

LEAST SQUARES FOR DENSE OR SPARSE SEMI-DEFINITE MATRICES

WOLFGANG M. HARTMANN* AND ROBERT E. HARTWIG†

Abstract. Minimum norm least squares solutions are computed for a large dense singular positive semi-definite matrix. This is done by first tridiagonalizing the matrix with aid of the Aasen method, followed by the computation of the Moore-Penrose inverse of the triangular congruence map LTL^T , where L is lower triangular and T is tridiagonal. This method is shown to be stable in that it yields the correct rank. It is also considerably faster than the traditional use of the QR or SVD algorithms. It is further shown that this algorithm can be extended to yield minimum norm least squares solutions for very large sparse singular systems with small nullity.

Key words. least squares, dense, sparse, semi-definite matrices, Aasen.

AMS subject classifications. 15A09, 15A23, 15A57, 65F05, 65F20, 65F50

1. Introduction. In a recent paper [12], it was shown that the computation of minimum norm least squares solutions for a large dense singular positive semidefinite (p.s.d) matrix A , can be executed in two phases. In phase (I), the matrix A is tridiagonalized using a real orthogonal matrix θ , say:

$$\theta A \theta^T = T,$$

with T tridiagonal symmetric, while in phase (II), the Moore-Penrose (MP) inverse is efficiently computed for the tridiagonal matrix T .

It is well known that in general, neither similarity nor congruence preserve least squares solutions or Moore-Penrose inverses. That is, generally

$$(QAQ^{-1})^\dagger \neq QA^\dagger Q^{-1} \quad \text{and} \quad (RAR^T)^\dagger \neq R^{-T}A^\dagger R^{-1}.$$

As such any attempt to replace the orthogonal matrix with just a non-singular matrix, seems to invite a theoretical as well as a computational nightmare. It comes then somewhat as a surprise that in the special case where θ is the product of a lower triangular and a permutation matrix, we can get around this problem. Indeed, our aim is to compute A^\dagger for a large singular positive semidefinite matrix A , from the decomposition

$$QAQ^T = LTL^T = M,$$

where T is tridiagonal symmetric, Q is a permutation matrix and L is lower triangular and unital. For example, this may be obtained via the Aasen reduction of the symmetric matrix A [1]. This time however L need *not* be tri-sparse, as was the case in [12]!

We will find A^\dagger by computing the Moore Penrose (MP) inverse of the p.s.d. matrix $M = LTL^T = M^T$, and subsequently forming $A^\dagger = Q^T M^\dagger Q$.

Since our algorithm may be used for large scale dense applications in Statistics, it is important to know how it performs in its computation of $\mathbf{x} = A^\dagger \mathbf{b}$ i.e. the minimum norm least squares solution of a linear system $A\mathbf{x} = \mathbf{b}$, where A is dense,

*email saswmh@unx.sas.com SAS Institute Inc, SAS Campus Drive, Cary NC 27513 USA

†email hartwig@math.ncsu.edu North Carolina State University, Raleigh N.C. 27695-8205 USA

$n \times n$, positive semidefinite and singular. We shall compare the real CPU times of our two algorithms with three other algorithms currently being used for minimum norm least squares solutions of singular systems.

Throughout we shall assume familiarity with the basic results on g-inverses as given in [3].

2. An Expression for the Moore Penrose Inverse. Our key observation is that because M is real symmetric, its MP inverse *also* equals its group inverse $M^\#$ [3], which we may compute using the theory developed in [16]. Indeed, it was shown in [16], that

$$M^\dagger = M^\# = L\alpha T\beta L^T = L\alpha^2 T L^T,$$

where, $\alpha = TXT^- + I - TT^-$ and $\beta = T^-TX + I - T^-T$, $TT^-T = T$ and

$$(2.1) \quad T(L^T L)TX = T.$$

Again, T is symmetric tridiagonal. Now since $\alpha T = TXT^-T$ and $T\beta = TX$ we see that

$$\alpha T\beta = TXT^-TX \text{ and } \alpha^2 T = TXT^-TX T^-T.$$

Our main problem then is to compute T^-T and finds a solution matrix X to (1).

We can at once simplify M^\dagger to

$$M^\dagger = (LTL^T)^\dagger = L(TXT^-TX)L^T,$$

which we shall now examine.

We next recall from [12] that generally T will have a block diagonal form

$$T = \text{diag}(T_1, T_2, \dots, T_r),$$

where each T_i is $n_i \times n_i$ and can be factored as $T_i = P_i L_{i1} E_i L_{i1}^T P_i^T$. Here P_i is a permutation matrix, E_i is a positive diagonal matrix and $L_{i1} = \begin{bmatrix} K_i \\ C_i \end{bmatrix}$ is an $n_i \times t_i$ matrix. — Note that for notational convenience we replaced the matrices P_i obtained in [12] by their transposes— Factoring out the $t_i \times t_i$ factor K_i we arrive at $L_{i1} = B_i K_i$, where $B_i =$

$$\begin{bmatrix} I_{t_i} \\ -U_i \end{bmatrix} \text{ and } C_i = -U_i K_i. \text{ This gives } T_i = P_i B_i E_i' B_i^T P_i^T \text{ where } E_i' = K_i E_i K_i^T.$$

We note in passing that since B_i is of full column rank, it has a left inverse $B_i^- = [I, 0]$, which we shall use repeatedly. As such we have

$$(2.2) \quad T = P B E' B^T P^T,$$

where $P = \text{diag}(P_1, \dots, P_r)$, $B = \text{diag}(B_1, \dots, B_r)$ and $E' = \text{diag}(E'_1, \dots, E'_r)$.

