Solving mixed sparse-dense linear least-squares problems by preconditioned iterative methods

SIAM J. SCI. COMPUT.
Vol. 39, No. 6, pp. A2422–A2437

SOLVING MIXED SPARSE-DENSE LINEAR LEAST-SQUARES PROBLEMS BY PRECONDITIONED ITERATIVE METHODS*

JENNIFER SCOTT† AND MIROSLAV TŮMA‡

Abstract. The efficient solution of large linear least-squares problems in which the system matrix A contains rows with very different densities is challenging. Previous work has focused on direct methods for problems in which A has a few relatively dense rows. These rows are initially ignored, a factorization of the sparse part is computed using a sparse direct solver, and then the solution is updated to take account of the omitted dense rows. In some practical applications the number of dense rows can be significant, and for very large problems, using a direct solver may not be feasible. We propose processing rows that are identified as dense separately within a conjugate gradient method using an incomplete factorization preconditioner combined with the factorization of a dense matrix of size equal to the number of dense rows. Numerical experiments on large-scale problems from real applications are used to illustrate the effectiveness of our approach. The results demonstrate that we can efficiently solve problems that could not be solved by a preconditioned conjugate gradient method without exploiting the dense rows.

Key words. sparse matrices, least-squares problems, conjugate gradients, preconditioning, incomplete factorizations

AMS subject classifications. Primary, 65F08, 65F20, 65F50; Secondary, 15A06, 15A23

DOI. 10.1137/16M1108339

1. Introduction.

Linear least-squares (LS) problems occur in a wide variety of practical applications, both in their own right and as subproblems of nonlinear LS problems. In this paper, we consider the unconstrained linear LS problem

$$\min_x \|Ax - b\|_2, \tag{1}$$

where $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ is a large sparse matrix and $b \in \mathbb{R}^m$ is given. Solving (1) is mathematically equivalent to solving the $n \times n$ *normal equations*

$$Cx = A^T b, \qquad C = A^T A, \tag{2}$$

where, if A has full column rank, the *normal matrix* C is symmetric and positive definite. In many cases, the number of entries in the rows of A can vary considerably. That is, some of the rows may be highly sparse while others contain a significant number of entries. The former are referred to as the *sparse rows* and the latter as the *dense rows* (although they may contain far fewer than n entries). If a sparse Cholesky or sparse QR factorization of the normal matrix is computed, the fill in the factors is catastrophic. Consequently, for large problems, a direct solver may fail

*Submitted to the journal's Methods and Algorithms for Scientific Computing section December 16, 2016; accepted for publication (in revised form) August 10, 2017; published electronically November 7, 2017.

http://www.siam.org/journals/sisc/39-6/M110833.html

Funding: The first author's work was partially supported by EPSRC grant EP/M025179/1. The second author's work was supported by project 17-04150J of the Grant Agency of the Czech Republic.

†Corresponding author. STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, Oxfordshire, OX11 0QX, UK, and School of Mathematical, Physical and Computational Sciences, University of Reading, Reading RG6 6AQ, UK (jennifer.scott@stfc.ac.uk).

‡Department of Numerical Mathematics, Faculty of Mathematics and Physics, Charles University, 186 75 Praha 8, Czech Republic (mirektuma@karlin.mff.cuni.cz).

because of insufficient memory, and if an incomplete factorization of C is employed as a preconditioner for an iterative solver, the error in the factorization can be so large as to prohibit its effectiveness as a preconditioner.

We assume that the rows of the (permuted) system matrix A are split into two parts with a conformal splitting of the right-hand side vector b as follows:

$$A = \begin{pmatrix} A_s \\ A_d \end{pmatrix},\ A_s \in R^{m_s \times n},\ A_d \in R^{m_d \times n},\ b = \begin{pmatrix} b_s \\ b_d \end{pmatrix},\ b_s \in R^{m_s},\ b_d \in R^{m_d}, \tag{3}$$

with $m = m_s + m_d$, $m_s \geq n$, and $m_d \geq 1$ (in general, $m_s \gg m_d$). Problem (1) then becomes

$$\min_x \left\| \begin{pmatrix} A_s \\ A_d \end{pmatrix} x - \begin{pmatrix} b_s \\ b_d \end{pmatrix} \right\|_2. \tag{4}$$

We initially assume that A_s has full column rank; rank deficiency of A_s is discussed in section 4. We define $C_s = A_s^T A_s$ to be the *reduced normal matrix*.

