and column of $\mathcal{K}$ are scaled by the diagonal matrix D_B. The scaling matrix for $\mathcal{K}$ is

$$\mathcal{D} = \begin{bmatrix} D_H & 0 \\ 0 & D_B \end{bmatrix}.$$

The scaling $\mathcal{D}\mathcal{K}\mathcal{D}$ implicitly includes $G = \text{diag}(H)$ in the (1,1) block of (2), and also ensures BB^T has a unit diagonal. That is, the scaled matrix $\bar{\mathcal{K}} = \mathcal{D}\mathcal{K}\mathcal{D}$ is

$$\begin{bmatrix} H_s & B_s^T \\ B_s & 0 \end{bmatrix} = \begin{bmatrix} D_H & 0 \\ 0 & D_B \end{bmatrix} \begin{bmatrix} H & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} D_H & 0 \\ 0 & D_B \end{bmatrix}. \tag{6}$$

For the remainder of this paper, unless otherwise noted, we assume $\mathcal{K}$ is scaled as in (6). For simplicity, we drop the associated notation.

2.2. REMARKS

The constraint preconditioner (2) has been shown to be effective for KKT systems [23, 26, 30, 35, 37, 38, 39]. In some cases, the identity matrix in the (1,1) block of (2) is replaced with some approximation to the (1,1) block of the original matrix (1), say with some $G \approx H$. A common choice is to set $G = \text{diag}(H)$; as detailed in [30], this approach can produce a better preconditioner by improving the clustering of the eigenvalues of the preconditioned coefficient matrix $\mathcal{P}^{-1}\mathcal{K}$. Let $X \in \mathbb{R}^{n\times(n-m)}$ be a matrix whose columns form a basis for the nullspace of B^T. Then the preconditioned matrix has an eigenvalue 1 with multiplicity $2m$, and $n-m$ eigenvalues defined by the coefficient matrix

$$(X^T G X)^{-1} X^T H X. \tag{7}$$

This observation is based upon the fact that the preconditioned matrix $\mathcal{P}^{-1}\mathcal{K}$ is similar to a matrix of the form

$$\begin{bmatrix} I & 0 & 0 \\ \star & (X^T G X)^{-1} X^T H X & 0 \\ \star & \star & I \end{bmatrix},$$

and the eigenvalues of the matrix (7) are clustered inside those of $G^{-1}H$. That is, if G is a good approximation to H (in a preconditioning sense), then $X^T G X$ is at least as good an approximation to $X^T H X$. See [30] or [23] for details.

3. THE APPROXIMATE INVERSE SAINV AND INCOMPLETE FACTORIZATION RIF

In this section, we describe the Stabilized Approximate Inverse (SAINV) and Robust Incomplete Factorization (RIF) algorithms and variants that lead to efficient robust factored approximations of $(BB^T)^{-1}$ and BB^T. A robust factorized approximate inverse for symmetric positive definite matrices was described in [5]; see also [7, 8, 31, 32]; RIF is described in [10, 11].

3.1. APPROXIMATING $(BB^T)^{-1}$: SAINV

Both SAINV and RIF are based upon an incomplete A-conjugation algorithm. Let A be a general $m \times m$ symmetric positive definite matrix. The A-conjugation process constructs an upper triangular factor Z and diagonal matrix D such that

$$Z D^{-1} Z^T = A^{-1}.$$

In exact arithmetic, Z is the inverse of the L^T factor in the LDL^T decomposition of A, and the diagonal D matrix is the same in the two factorizations. The formal algorithm is given in Algorithm 1, where e_i denotes the ith unit basis vector.

Algorithm 1 A-orthogonalization

1: $z_i = e_i, \quad i = 1, \ldots, m.$
2: **for** $i = 1, 2, \ldots, m$ **do**
3: compute pivots:
4: $p_j = z_i^T A z_j, \qquad j = i, \ldots, m$
5: update z_j's:
6: $z_j = z_j - \frac{p_j}{p_i} z_i, \qquad j = i+1, \ldots, m$
7: $Z = [z_1, z_2, \ldots, z_m], \quad D = diag(p_1, p_2, \ldots, p_m)$
8: **end for**

The process is made incomplete (and the factors are kept sparse) by dropping elements according to some rule. In the SAINV algorithm, small off-diagonal elements in the factor Z are dropped after the update in step 6 of Algorithm 1, with no a-priori sparsity pattern determined. The idea is that when many entries in L^{-1} are small, the incomplete Z^T will be a good approximation to L^{-1}.