Let us next compute an idempotent $F' = \text{diag}(F'_1, \dots, F'_r)$, such that $RS(F') = RS(T)$. This idempotent must be of the form T^-T for some T^- , and can be used in the above. Since $RS(T_i) = RS(B_i^T P_i^T)$ we see that $F_i = (B_i^T P_i^T)^-(B_i^T P_i^T)$, for which we may select

$$F_i = P_i \begin{bmatrix} I_{t_i} \\ 0 \end{bmatrix} [I, -U_i^T] P_i^T.$$

We now turn to equation (2.1), and substitute the expression for T from (2.2). This gives

$$PBE'B^T P^T (L^T L) PBE'B^T P^T X = PBE'B^T P^T.$$

In this we may cancel P and set $Y = P^T X P$ to yield $X = P Y P^T$ and

$$(2.3) \quad BE'WE'B^T Y = BE'B^T,$$

in which $E' = KEK^T$ and $W = (LPB)^T(LP B) = G^T G$, with $G = LPB$. We note that because L and P are invertible, and B has full column rank, W is *also* invertible.

We now may select Y as $Y = (B^T)^-(E')^{-1}W^{-1}B^T$ where $(B_i^T)^- = [I, 0]^T$. Hence $X = P(B^T)^-(E')^{-1}W^{-1}B^T P^T$ and consequently,

$$TX = PBE'B^T P^T P(B^T)^-(E')^{-1}W^{-1}B^T P^T = PBW^{-1}B^T P^T.$$

From this we see that $RS(TX) = RS(B^T P^T) = RS(T)$ and thus $TXT^{-T} = TX$. This in turn, means that $M^\dagger = LTX^2L^T$ in which

$TX^2 = PBW^{-1}B^T P^T P(B^T)^-(E')^{-1}W^{-1}B^T P^T = PBW^{-1}(E')^{-1}W^{-1}B^T P^T$. Since $LPBW^{-1} = G(G^T G)^{-1} = G^\dagger$, we consequently have

$$(2.4) \quad M^\dagger = (LTL^T)^\dagger = (G^T)^\dagger E'^{-1} G^\dagger,$$

in which again $G = LPB$, $P = \text{diag}(P_1, \dots, P_r)$, $K = \text{diag}(K_1, \dots, K_r)$, $E = \text{diag}(E_1, \dots, E_r)$, and $B = \text{diag}(B_1, \dots, B_r)$ with $B_i = [I_{t_i}, -U_i^T]^T$. We note that because G is of full column rank we could write this as $M^\dagger = (GE'G^T)^\dagger$. We *cannot* however, use the reverse order law to simplify the computation of W^{-1} . To get around this difficulty we shall need several preliminary results on Schur complements and MP inverses. In the numerical computation of W^{-1} , our aim is to make use of the *small* values of s_i as well as the triangular nature of L and K_i , in addition to the tri-sparsity of the C_i and K_i .

3. Schur Complements. Associated with the matrix $M = \begin{bmatrix} A & C \\ B & D \end{bmatrix}$, we may define the two principal Schur complements as

$$Z(M) = D - BA^+C \quad \text{and} \quad \zeta(M) = A - CD^+B,$$

where A^+ is any 1-2 inverse of A [3]. That is, any solution to $AXA = A$ and $XAX = X$. If we assume that

$$(3.1) \quad R(C) \subseteq R(A) \quad \text{and} \quad RS(B) \subseteq RS(A),$$

then Z is *invariant* under the choice of 1-2 inverse. In this case we can construct a 1-2 inverse for M of the form [13, page 212]:

$$M^+ = \begin{bmatrix} A^+ + A^+CZ^+BA^+ & -A^{-1}CZ^+ \\ -Z^+BA^+ & Z^+ \end{bmatrix} = \begin{bmatrix} \pi & \rho \\ q & \sigma \end{bmatrix}.$$

From this we automatically see that

$$(3.2) \quad RS(\rho) \subseteq RS(\sigma) \quad \text{and} \quad R(q) \subseteq R(\sigma).$$

As such we have the invariance of the Schur complement $\zeta(M^+) = \pi - \rho(\sigma)^+q$ under the choice of $(\cdot)^+$. We may thus take $\sigma^+ = (Z^+)^+ = Z$. For this choice of 1-2 inverse we get:

THEOREM 3.1. Given that (3.1) holds, there exists a reflexive inverse M^+ for M for which the trailing Schur complement satisfies

$$(3.3) \quad \zeta(M^+) = A^+.$$

This formula should be contrasted with the Haynsworth Quotient Formula [13, pg 214]. Moreover, if equation (3.2) holds then by symmetry $\zeta(M) = \pi^+$.

In the special case where A is invertible (3.1) automatically holds and M will be invertible exactly when $Z(M)$ is invertible. In which case

$$(3.4) \quad M^+ = \begin{bmatrix} A^{-1} + A^{-1}CZ^{-1}BA^{-1} & -A^{-1}CZ^{-1} \\ -Z^{-1}BA^{-1} & Z^{-1} \end{bmatrix} = \begin{bmatrix} \pi & \rho \\ q & \sigma \end{bmatrix},$$

and $\zeta(M^{-1}) = A^{-1}$.

Let us now turn to the applications of this crucial fact.