While the number of nonzeros in A^{-1} is independent of the ordering of the rows and columns of A, the sparsity of the inverse of the L factor is highly dependent on the ordering. In practical implementations of SAINV, sparsity is preserved by combining the dropping of small elements with a sparse reordering such as multiple minimum degree [34], reverse Cuthill-McKee [16], or generalized nested dissection [33]. If an ordering that limits sparsity in the inverse factors is applied to the matrix, then we limit the loss of information due to dropping in an incomplete conjugation process. Such strategies have been shown to be effective in reducing construction and storage requirements for SAINV and also in improving the effectiveness of the preconditioner [9, 14].

Note that Algorithm 1 requires only matrix-vector products with A in step 4. If we define breakdown as encountering a zero or negative pivot p_i at step i of the algorithm, then the positive definiteness of A guarantees no breakdown with this formulation. The z_i's are initialized to unit basis vectors, no dropping is applied to diagonal elements, and at iteration i only elements $1, \ldots, i-1$ of z_i through z_m are modified. Thus the z_i's remain non-zero throughout the incomplete conjugation process. Indeed, because we are assuming A is symmetric positive definite,

$$p_i = z_i^T A z_i > 0 \quad \forall \quad z_i \neq 0.$$

This is desirable, as a negative pivot would result in an approximation that is not positive definite; a zero pivot would lead to division by zero.

3.2. APPROXIMATING BB^T: RIF

We return to the factors Z and D computed by the exact A-conjugation process, such that $ZD^{-1}Z^T = A^{-1}$. Note that the Z factors are A-conjugate:

$$Z^T A Z = D.$$

From this relation, and using the fact that (in exact arithmetic) $Z^T = L^{-1}$, we have $AZ = LD$. Let a_j^T denote the jth row of A. Then $L_{ji} = p_j^{-1} a_j^T z_i$. Since Z is unit upper triangular,

$$a_j^T z_i = z_j^T A z_i.$$

That is, using the notation of Algorithm 1,

$$L_{ji} = p_i / p_j. \tag{8}$$

Again, we are automatically protected against breakdown due to the definiteness of A and the method of computing the pivots p_j:

$$p_j = z_j^T A z_j.$$

Therefore, we can use the A-conjugation algorithm to compute an incomplete Cholesky factorization by computing the elements (8), and breakdown is avoided for all positive definite matrices. There are two dropping points to focus on to maintain sparsity:

1. drop elements after updating the z vectors in step 6 (as in the SAINV) algorithm, and
2. drop small elements L_{ji} after the computation given by (8).

Note that the second choice in dropping is an *a posteriori* dropping strategy, and thus has no effect on the accuracy of the subsequent computations. The first dropping point is necessary to maintain sparsity in the z_i's. The second dropping is usually unnecessary (for sparsity) and offers little improvement in preconditioner performance. Often the associated drop tolerance can simply be set to zero; see [10].

3.3. APPROXIMATING NORMAL EQUATIONS

Details on the use of incomplete orthogonalization algorithms to construct incomplete factorizations, in the context of least squares, are given in [11]. Here we provide a brief summary.

With a simple variation, the SAINV algorithm constructs the factor Z and diagonal matrix D, such that

$$ZD^{-1}Z^T \approx (BB^T)^{-1}$$

and the RIF algorithm constructs the factor L and diagonal matrix D such that

$$LDL^T \approx BB^T.$$

In both cases, the construction is guaranteed to be breakdown-free for symmetric positive definite matrices, and requires only matrix-vector products with B^T; one need not explicitly form BB^T.

To see that breakdown is avoided, and the product BB^T need not be formed, consider the calculation of the p_j's:

$$\begin{aligned} p_j &= z_i^T (BB^T) z_j \\ &= (B^T z_i)^T B^T z_j. \end{aligned}$$

Then the final pivots $p_i = (B^T z_i)^T B^T z_i$ will be positive when B has full rank. The formal modification of Algorithm 1 is straightforward and given in Algorithm 2, where B is a $m \times n$ matrix with full row rank.

Algorithm 2 Orthogonalization for normal equations

1: $z_i = e_i, \quad i = 1, \ldots, m.$
2: **for** $i = 1, 2, \ldots, m$ **do**
3: compute pivots:
4: $v_j = B^T z_j, \quad p_j = v_i^T v_j, \quad j = i, \ldots, m$
5: update z_j's:
6: $z_j = z_j - \frac{p_j}{p_i} z_i, \quad j = i+1, \ldots, m$
7: $Z = [z_1, z_2, \ldots, z_m], \quad D = diag(p_1, p_2, \ldots, p_m)$
8: **end for**

4. PERMUTING LARGE ENTRIES TO THE DIAGONAL

In this section, we describe permutations that place large entries on the diagonal of a matrix [17, 18]. The permutations maximize the product of the absolute values of the entries along the diagonal, which improves the reliability and performance of preconditioned iterative solvers [6]. We apply these permutations in combination with an ILU preconditioner to the whole of $\mathcal{K}$. This approach differs from that described in the Section 2, especially in that we disregard any symmetry in the matrix and apply different scalings.