4. Applications of Schur Complements. Consider the invertible matrix PAQ , where A is $n \times n$ invertible and P^T and Q are $n \times k$ and necessarily of full column rank. We now have

THEOREM 4.1. Suppose that $\begin{bmatrix} P \\ Q \end{bmatrix} [T, U] = I$ and $[Q, S] \begin{bmatrix} V \\ W \end{bmatrix} = I$. Then

$$(4.1) \quad (PAQ)^{-1} = VA^{-1}T - VA^{-1}U(WA^{-1}U)^{-1}WA^{-1}T$$

Proof: Let $M = \begin{bmatrix} P \\ Q \end{bmatrix} A [Q, S] = \begin{bmatrix} PAQ & PAS \\ RAQ & RAS \end{bmatrix}$ with inverse

$$M^{-1} = \begin{bmatrix} V \\ W \end{bmatrix} A^{-1} [T, U] = \begin{bmatrix} VA^{-1}T & VA^{-1}U \\ WA^{-1}T & WA^{-1}U \end{bmatrix}.$$

Since PAQ is invertible we know that (3.1) holds and that the invertibility of M and A guarantee that of $Z(M)$. We may thus apply Theorem 1 to give

$$A^{-1} = \pi - \rho\sigma^{-1}q = VA^{-1}T - VA^{-1}U(WA^{-1}U)^{-1}WA^{-1}T.$$

as desired.

Remark The matrices $A^{-1}U(WA^{-1}U)^{-1}W$ and $U(WA^{-1}U)^{-1}WA^{-1}$ are both idempotent. Let us next use Theorem 2 to compute the Moore Penrose inverse of the product AB, where A is invertible and B is of full column rank. This has been an unsolved problem for a long time.

THEOREM 4.2. If A is invertible and B had full column rank, then

$$(4.2) \quad (AB)^\dagger = DA^{-1}(I - YY^\dagger) \quad \text{and} \quad (AB)(AB)^\dagger = I - YY^\dagger,$$

where $[B, C] \begin{bmatrix} D \\ F \end{bmatrix} = I$ and $Y = A^{-T}F^T$. *Proof:* Since AB has full column rank, $(AB)^\dagger = (B^T A^T AB)^{-1} B^T A^T$, and we may apply theorem 2 with $P = B^T$ and $Q = B$. Again letting $[B, C] \begin{bmatrix} D \\ F \end{bmatrix} = I$, we arrive at $M = \begin{bmatrix} B^T \\ C^T \end{bmatrix} A^T A [B, C]$ and

$M^{-1} = \begin{bmatrix} D \\ F \end{bmatrix} A^{-1} A^{-T} [D^T, F^T]$. By Theorem 2,

$$(B^T A^T AB)^{-1} = DA^{-1} A^{-T} D^T - DA^{-1} A^{-T} F^T (FA^{-1} A^{-T} F^T)^{-1} FA^{-1} A^{-T} D^T,$$

where $Y = A^{-T} F^T$. This reduces to $DA^{-1}(I - YY^\dagger)A^{-T} D^T$. We then have

$$(AB)^\dagger = (B^T A^T AB)^{-1} B^T A^T = DA^{-1}(I - YY^\dagger)A^{-T} (D^T B^T) A^T.$$

Now $BD + CF = I$ and so $D^T B^T = I - F^T C^T$. Consequently, $A^{-T} D^T B^T = A^{-T}(I - F^T C^T) = A^{-T} - Y C^T$. Substituting this in the above now gives

$$(AB)^\dagger = DA^{-1}(I - YY^\dagger)(A^{-T} - Y C^T) A^T = DA^{-1}(I - YY^\dagger),$$

as desired.

Lastly we form

$$\begin{aligned} AB(AB)^\dagger &= A(BD)A^{-1}(I - YY^\dagger) = A(I - CE)A^{-1}(I - YY^\dagger) \\ &= (I - ACFA^{-1})(I - YY^\dagger) = I - YY^\dagger. \end{aligned}$$

We note in passing that when $[B, C] \begin{bmatrix} D \\ F \end{bmatrix} = I$ we also know that $R(I - BB^\dagger) = R(E^T)$ or equivalently $I - BB^\dagger = E^\dagger E$. As such we can replace D by *any* matrix D' for which $D'B = I$, such as B^\dagger .

In the special case where we select C such that $B^T C = 0$, we may take $D = B^\dagger$ and $F = C^\dagger$. In this case (4.2) reduces to

$$(AB)^\dagger = B^\dagger A^{-1}(I - YY^\dagger),$$

where $Y = A^{-T}(C^T)^\dagger$. Now because $R(C) = R((C^T)^\dagger)$ we may replace Y by $Y' = A^{-T} C$. Thus we also have

$$(4.3) \quad (AB)^\dagger = B^\dagger A^{-1}[I - Y'(Y')^\dagger]$$

Let us now return to equation (2.4).

5. Numerical Computation of $G^\dagger = (LPB)^\dagger$. We recall that we have to compute the least squares solution $A^\dagger \mathbf{b} = Q^T (G^T)^\dagger (E')^{-1} G^\dagger Q \mathbf{b}$. Since Q is a permutation matrix, the first product $Q \mathbf{b} = \mathbf{c}$ is trivial, and we shall concentrate on computing $G^\dagger \mathbf{c}$, where $G = LPB$ which is given as in section 2. Again if we complete

B to a non-singular matrix $[B, C]$ and suppose that $[B, C] \begin{bmatrix} D \\ F \end{bmatrix} = I$, then

$$G^\dagger = DP^T L^{-1}(I - YY^\dagger),$$

where $Y = L^{-T} P F^T$. Let us consider the special case where $B = \text{diag}(B_1, B_2, B_3)$ and $r = 3$. That is, suppose we are given the $(n_1 + n_2 + n_3) \times (t_1 + t_2 + t_3)$ matrix

$$B = \begin{bmatrix} I_{t_1} & 0 & 0 \\ -U_1 & 0 & 0 \\ 0 & I_{t_2} & 0 \\ 0 & -U_2 & 0 \\ 0 & 0 & I_{t_3} \\ 0 & 0 & -U_3 \end{bmatrix}.$$

There are *several* ways of completing it to an invertible matrix $[B, C]$. Needless to say we have to select C, D and F all simultaneously.