Breakdown-free construction for ILU preconditioners is guaranteed only for special matrices, such as H-matrices. In particular, very small or zero pivots can lead to instabilities during the construction. Instabilities associated with ILU preconditioners are described in detail in [15]. However, the constructions are guaranteed to be breakdown-free for diagonally dominant matrices.

By employing nonsymmetric permutations and scalings that place entries of large magnitude on the diagonal, these instabilities can often be remedied, thereby dramatically improving the effectiveness of standard preconditioners. The algorithms in MC64, a set of FORTRAN subroutines from HSL [27] (formerly the Harwell Subroutine Library), determine permutations of a matrix that place entries of large absolute value on the diagonal. Details on the algorithms and implementations are described in [17, 18]; the use of the algorithms for improving the robustness of standard preconditioning techniques for nonsymmetric and highly indefinite systems is examined in [6].

In [6], one of the most effective techniques involved permuting a matrix so that the product of the absolute values of the entries on the diagonal is maximized. The strategy was introduced in [36] for pivoting in dense Gaussian elimination. Because the technique is applicable to nonsingular matrices in general, let $A = (a_{ij})$ denote a general $N \times N$ matrix. What follows is a brief description of the technique.

A matching is a set $\mathcal{M}$ of at most N ordered pairs $(i, j), 1 \leq i, j \leq N$ in which each row index i and each column index j appears at most once. When $\mathcal{M}$ has maximum cardinality (N for structurally nonsingular matrices), $\mathcal{M}$ is called a *maximum transversal* or *maximum matching*.

In the case where $|\mathcal{M}| = N$, then $\mathcal{M}$ defines an $N \times N$ permutation matrix $Q = (q_{ij})$, where

$$\begin{cases} q_{ji} = 1 & \text{for } (i, j) \in \mathcal{M}, \\ q_{ji} = 0 & \text{otherwise.} \end{cases}$$

Then QA is a matrix with the transversal entries on the diagonal.

To maximize the product of the entries on the diagonal, we look for a permutation σ that maximizes

$$\prod_{i=1}^{N} |a_{\sigma(i)i}|. \tag{9}$$

The maximization problem is translated into a minimization problem by defining the matrix $C = (c_{ij})$ by

$$c_{ij} = \begin{cases} \log a_i - \log |a_{ij}|, & \text{for } a_{ij} \neq 0, \\ \infty, & \text{otherwise,} \end{cases}$$

where $a_i = \max_j |a_{ij}|$. Then maximizing (9) is equivalent to minimizing

$$\sum_{i=1}^{N} c_{\sigma(i)i}. \tag{10}$$

Minimizing the sum (10) is equivalent to finding a minimum weight perfect matching. In combinatorial optimization, this is known as a bipartite weighted matching problem, and MC64 uses a sparse variant of the bipartite weighted matching algorithm introduced in [36].

The application of the bipartite weighted matching algorithm to a matrix C produces an important byproduct. A perfect matching $\mathcal{M}$ has minimum weight if and only if there exist vectors $u = (u_i)$ and $v = (v_i)$, each of length N, such that

$$\begin{cases} u_i + v_j = c_{ij} & \text{for } (i,j) \in \mathcal{M}, \\ u_i + v_j \leq c_{ij} & \text{for } (i,j) \notin \mathcal{M}. \end{cases} \tag{11}$$

These vectors are used to scale the matrix. Using the vectors u and v from (11), define the diagonal matrices D_1 and D_2 by

$$\begin{aligned} D_1 &= \operatorname{diag}(d_1^1, d_2^1, \ldots, d_N^1), \quad d_j^1 = \exp(v_j)/a_j, \\ D_2 &= \operatorname{diag}(d_1^2, d_2^2, \ldots, d_N^2), \quad d_j^2 = \exp(u_i). \end{aligned} \tag{12}$$

Then QD_1AD_2 is a matrix whose diagonal entries are 1 in absolute value and whose off-diagonal entries are all less than or equal to 1, in absolute value. We can simply change the signs of rows to obtain a matrix with unit diagonal. The eigenvalues of such a matrix lie in discs centered at 1, with radii equal to the sum of the absolute values of the off-diagonal entries in the corresponding row. In the ideal case, the radii of the discs will be less than 1, and the matrix will be row-wise diagonally dominant, which guarantees that an ILU preconditioner will be well defined.

In [6], the combination of these reorderings and scalings with ILUT factorizations provided a very effective preconditioner for difficult problems. We pursue this preconditioning approach for KKT matrices in our experiments.

5. DESCRIPTION OF TEST MATRICES

In all experiments, we solve linear systems $\mathcal{K}x = b$, where the coefficient matrix has KKT structure.

The first set of matrices used in our experiments are *stiffness matrices and mass matrices* from mixed finite-element approximations of variational problems. The (1,1) block of these

matrices is symmetric positive definite. These matrices were examined in [30] and are described therein.

Our second set of matrices comes from *sparse optimal control*, and in particular from the Boeing Company's sparse optimal control software SOCS [44]. For these problems, the (1,1) block H represents an approximate Hessian of the Lagrangian for a nonlinear optimization problem and is not guaranteed to be definite; the B blocks correspond to constraints.

The test matrices come out of the nonlinear programming (NLP) subproblem that results when the optimal control problem is converted from a dynamic system of continuous functions to a finite-dimensional problem. The Hessian matrix used in the NLP subproblem is

$$H = H_L + \tau(\|\sigma\| + 1)I$$

where H_L approximates the Hessian of the Lagrangian (and is in general not positive definite), τ is a Levenberg parameter, σ is a bound on the most negative eigenvalue of H_L, and I is the identity matrix. For a certain choice of τ (e.g. $\tau = 1$), the reduced H will be definite and the reduced conjugate gradient method would be applicable and probably the iterative method of choice. However, for fast convergence of the NLP algorithm, τ should be small. SOCS tries to choose τ carefully to obtain fast convergence, but it is possible that τ is chosen so that the reduced H is not positive definite. In that case, τ is increased.

To this end, SOCS currently computes a symmetric indefinite factorization of the KKT matrix and calculates its inertia, and thereby (using results from [21]) checks the definiteness of H. When the inertia does not align with the dimensions of the KKT matrix, i.e. when the reduced Hessian is not positive definite, the Levenberg parameter is increased, and the matrix is refactored. Once the reduced Hessian is positive definite, a new search direction is computed.

However, if a symmetric indefinite factorization is not used (e.g. if an iterative solver replaces the direct solver), we do not have easy access to the inertia, and thus must resort to some other means of checking if the computed search direction is "downhill". This is the scenario we are exploring with our experiments. For details on the NLP algorithms and on sparse optimal control problems, see [12].

6. NUMERICAL EXPERIMENTS

We experiment with two approaches for exact constraint preconditioning, two approximate constraint preconditioners, a more general preconditioner, and a direct solver. For both constraint preconditioning and approximate constraint preconditioning, we scale the matrix as in (6).

We apply exact *constraint preconditioning* with 2 different approaches.

CP–MA49: We compute an exact QR factorization of B^T to apply the preconditioner (3), using MA49, the HSL code for sparse QR factorization.

CP-EXT: We factor the constraint preconditioner directly, using the Boeing Company's sparse matrix package BCSLIB-EXT [4]. We compute an LDL^T factorization of the constraint preconditioner (2), in which the D matrix is block-diagonal, with blocks of size 1×1 or 2×2. For details on the algorithm see [3, 19].

We compute an *approximate constraint preconditioner* with two related algorithms.

CP–SAINV: We approximate the exact constraint preconditioner by approximating $(BB^T)^{-1}$ using the SAINV factorization. When constructing the SAINV factorization, we use a default drop tolerance of 0.1 for the underlying incomplete orthogonalization process. For the matrix mass06, we use a drop tolerance of 0.01, which provided better results than 0.1.

CP–RIF: We approximate the exact constraint preconditioner by approximating BB^T using the RIF factorization. When constructing the RIF factorization, we use a drop tolerance of 0.1 for the underlying incomplete orthogonalization process, and we use no a posteriori dropping.

Multiple minimum degree (MMD) orderings generally improve the performance of SAINV [9], and so for both CP–SAINV and CP–RIF we reorder the columns of B^T so that the product BB^T has MMD ordering.

With **MC64–ILUT**, we *permute large entries to the diagonal* with the permutations described in Section 4. We then construct a standard ILUT preconditioner [41]. We do not apply the scalings described in Section 2.1. For the mass and stiffness matrices, we use the reverse Cuthill-McKee (RCM) symmetric ordering after applying the MC64 scalings and one-sided permutations. For the optimal control problems, we use multiple minimum degree (MMD). While these orderings may not provide the best performance among all available orderings, they provided good performance for our problems. For the ILUT factorization, we used a drop tolerance of 0.01 and fill limit of 10 nonzeros per row for the mass and stiffness matrices. For the optimal control matrices, we used drop tolerance and fill limit of 0.0001 and 15, respectively (0.00001 and 25 for the matrices traj27 and traj33).

Finally, for completeness, we include experiments where the original linear system is solved with the *direct solver* BCSLIB-EXT. **EXT** denotes these experiments. We use a MMD reordering for the direct solver, except for the matrices mass05 and mass06, for which we used GND (we do not apply the scalings of Section 2.1).

For all methods, we did not fine tune input parameters or symmetric ordering choices. When we deviate from defaults choices, it is because other options failed. In some cases, performance could certainly be improved by tweaking. However, our aim is to compare the preconditioners from a common standpoint, and hence we left most parameters at default settings.