Selection 1: We complete B to the matrix:

$$(5.1) \quad [B, C] = \begin{bmatrix} I_{t_1} & 0 & 0 & 0 & 0 & 0 \\ -U_1 & 0 & 0 & I_{s_1} & 0 & 0 \\ 0 & I_{t_2} & 0 & 0 & 0 & 0 \\ 0 & -U_2 & 0 & 0 & I_{s_2} & 0 \\ 0 & 0 & I_{t_3} & 0 & 0 & 0 \\ 0 & 0 & -U_3 & 0 & 0 & I_{t_3} \end{bmatrix},$$

where C is $(n_1 + n_2 + n_3) \times (s_1 + s_2 + s_3)$. Its inverse has the form

$$(5.2) \quad [B, C]^{-1} = \begin{bmatrix} I_{t_1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & I_{t_2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{t_3} & 0 \\ U_1 & I_{s_1} & 0 & 0 & 0 & 0 \\ 0 & 0 & U_2 & I_{s_2} & 0 & 0 \\ 0 & 0 & 0 & 0 & U_3 & I_{s_3} \end{bmatrix} = \begin{bmatrix} D \\ F \end{bmatrix}.$$

With this choice of C and D we have

$$G^\dagger = \begin{bmatrix} I_{t_1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & I_{t_2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_{t_3} & 0 \end{bmatrix} P^T L^{-1} (I - YY^\dagger),$$

where

$$(5.3) \quad Y = L^{-T} P \begin{bmatrix} U_1^T & 0 & 0 \\ I_{s_1} & 0 & 0 \\ 0 & U_2^T & 0 \\ 0 & I_{s_2} & 0 \\ 0 & 0 & U_3^T \\ 0 & 0 & I_{s_3} \end{bmatrix}$$

$$\textit{Selection 2:} \text{ As an alternative selection we could pick } C = \begin{bmatrix} U_1^T & 0 & 0 \\ I_{s_1} & 0 & 0 \\ 0 & U_2^T & 0 \\ 0 & I_{s_2} & 0 \\ 0 & 0 & U_3^T \\ 0 & 0 & I_{s_3} \end{bmatrix}.$$

Since now $C^T B = 0$ and $[B, C]^{-1} = \begin{bmatrix} B^\dagger \\ C^\dagger \end{bmatrix}$, we would be selecting $D = B^\dagger$ and $F = C^\dagger$. This gives $G^\dagger = B^\dagger P^T L^{-1} (I - YY^\dagger)$, with $Y = L^{-T} P (C^T)^\dagger$. Now since $R(Y) = R(Y')$ where $Y' = L^{-T} P C$ we see that $YY^\dagger = Y'(Y')^\dagger$, which yields the same matrix as found in (5.3).

Since the matrix D is of much simpler form than B^\dagger , we proceed by making the first selection which requires the computation of $G^\dagger = D P^T L^{-1} (I - YY^\dagger)$.

Step 1 We start with the computation of $Y = L^{-T} P F^T$ and assume that we have computed the sparse matrices $U_i = -C_i (K_i)^{-1}$ in the first phase and have stored

them for later use.

Now from (5.2) we obtain the matrix F^T which we permute to give the matrix

$$PF^T = P \begin{bmatrix} U_1^T & 0 & 0 \\ I_{s_1} & 0 & 0 \\ 0 & U_2^T & 0 \\ 0 & I_{s_2} & 0 \\ 0 & 0 & U_3^T \\ 0 & 0 & I_{s_3} \end{bmatrix} = \begin{bmatrix} V_1 & 0 & 0 \\ 0 & V_2 & 0 \\ 0 & 0 & V_3 \end{bmatrix}.$$

This only uses, in succession, n_i swaps of $1 \times s_i$ row vectors, for $i = 1, 2, 3$.

Step 2 We next back-solve the system $L^T Y = PF^T$ for Y , which takes $O(n^2 s/2)$ operations.

Step 3 We compute the orthogonal projection of \mathbf{c} onto the range of Y . That is, we compute $YY^\dagger \mathbf{c}$. This is one of the most fundamental computations in *ALL* of applied mathematics and statistics!

We shall now pursue this by making use of the crucial facts that the s_i are generally small and that the Y matrix is “skinny” block triangular. Indeed,

$$Y = \begin{bmatrix} L_1^{-T} & * & * \\ 0 & L_2^{-T} & * \\ 0 & 0 & L_3^{-T} \end{bmatrix} \begin{bmatrix} V_1 & 0 & 0 \\ 0 & V_2 & 0 \\ 0 & 0 & V_3 \end{bmatrix} = \begin{bmatrix} Y_1 & * & * \\ 0 & Y_2 & * \\ 0 & 0 & Y_3 \end{bmatrix}$$

As such we shall use Householder transformations to compute the QR factorization of Y , taking advantage of its “skinny” block triangular structure. Subsequently we compute YY^\dagger and $YY^\dagger \mathbf{c}$.

Consider the n by s matrix $Y = \begin{bmatrix} Y_1 & K & L \\ 0 & Y_2 & M \\ 0 & 0 & Y_3 \end{bmatrix}$, where $n = n_1 + n_2 + n_3$ and

$s = s_1 + s_2 + s_3$ and Y_i is $n_i \times s_i$.

We first use Householder transformations to construct an $n_1 \times n_1$, matrix Q_1 , such that $Q_1 Y_1 = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$, where R_1 is $s_1 \times s_1$ and upper triangular. Also let $Q_1 [Y_1, K, L] = \begin{bmatrix} R_1 & * & * \\ 0 & K' & L' \end{bmatrix}$, where $[K', L']$ is $(n_1 - s_1) \times (s_2 + s_3)$. If we apply $\text{diag}(Q_1, I_{n_1+n_2})$ to Y then we arrive at the matrix

$$\begin{bmatrix} R_1 & * & * \\ 0 & K' & L' \\ 0 & Y_2 & M \\ 0 & 0 & Y_3 \end{bmatrix}.$$

For convenience we now set $w_1 = n_1, w_2 = n_1 + n_2 - s_1$ and $w_3 = n - s_1 - s_2$.

In the second stage we use Householder maps to obtain the QR decomposition of the $w_2 \times s_2$ matrix $\begin{bmatrix} K' \\ Y_2 \end{bmatrix}$, say $Q_2 \begin{bmatrix} K' \\ Y_2 \end{bmatrix} = \begin{bmatrix} R_2 \\ 0 \end{bmatrix}$, where R_2 is $s_2 \times s_2$ and Q_2 is of size $w_2 \times w_2$.

Further suppose that

$$Q_2 \begin{bmatrix} K' & L' \\ Y_2 & M \end{bmatrix} = \begin{bmatrix} R_2 & * \\ 0 & M' \end{bmatrix}.$$

Then apply

$$\begin{bmatrix} I_{s_1} & 0 & 0 \\ 0 & Q_2 & 0 \\ 0 & 0 & I_{n_3} \end{bmatrix} \begin{bmatrix} R_1 & * & * \\ 0 & K' & L' \\ 0 & Y_2 & M \\ 0 & 0 & Y_3 \end{bmatrix} = \begin{bmatrix} R_1 & * & * \\ 0 & R_2 & * \\ 0 & 0 & M' \\ 0 & 0 & Y_3 \end{bmatrix}.$$

In the last stage we compute the QR factorization of the matrix $\begin{bmatrix} M' \\ Y_3 \end{bmatrix}$ which is $w_3 \times s_3$. Say $Q_3 \begin{bmatrix} M' \\ Y_3 \end{bmatrix} = \begin{bmatrix} R_3 \\ 0 \end{bmatrix}$, where Q_3 is of size w_3 and R_3 is $s_3 \times s_3$. . Combining these three steps, we obtain:

$$\begin{bmatrix} I_{s_1} & & & \\ & I_{s_2} & & \\ & & & Q_3 \end{bmatrix} \begin{bmatrix} I_{s_1} & & & \\ & Q_2 & & \\ & & & I_{n_3} \end{bmatrix} \begin{bmatrix} Q_1 & & & \\ & I_{n_1+n_2} & & \\ & & & \\ & & & \end{bmatrix} Y = \begin{bmatrix} R_1 & * & * \\ 0 & R_2 & * \\ 0 & 0 & R_3 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} T \\ 0 \end{bmatrix},$$

where T is $s \times s$ upper triangular.

In conclusion let us now consider in more detail the $w_i \times w_i$ orthogonal matrices Q_i .

We know that $Q_1 = H_{s_1} \dots H_1$ where $H_i = \begin{bmatrix} I_{i-1} & 0 \\ 0 & \pi_i \end{bmatrix}$ $i = 1, \dots, s_1$. In this the π_i are Householder transformations of size $w_1 - i + 1$ and are determined (and stored) by the $(w_1 - i + 1) \times 1$ vectors \mathbf{v}_i . Likewise $Q_2 = H'_{s_2} \dots H'_1$, where $H'_i = \begin{bmatrix} I_{i-1} & 0 \\ 0 & (\pi_i)'\end{bmatrix}$ $i = 1, \dots, s_2$. In this the $(\pi_i)'$ are Householder transformations of size $w_2 - i + 1$ determined by the $(w_2 - i + 1) \times 1$ vectors \mathbf{v}'_i . Lastly, $Q_3 = H''_{s_3} \dots H''_1$. In this the $(\pi_i)''$ are Householder transformations of size $w_3 - i + 1$ and are determined (stored) by the $(w_3 - i + 1) \times 1$ vectors \mathbf{v}''_i . That is, $H''_i = \begin{bmatrix} I_{i-1} & 0 \\ 0 & (\pi_i)''\end{bmatrix}$.

As always we store the vectors $\mathbf{v}_i, \mathbf{v}'_i$ and \mathbf{v}''_i as we compute them. Let us next define

$$\Gamma_i = \text{diag}(H_i, I_{n_1+n_2}) = \text{diag}(I_{i-1}, \pi_i, I_{n_1+n_2}), \quad i = 1, \dots, s_1,$$

as well as

$$\Delta_i = \text{diag}(I_{s_2}, H'_i, I_{n_3}) = \text{diag}(I_{s_2}, I_{i-1}, (\pi_i)', I_{n_3}), \quad i = 1, \dots, s_2,$$

in addition to:

$$\rho_i = \text{diag}(I_{s_1}, I_{s_2}, H''_i) = \text{diag}(I_{s_1}, I_{s_2}, I_{i-1}, (\pi_i)''), \quad i = 1, \dots, s_3.$$

We thus arrive at the QR factorization of Y in the product form

$$(\rho_{s_3} \dots \rho_1)(\Delta_{s_2} \dots \Delta_1)(\Gamma_{s_1} \dots \Gamma_1)Y = \begin{bmatrix} T \\ 0 \end{bmatrix},$$

where T is upper triangular. We then compute the product

$$YY^\dagger \mathbf{c} = (\Gamma_1 \dots \Gamma_{s_1})^\dagger (\Delta_1 \dots \Delta_{s_2}) (\rho_1 \dots \rho_{s_3}) \begin{bmatrix} I_s & 0 \\ 0 & 0 \end{bmatrix} (\rho_{s_3} \dots \rho_1) (\Delta_{s_2} \dots \Delta_1) (\Gamma_{s_1} \dots \Gamma_1) \mathbf{c}.$$

Needless to say, we could compute the matrices $(\rho_{s_3} \dots \rho_1) (\Delta_{s_2} \dots \Delta_1) (\Gamma_{s_1} \dots \Gamma_1) \mathbf{c}$ as we go, but we would still have to keep the associated vectors in storage.