We use BiCGStab [45] as the iterative method. When using symmetric indefinite preconditioners, an alternative choice is simplified QMR [20]. Because the preconditioners produced by MC64–ILUT are nonsymmetric, we used BiCGStab. Right-hand sides were constructed using the original matrix $\mathcal{K}$ and the solution vector $\hat{x} = [1, 2, \ldots, N]^T$, where N represents the dimension of the coefficient matrix $\mathcal{K}$. For each matrix, we use the zero vector for the initial guess. In all cases, iterations were terminated when the relative residual (that is, the current residual divided by the right-hand side), measured in the 2-norm, was reduced by at least 10^{-8}, or when a maximum of 2000 iterations were completed.

All tests were performed on an Advanced Micro Devices (AMD) Duron 800Mhz workstation running Linux, kernel release 2.4.18. Routines were linked against the automatically tuned ATLAS BLAS libraries version 3.4.1 [48]. CPU times were computed using the Fortran 95 intrinsic function `cpu_time`.

Experimental results are reported in Tables I and II. For each matrix, we report dimensions n and m and total nonzeros in the coefficient matrix $\mathcal{K}$. For each solve methodology, we report setup and solve times, nonzeros in stored factors, iteration counts (when applicable), and total

Table I. Comparison of preconditioners for stiffness and mass matrices.

		stiff4	stiff5	mass04	mass05	mass06
	n	8450	33282	8450	33282	33282
	m	46	128	46	128	512
	nnz	41318	177256	56818	241012	257220
CP–MA49	nnz	144	1563	144	1563	11082
	its	353	469	19	21	17
	setup	0.002	0.016	0.001	0.016	0.067
	solve	2.226	13.963	0.134	0.692	0.690
	total	2.228	13.979	0.135	0.708	0.757
CP–EXT	nnz	121681	505714	121681	505714	579439
	its	359	469	19	21	17
	setup	0.064	0.305	0.063	0.302	0.401
	solve	4.795	28.536	0.251	1.342	1.299
	total	4.859	28.841	0.314	1.644	1.700
CP–SAINV	nnz	46	624	46	624	9464
	its	353	583	19	41	37
	setup	0.058	0.065	0.049	0.057	0.086
	solve	2.163	16.569	0.132	1.288	1.125
	total	2.221	16.634	0.181	1.345	1.211
CP–RIF	nnz	349	584	75	584	3132
	its	75	541	19	35	57
	setup	0.050	0.077	0.049	0.078	0.105
	solve	2.329	15.499	0.132	1.094	1.968
	total	2.379	15.576	0.181	1.172	2.073
MC64–ILUT	nnz	77118	341417	68875	323299	394821
	its	28	135	3	32	30
	setup	0.129	0.575	0.126	0.705	0.806
	solve	0.786	15.482	0.186	3.902	4.038
	total	0.915	16.057	0.312	4.606	4.844
EXT	nnz	209412	1263343	284867	2899009	2658291
	setup	0.252	1.533	0.295	10.578	6.535
	solve	0.021	0.091	0.024	0.173	0.160
	total	0.273	1.624	0.319	10.751	6.695

solution time. For the preconditioners, setup times include reorderings and preconditioner construction, and solve times include accelerator iterations and preconditioner solves. For the direct solver EXT, setup times include matrix reordering and factorization. Solve times include solves with the factors.

For the matrices in Table I, all methods converged to a solution within 2000 iterations, and for the stiffness matrices the direct solver gave the fastest total solution time. Note that for all five problems in Table I, EXT gave the fastest solve time, an advantage when setup data can be reused. CP–MA49 was in general the most effective iterative approach, with the fastest total solution times for the mass matrices and the fastest preconditioning approach for stiff5. CP–EXT took twice as long as CP–MA49 for these problems: the CP–EXT factors encountered high amounts of fill.

The approximate constraint preconditioners CP–SAINV and CP–RIF performed reasonably well for the mass and stiffness matrices in Table I; the approaches were not the fastest, but

Table II. Comparison of preconditioners for sparse optimal control problems.