6. Computational Order. To facilitate an overview of our algorithm we now list the computational steps that have to be taken to obtain $A^\dagger \mathbf{b}$. To obtain the product $Q^T (G^T)^\dagger (E')^{-1} G^\dagger Q \mathbf{b}$ where $G^\dagger = DP^T L^{-1} (I - YY^\dagger)$, we perform the following sequence of steps.

1. permute $Q \mathbf{b} = \mathbf{c}$
2. compute $(I - YY^\dagger) \mathbf{c} = \mathbf{d}$
3. forward-solve $L \mathbf{e} = \mathbf{d}$ for \mathbf{e}
4. permute $P^T \mathbf{e} = \mathbf{f}$
5. pick out the elements according to $D \mathbf{f} = \mathbf{g}$
6. sparse solve $K \mathbf{h} = \mathbf{g}$ for \mathbf{h} i.e. solve $K \mathbf{h}_i = \mathbf{g}_i$ and use the sparsity of the K_i
7. form the diagonal product $E^{-1} \mathbf{h} = \mathbf{m}$ or $(E_i)^{-1} \mathbf{h}_i = \mathbf{m}_i$
8. sparse back-solve $K^T \mathbf{n} = \mathbf{m}$ for \mathbf{n} .
9. pick out the pieces according to $D^T \mathbf{n} = \mathbf{q}$
10. swap $P \mathbf{q} = \mathbf{r}$
11. back-solve $L^T \mathbf{s} = \mathbf{r}$ to yield \mathbf{s}
12. form $(I - YY^\dagger) \mathbf{s} = \mathbf{t}$
13. permute $Q^T \mathbf{t} = \mathbf{u}$.

7. Performance of Dense and Sparse Solvers.

7.1. Solving Systems with Dense PSD Matrices. We next give a table contrasting the running times of the present implementation (called Aasen), as compared to our previous algorithm PSD3, as well as the three other algorithms (COS, SVS and EVS) currently being used in the computation, of least squares solutions. We begin with a short description of each of these algorithms.

Aasen : Aasen Tridiagonalizing and Tridiagonal PSD Solver:

This is the algorithm described above. The nonsingular version can compactly be described as:

1. Tridiagonalize A using Aasen's algorithm ([1] and [10] p. 162). This uses $n^3/6$ flops, like the Cholesky factorization.
2. Forward solve a lower triangular system, with a pivoted right hand side. This requires $O(n^2)$ flops.
3. Use our algorithm in [11] to obtain the solution of the tridiagonal problem. Together, decomposition and solution uses only $O(n)$.
4. Backward solve an upper triangular system, which requires $O(n^2)$ flops.

In the nonsingular case and for sufficiently large n this algorithm approaches the speed of the Cholesky factorization. In the singular case the algorithm is slightly modified to:

1. Tridiagonalize A using Aasen's algorithm ([1] and [10] p. 162).
2. Decompose the positive semi-definite tridiagonal matrix ([12] with Frane pivoting) and obtain the nullity $d(A)$.
3. Allocate the memory necessary for the range space computation $Y \in \mathcal{R}^{n \times d(A)}$.

4. Perform the multi-stage solution process as described above.

When the nullity is not too large, and n is sufficiently large, the speed of this algorithm theoretically approaches that of the Cholesky solution process.

PSD3 : Orthogonal Tridiagonalizing and Tridiagonal PSD Solver:

1. Tridiagonalize A as in [10], p. 419, using $n - 1$ symmetric Householder transformations without explicitly computing P . This uses $2n^3/3$ flops.
2. Decompose the positive semi-definite tridiagonal matrix with Frane pivoting and obtain the nullity $d(A)$. This takes $O(n)$ flops.
3. If the tridiagonal matrix consists of submatrices having a nullity larger than one, we must allocate the memory necessary for the corresponding trispase solution.
4. Apply the sequence of Householder maps to each right-hand column, requiring n^2 flops.
5. Use the trispase algorithm to solve the tridiagonal problem. This takes $O(n)$ flops for unit nullities.
6. Again apply the Householder transformations now on the solution of the tridiagonal problem, requiring n^2 flops.

The influence of rank deficiencies on the flop count depends on the size of the matrix split offs and is not easily predicted.

COS : Complete Orthogonal Solver: The algorithm is based on [10], p.220 and 236, and implemented as in [11]:

1. Compute the complete orthogonal decomposition

$$A = Q \begin{bmatrix} L^T & 0 \\ 0 & 0 \end{bmatrix} P^T$$

$= Y[L^T \ 0]P^T$ with nonsingular upper triangular $L^T \in \mathcal{R}^{r \times r}$ and orthogonal $Q = [Y \ Z]$ and P . The orthogonal matrices Q and P are not explicitly computed. This takes $2n^2r - nr^2 - r^3/3$ flops (see [11], appendix 1).