		capt12	plnt01	heat02	lnts09	traj27	traj33
	n	1271	1455	5197	9994	10003	12861
	m	1172	1362	5098	7996	7145	7145
	nnz	26935	51529	85031	95295	235141	496945
CP–MA49	*nnz*	51302	175276	243788	†	†	†
	its	9	11	7	†	†	†
	setup	0.056	0.402	0.287	†	†	†
	solve	0.063	0.144	0.221	†	†	†
	total	0.119	0.546	0.508	†	†	†
CP–EXT	*nnz*	37440	43794	705112	260708	283741	250083
	its	9	11	7	951	9	9
	setup	0.039	0.047	0.844	0.360	1.508	0.367
	solve	0.055	0.078	0.328	39.312	0.421	0.531
	total	0.094	0.125	1.172	39.672	1.929	0.898
CP–SAINV	*nnz*	—	—	—	—	—	—
	its	—	—	—	—	—	—
	setup	—	—	—	—	—	—
	solve	—	—	—	—	—	—
	total	—	—	—	—	—	—
CP–RIF	*nnz*	—	—	—	—	—	—
	its	—	—	—	—	—	—
	setup	—	—	—	—	—	—
	solve	—	—	—	—	—	—
	total	—	—	—	—	—	—
MC64–ILUT	*nnz*	36124	46328	220806	140036	326554	369391
	its	4	7	24	15	49	48
	setup	0.109	0.158	0.485	2.115	15.105	7.295
	solve	0.085	0.117	1.225	0.970	4.590	6.133
	total	0.194	0.275	1.710	3.085	19.695	13.429
EXT	*nnz*	38188	67276	680213	287134	289554	333856
	setup	0.063	0.061	0.721	0.923	1.129	4.380
	solve	0.005	0.006	0.046	0.033	0.031	0.039
	total	0.068	0.067	0.767	0.956	1.160	4.419

were generally competitive with the other preconditioners. For the matrices stiff4 and mass04, the CP–SAINV and CP–RIF preconditioners simply computed a diagonal approximation to BB^T, yet this proved effective. For the matrices mass05 and mass06, iteration counts for CP–SAINV and CP–RIF increased significantly over the exact constraint preconditioners, thereby increasing total solution times.

While MC64–ILUT solved all problems in Table I, total solution times were generally greater, especially for the matrices mass05 and mass06. The setup costs for the MC64–ILUT approach

were higher than for the other approaches; however, solve times were low due to the low iteration counts. For the matrix stiff4, MC64–ILUT was the clear winner among the iterative approaches.

The results in Table II are more varied. *Among the preconditioning approaches*, CP–EXT provided the fastest solution times for the problems capt12, plnt01, traj27, and traj33, but was slow for lnts09. For the larger matrices lnts09, traj27, and traj33, sufficient memory was not available to complete the MA49 factorizations and the formation of the preconditioner failed, indicated by the dagger symbol †. For each of these matrices, the resulting R for the QR factorization is dense. This results from a dense first column of B, in which case BB^T will be dense. Note also that the density of BB^T is independent of the ordering of the rows and columns of B. Dense columns and their impact on the normal equations and Schur complements are a well-known issue in linear programming; see [2, 43, 46], for example.

The approximate constraint preconditioners failed to converge within 2000 iterations for the optimal control matrices in Table II. We experimented with different drop tolerances, but found that only by allowing extremely dense approximations (essentially exact factorizations) could we obtain convergence. In order to gain further insight into the failure of these approaches, we examined $(BB^T)^{-1}$ for some smaller problems. When many entries of the inverse of a matrix are small in absolute value, such as when the magnitude of the entries decays away from the diagonal, the approaches based on incomplete A-orthogonalization can be effective. However, for the sparse optimal control matrices, in addition to having relatively dense (and in some cases completely dense) factors, the inverses showed little decay in the magnitude of entries. See [25, section 3.5] for details.

MC64–ILUT usually showed high setup costs, but also solved every problem in Table II. It should be noted that the setup times were primarily consumed in the ILUT factorization; nonsymmetric reordering times were small. For the traj27 and traj33 problems, the total solution times are high. For these matrices, MMD orderings took a long time. This was only true when computing MMD orderings for the entire matrix $\mathcal{K}$; ordering times for the $(2,1)$ blocks for the approximate constraint preconditioners were not abnormal.

The direct method was fastest for the matrices capt12, plnt01, lnts09, and traj27. In fact, the CP–EXT factorization of the constraint preconditioner for traj27 required more time than for the original matrix.

7. CONCLUSIONS

We considered approaches to approximating constraint preconditioners, in which we applied the preconditioner in factored form (3) by approximating the products $(BB^T)^{-1}$ and BB^T. These approximate constraint preconditioners were compared to two different applications of exact constraint preconditioning, to a more general preconditioning approach which ignores structure in the original system, and to a direct solver based on a symmetric indefinite factorization.

For the mass and stiffness matrices, the exact constraint preconditioner based on a QR factorization of the constraint matrix was effective. However, this same method failed for some of the optimal control matrices, where dense columns in the constraint matrix resulted in excessive storage requirements. In this case, a symmetric indefinite factorization of the constraint preconditioner was usually the most effective preconditioning approach. The

approximate constraint preconditioners were competitive for the mass and stiffness matrices, but failed for the optimal control matrices.