2. After or while performing the first step of the complete orthogonal decomposition we apply the r Householder transformations in Q to the right-hand column \mathbf{b} forming $\mathbf{c} = Q^T \mathbf{b}$. This takes rn flops.
3. The least-squares solution

$$\mathbf{x} = P \begin{bmatrix} L^{-T} & 0 \\ 0 & 0 \end{bmatrix} Q^T \mathbf{b}$$

is completed by solving $(L^T)^{-1} \mathbf{c}$ via backward substitution and applying the r Householder transformations in P which takes together $r^2/2 + r(n - r)$ flops.

SVS : Singular Value Solver: Our implementation is a C version of the LINPACK subroutine DSVDC ([5] chapter 11). The svd for symmetric A is $A = UDV^T = VDV^T$ and we only need to compute V and D :

1. Compute V and D , requiring $6n^3$ flops ([10] p.248).
2. Compute $\mathbf{c} = V^T \mathbf{b}$. This takes nr flops for each right-hand column.
3. Compute $\mathbf{x} = VD^\dagger \mathbf{c}$, using nr flops for each right-hand column.

EVS : Eigen Value Solver: Our implementation is a C version of the EISPACK subroutines TRED2 and IMTQL2 ([8] pp.308 and pp.291).

1. Apply the symmetric QR algorithm to yield $A = QDQ^T$, where Q is orthogonal and D is diagonal. Since Q must be accumulated, this uses about $5n^3$ flops ([10] p.424).

2. Compute $\mathbf{c} = Q^T \mathbf{b}$, taking nr flops for each right-hand column using only the eigenvectors corresponding to nonzero eigenvalues.
3. Compute $\mathbf{x} = QD^\dagger \mathbf{c}$, requiring nr flops for each right-hand column.

There is a major difference in the memory need of the first two algorithms: Our Aasen based algorithm uses the $n \times d$ matrix Y , whereas our second PSD3 algorithm uses a $n_{max} \times n_{max}$ allocation where n_{max} is the size of the largest submatrix with nullity larger than one. For applications with a large nullity, say, more than $n/2$, the second algorithm seems to have an advantage. We construct randomly generated dense psd matrices A with $r(A) = n, .9n, .8n$ using the following approach:

1. The n elements of an $n \times n$ diagonal matrix \bar{D} are randomly generated (uniformly distributed in $[0,10]$). An orthogonal $n \times n$ matrix V is found by random generation (uniformly distributed in $[0,1]$) and columnwise orthogonalization by Gram-Schmidt.
2. After sorting the diagonal entries of \bar{D} in decreasing order, $d = n - r$ entries in equally distant locations are set to zero defining the diagonal matrix D .
3. Then matrix A is computed by $A = V^T D V$.

Table 1 shows the CPU time in seconds on a Pentium Pro 200 PC when only one right hand column b was used. As expected, our two methods become slower in the singular cases, due to

Aasen : the computation of the range space matrix Y

PSD3 : the solution of linear least-squares problems for submatrices A_σ with a nullity larger than one

Therefore, in general, our algorithms unlike SVS and EVS do not necessarily become faster when the tridiagonal matrix splits into smaller blocks.

n	d(A)	Aasen	PSD3	COS	SVS	EVS
100	0	0.0160	.0780	.0940	.5470	.6250
100	10	0.0320	.0940	.0940	.4840	.5000
100	20	0.0310	.0930	.1090	.4370	.4530
300	0	0.5160	2.8750	3.4220	25.078	10.938
300	30	0.7190	3.0000	3.5310	22.968	10.047
300	60	0.9380	2.9060	3.6880	20.640	9.1570
500	0	3.0780	13.812	17.859	120.859	54.312
500	50	4.0930	8.9370	18.140	111.093	50.078
500	100	5.3130	13.969	19.125	102.000	45.891
800	0	12.672	37.860	84.781	483.250	235.812
800	80	17.047	38.797	89.328	446.125	219.609
800	160	22.906	38.578	91.906	202.797	410.718
1000	0	24.468	74.953	143.797	921.797	441.141
1000	100	32.969	76.844	147.187	826.360	404.657
1000	200	43.219	76.094	158.234	764.031	372.984

7.2. Solving Systems with Sparse PSD Matrices. The extension of our algorithm to sparse matrices is a little delicate but can be pushed through. Our preliminary sparse version is only intended for illustrative purposes and can undoubtedly be improved further. For our purpose the following algorithm behaved (surprisingly) well:

1. As a preprocessor we performed optional Tarjan's or Sloan's algorithm to reduce the profile of the symmetric matrix A (see [6]).
2. We modified the pivot selection in Aasen's method to deal with the sparse dilemma, which requires us to find a compromise between numerical stability

(choosing a sufficiently large pivot) and keeping a small profile (small nonzero fillin).

3. Partial pivoting is combined with a one-step look-ahead Boolean algorithm which (sometimes over-) estimates the nonzero fillin for a small number of selections, of sufficiently large pivots.
4. The range space computations are still based on the dense algorithm (Householder). The memory for this step clearly restricts the size of the nullity that is permitted in practical applications. A sparse modification of this step would extend the applicability of the entire sparse algorithm.

As it turned out in practice, the partial pivoting can sometimes generate tridiagonal matrices with quite large (column) norms. Fortunately, the Frane (see [7]) pivoting method in our tri-sparse algorithm behaved very well *even* under those extreme circumstances. On the other hand, the norm of the tridiagonal matrix is easily controlled by shifting the partial pivoting criterion more toward numerical stability and away from the amount of nonzero fillin, with the associated loss of speed.

The set of matrices used in our experiments were assembled using the following two types of basic matrices:

1. Full rank 5-diagonal positive definite diagonal dominant bandmatrices
2. Arrowhead matrices of nullity one with 9-diagonal spikes.