We also experimented with permutations and scalings that maximize the product of the entries along the diagonal of the matrix, in combination with a standard ILU preconditioner. This approach proved to be an effective technique, reaffirming the results detailed in [6]. In all cases, we constructed an incomplete LU factorization, and in most cases solved the systems efficiently. However, the one–sided permutations ignore, and in fact completely destroy, any structure present in the original system.

The direct method was fastest for more than half of the matrices considered in our experiments, but storage requirements were almost always higher for the direct method. For larger problems this could become more of an issue. On the other hand, for all the problems we considered, if multiple systems with the same coefficient matrix were to be solved, the direct method would be the fastest.

ACKNOWLEDGEMENTS

The authors would like to thank John Betts and Carsten Keller for providing many of the test matrices used in the experiments; Iain Duff for providing MA64 and MA49 routines; Michele Benzi and Miroslav Tůma for the routines used in the incomplete factorizations; and Ilse Ipsen, Benzi, Nick Gould and two anonymous referees for their insightful comments and suggestions.

REFERENCES

1. M. A. Ajiz and A. Jennings. A robust incomplete Choleski-conjugate gradient algorithm. *Internat. J. Numer. Methods Engrg.*, 20(5):949–966, 1984.
2. K. D. Andersen. A modified Schur-complement method for handling dense columns in interior-point methods for linear programming. *ACM Trans. Math. Software*, 22(3):348–356, 1996.
3. C. Ashcraft, R. G. Grimes, and J. G. Lewis. Accurate symmetric indefinite linear equation solvers. *SIAM J. Matrix Anal. Appl.*, 20(2):513–561 (electronic), 1999.
4. BCSLIB-EXT Sparse Matrix Package, The Boeing Company, 2002. http://www.boeing.com/phantom/bcslib-ext/.
5. M. Benzi, J. K. Cullum, and M. Tůma. Robust approximate inverse preconditioning for the conjugate gradient method. *SIAM J. Sci. Comput.*, 22(4):1318–1332 (electronic), 2000.
6. M. Benzi, J. C. Haws, and M. Tůma. Preconditioning highly indefinite and nonsymmetric matrices. *SIAM J. Sci. Comput.*, 22(4):1333–1353 (electronic), 2000.
7. M. Benzi, C. D. Meyer, and M. Tůma. A sparse approximate inverse preconditioner for the conjugate gradient method. *SIAM J. Sci. Comput.*, 17(5):1135–1149, 1996.
8. M. Benzi and M. Tůma. A sparse approximate inverse preconditioner for nonsymmetric linear systems. *SIAM J. Sci. Comput.*, 19(3):968–994 (electronic), 1998.
9. M. Benzi and M. Tůma. Orderings for factorized sparse approximate inverse preconditioners. *SIAM J. Sci. Comput.*, 21(5):1851–1868, Sept. 2000.
10. M. Benzi and M. Tůma. A robust factorization preconditioner for positive definite matrices. to appear in *Numerical Linear Algebra with Applications*, September 2001.
11. M. Benzi and M. Tůma. A robust preconditioner with low memory requirements for large sparse least squares problems. submitted to *SIAM Journal on Scientific Computing*, April 2002.
12. J. T. Betts. *Practical methods for optimal control using nonlinear programming.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001.
13. Å. Björck. *Numerical Methods for Least Squares Problems.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1996.
14. R. Bridson and W.-P. Tang. Ordering, anisotropy, and factored sparse approximate inverses. *SIAM J. Sci. Comput.*, 21(3):867–882, 1999.
15. E. Chow and Y. Saad. Experimental study of ILU preconditioners for indefinite matrices. *J. Comput. Appl. Math.*, 86(2):387–414, 1997.

16. I. S. Duff, A. M. Erisman, and J. K. Reid. *Direct Methods for Sparse Matrices*. Clarendon Press, Oxford, 1986.
17. I. S. Duff and J. Koster. The design and use of algorithms for permuting large entries to the diagonal of sparse matrices. *SIAM J. Matrix Anal. Appl.*, 20(4):889–901 (electronic), 1999. Sparse and structured matrices and their applications (Coeur d'Alene, ID, 1996).
18. I. S. Duff and J. Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. *SIAM J. Matrix Anal. Appl.*, 22(4):973–996 (electronic), 2001.
19. I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Software*, 9(3):302–325, 1983.
20. R. W. Freund and N. M. Nachtigal. Software for simplified Lanczos and QMR algorithms. *Appl. Numer. Math.*, 19(3):319–341, 1995. Special issue on iterative methods for linear equations (Atlanta, GA, 1994).
21. N. I. M. Gould. On practical conditions for the existence and uniqueness of solutions to the general equality quadratic programming problem. *Math. Programming*, 32(1):90–99, 1985.
22. N. I. M. Gould. Iterative methods for ill-conditioned linear systems from optimization. In G. Di Pillo and F. Giannessi, editors, *Nonlinear Optimization and Related Topics*, pages 123–142. Kluwer Publishing, 1999.
23. N. I. M. Gould, M. E. Hribar, and J. Nocedal. On the solution of equality constrained quadratic programming problems arising in optimization. *SIAM J. Sci. Comput.*, 23(4):1375–1394, 2001.
24. A. Greenbaum. *Iterative methods for solving linear systems*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1997.
25. J. C. Haws. *Preconditioning KKT Systems*. PhD thesis, North Carolina State University, Department of Mathematics, Raleigh NC, 2002.
26. L. Hemmingsson-Frändén and A. Wathen. A nearly optimal preconditioner for the Navier-Stokes equations. *Numer. Linear Algebra Appl.*, 8(4):229–243, 2001.
27. HSL (2002), a collection of Fortran codes for large scale scientific computation. http://www.cse.clrc.ac.uk/nag/hsl/.
28. D. James. *Conjugate Gradient Methods for Constrained Least Squares Problems*. PhD thesis, North Carolina State University, Department of Mathematics, 1990.
29. A. Jennings and M. A. Ajiz. Incomplete methods for solving $A^T Ax = b$. *SIAM J. Sci. Statist. Comput.*, 5(4):978–987, 1984.
30. C. Keller, N. I. M. Gould, and A. J. Wathen. Constraint preconditioning for indefinite linear systems. *SIAM J. Matrix Anal. Appl.*, 21(4):1300–1317 (electronic), 2000.
31. S. A. Kharchenko, L. Y. Kolotilina, A. A. Nikishin, and A. Y. Yeremin. A robust AINV-type method for constructing sparse approximate inverse preconditioners in factored form. *Numer. Linear Algebra Appl.*, 8(3):165–179, 2001.
32. L. Y. Kolotilina and A. Y. Yeremin. Factorized sparse approximate inverse preconditionings. I. Theory. *SIAM J. Matrix Anal. Appl.*, 14(1):45–58, 1993.
33. R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM J. Numer. Anal.*, 16(2):346–358, 1979.
34. J. W. H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Software*, 11(2):141–153, 1985.
35. L. Lukšan and J. Vlček. Indefinitely preconditioned inexact Newton method for large sparse equality constrained non-linear programming problems. *Numer. Linear Algebra Appl.*, 5(3):219–247, 1998.
36. M. Olschowka and A. Neumaier. A new pivoting strategy for Gaussian elimination. *Linear Algebra Appl.*, 240:131–151, 1996.
37. I. Perugia and V. Simoncini. An optimal indefinite preconditioner for a mixed finite element method. Technical Report 1098, IAN-CNR, November 1998.
38. I. Perugia and V. Simoncini. Block-diagonal and indefinite symmetric preconditioners for mixed finite element formulations. *Numer. Linear Algebra Appl.*, 7(7-8):585–616, 2000. Preconditioning techniques for large sparse matrix problems in industrial applications (Minneapolis, MN, 1999).
39. M. Rozložník and V. Simoncini. Short–term recurrences for indefinite preconditioning of saddle point problems. Technical Report 1181, IAN-CNR, February 2001.
40. Y. Saad. Preconditioning techniques for nonsymmetric and indefinite linear systems. *J. Comput. Appl. Math.*, 24(1-2):89–105, 1988. Iterative methods for the solution of linear systems.
41. Y. Saad. ILUT: a dual threshold incomplete *LU* factorization. *Numer. Linear Algebra Appl.*, 1(4):387–402, 1994.
42. Y. Saad. *Iterative methods for sparse linear systems*. PWS, Boston, MA, 1996.
43. M. A. Saunders. Major Cholesky would feel proud. *ORSA J. Computing*, 6(1):94–105, 1994.
44. SOCS Sparse Optimal Control Software, The Boeing Company, 2002. http://www.boeing.com/phantom/socs/.
45. H. A. van der Vorst. Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of

nonsymmetric linear systems. *SIAM J. Sci. Statist. Comput.*, 13(2):631–644, 1992.
46. R. J. Vanderbei. Splitting dense columns in sparse linear systems. *Linear Algebra Appl.*, 152:107–117, 1991. Interior point methods for linear programming.
47. X. Wang, K. A. Gallivan, and R. Bramley. CIMGS: an incomplete orthogonal factorization preconditioner. *SIAM J. Sci. Comput.*, 18(2):516–536, 1997.
48. R. C. Whaley and J. J. Dongarra. ATLAS: Automatically tuned linear algebra software.
49. J. Zhang. A multilevel dual reordering strategy for robust incomplete LU factorization of indefinite matrices. *SIAM J. Matrix Anal. Appl.*, 22(3):925–947 (electronic), 2000.
50. Z. Zlatev and H. B. Nielsen. Solving large and sparse linear least-squares problems by conjugate gradient algorithms. *Comput. Math. Appl.*, 15(3):185–202, 1988.