That means the number of matrices with unit nullity equals the nullity of the whole matrix. We also applied random symmetric row and column permutations on the matrix, which however were (almost) completely resolved by (Tarjan's) Sloan's algorithm. The following table shows the computer time in seconds measured on a Pentium 200 running windows NT. We also include the number of nonzero entries on or below the diagonal, in the original symmetric matrix A and the number of nonzero entries in the lower triangular factor L . The computer time does not include the preprocessing time needed for the Tarjan or Sloan algorithm.

n	d(A)	Time in sec	Nonzeros in A	Nonzeros in L
2500	0	7.09	7425	5554
2500	10	7.14	7365	5523
5000	0	28.31	14925	11081
5000	10	28.41	14865	11051
10000	0	125.97	29850	22152
10000	10	126.86	29790	22126
20000	0	608.53	59850	43634
20000	10	607.45	59790	43613
30000	0	1439.08	89850	64862
30000	10	1438.69	89790	64833

It is interesting to note that in all applications the number of nonzero entries in L is smaller than than the number of nonzero entries in A . The reason for this is the almost completely 5-banded diagonal structure of the test matrices A . Therefore a large part of nonzeros in A is caught in the tridiagonal matrix T .

Remarks

1. When we have many right hand columns, it may be faster to *first* compute the complete MP inverse A^\dagger starting from the middle with $K^{-T}E^{-1}K^{-1}$, ... and make use of the symmetry to cut down the operation by a factor of two. The critical number of columns for this has to be computed.
2. When we set $L = I$ we do get back the results of [12]
3. In [12] the most demanding calculation that we had to do was the *efficient* inversion of matrices of the form $I + U^T U$. Here we have replaced this by the

Schur inversion of (3.1). The new feature in this paper is the intertwining of the triangular matrix L with the permutation matrix P and the full rank matrices B_i .

4. For large values of n , say 1000 and above, the difference between the quadratic and cubic term becomes significant, and our method becomes much faster than the SVD approach, which is chained to the $O(n^3)$ operation count.
5. Aasen's method is extremely thrifty where storage is concerned, since we only have to read in half the matrix. Indeed, it takes $n(n+1)/2$ floating point values for the storage of the symmetric (semidefinite) matrix and the dense algorithm can be performed 'in-core'. This means, the triangular factor may be stored in the same storage area as the symmetric matrix, since it is computed column by column. All orthogonal methods take more memory. For example, the complete orthogonal decomposition (the method that competes best with our algorithm in speed and memory need) $A = Q[R, 0]V^T$ operates by moving the symmetric matrix into a dense $n \times n$ storage location, where it operates in-core. This requires that the following three matrices have to be stored in this area: (a) the upper triangular matrix R , (b) the full rank left Householder transformations Q for the pivoted QR algorithm and (c) the right Householder transformations V with nullity $n - \text{rank}$.

REFERENCES

- [1] J.O. Aasen (1971), "On the reduction of a symmetric matrix to tridiagonal form", *BIT*, **11**, 233-242.
- [2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen (1991), *LAPACK User's Guide*, Philadelphia: SIAM Publications.
- [3] A. Ben-Israel and T.N. Greville (1974), *Generalized Inverses: Theory and Applications*, New York: John Wiley & Sons, Inc., p. 290-297.
- [4] J.R. Bunch and L. Kaufman (1977), "Some Stable Methods for Calculating Inertia and Solving Symmetric Linear Systems", *Mathematics of Computation*, **31**, 163-179.
- [5] J.J. Dongarra, J.R. Bunch, C.B. Moler, and G.W. Stewart (1979), *LINPACK User's Guide*, Philadelphia: SIAM Publications.
- [6] Duff, I.S. and Reid, J.K. (1978), "Algorithm 529: Permutations to Block Triangular Form"; *ACM Transactions on Math. Software*, **4**, p.189-192.
- [7] J.W. Frane (1977), "A Note on Checking Tolerance in Matrix Inversion", *Technometrics*, **19**, no. 4, pp. 513-514.
- [8] B.S. Garbow, J.M. Boyle, J.J. Dongarra, and C.B. Moler (1977), *Matrix Eigensystem Routines - EISPACK Guide Extension*, Berlin: Springer Verlag.
- [9] A. George, J.R. Gilbert, and J.W.H. Liu (1993), *Graph Theory and Sparse Computations*, New York: Springer.
- [10] G.H. Golub and C.F. Van Loan (1989), *Matrix Computations*, 2nd Ed., Baltimore: J. Hopkins University Press.
- [11] W.M. Hartmann and R.E. Hartwig (1996), "Computing the Moore-Penrose Inverse for the Covariance Matrix in Constrained Nonlinear Estimation", *SIAM Journal on Optimization*, **6**, p. 727-747.
- [12] W. M. Hartmann and R. E. Hartwig (1997), "Tri diagonal $L - D - L^T$ factorization with pivoting", submitted for publication to SIMAX.
- [13] R.E. Hartwig (1976), "Block Generalized Inverses", *Arch. Ratl. Mech. Anal.*, **61**, p. 197-251.
- [14] M.T. Jones and M.L. Patrick (1993), "Bunch-Kaufman Factorization for Real Symmetric Indefinite Banded Matrices", *SIAM Journal Matrix Anal. Appl.*, **14**, 553-559.

- [15] B.N. Parlett (1980), *The Symmetric Eigenvalue Problem*, Englewood Cliffs, NJ.: Prentice Hall.
- [16] R. Puystjens and R. E. Hartwig (1997), "The group inverse of a companion matrix", *Lin. Mult. Alg.*, **43**, No 1-3, p. 137-150.
- [17] J.H. Wilkinson (1963), *Rounding Errors in Algebraic Processes*, Englewood Cliffs, NJ.: Prentice Hall, p.109.