There are other motivations for splitting the rows of A. For example, a set of additional rows, which are not necessarily dense, is obtained by repeatedly adding new data into the LS estimation of parameters in a linear model (see, for example, the early papers [5, 6]), although the history dates back to Gauss and Stewart [17] (see also the historical overview in [15, section 10.5]). Nowadays, there exist important applications based on this motivation related to Kalman filtering or solving recursive LS problems; see the seminal paper [35] or, for a comprehensive introduction, [13, 44]. Also, dense and sparse blocks can arise in linearly constrained and rank-deficient LS problems (see, for instance, [7, 8, 37]). An analogous problem arises with interior point methods when a set of linear constraints $Ax = b$ has some dense columns on A (see, e.g. [24]). In this case, a sequence of matrices of the form AD^2A^T has to be formed in which the diagonal matrix D^2 changes at each step [36, 53]; see also the transformation to the positive-definite system in [38] that is affected by both dense rows and columns.

We observe that the need to process additional rows separately from the rest of the matrix is closely related to a number of other numerical linear algebra problems. These include finding sparsity-preserving orderings for a sparse QR decomposition [18]. Another related problem is that of separately processing a set of dense rows when the sparsity structure of A^TA is used to obtain an upper bound on the fill in an LU factorization of A with partial pivoting [21, 22].

Over the last 35 years or more, a number of papers have addressed the problem of A having a small number of dense rows. In [18], George and Heath propose temporarily discarding such rows to avoid severe fill-in in their Givens rotation–based orthogonal factorization of A_s. They then employ an updating scheme to incorporate the effect of the dense rows (the solution but not the factorization is updated). No numerical results are given in [18], but a procedure based on this was included in the package SPARSPAK-B [20]. In [31] Heath considers several other cases, including updating a sparse unconstrained problem of full rank when constraints are added. A completely general updating algorithm for the constrained LS problem based on QR decomposition and direct elimination is given by Björck [8], but without experimental results (see also [9, 48, 49]). Another possibility is to use an implicit transformation of the dense constraints as proposed in [34]. Other approaches to handling dense rows are based on the Schur-complement method; see, for example, [4, 23, 42, 47].

The application of C^{-1} can directly combine the inverse of C_s with an additional update based on A_d. A standard tool for this is the Woodbury formula (see [29, 51, 52]

and the comprehensive discussion in [30]). This formula enables the inverse C^{-1} to be written in the form

$$C^{-1} = (C_s + A_d^T A_d)^{-1} = C_s^{-1} - C_s^{-1} A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1} A_d C_s^{-1}. \tag{5}$$

The LS solution may then be explicitly expressed as

$$x = x_s + C_s^{-1} A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1} (b_d - A_d x_s) \quad \text{with} \quad x_s = (A_s A_s^T)^{-1} A_s^T b_s. \tag{6}$$

Note that here and elsewhere, for $k \geq 1$, I_k denotes the identity matrix of order k.

While this requires a factorization of C_s to be computed explicitly, Peters and Wilkinson [41] avoided this by computing a factorization $PA = LU$ with row permutations P to keep L well-conditioned and then forming the well-conditioned matrix $C_L = L^T L$ and factorizing it. The original paper was solely for dense A (with A ill-conditioned); Björck and Duff [10] subsequently extended the idea to the sparse case and also proposed a general updating scheme for modifying the solution after the addition of a few (possibly dense) rows. As this approach is dependent upon the computation of an LU factorization of A and also needs a factorization of C_L (which can be denser than the normal matrix C), it is generally more expensive than working directly with the normal equations. Sautter [43] observed that for a slightly overdetermined system ($m - n < n$) an algebraic reformulation that splits the trapezoidal L into a triangular part and a rectangular part can be advantageous in terms of the resulting flop count; see [9, subsection 2.5.1].

An alternative approach for dealing with a small number of dense rows uses the idea of stretching. Stretching aims to split the rows of A_d to obtain a matrix A_δ that has extra rows such that the corresponding LS problem has the same solution but the associated normal matrix is not dense. Matrix stretching was originally developed by Grcar [28] (see also [3, 14, 50]). The use of matrix stretching combined with a direct QR decomposition–based solver for LS problems is described by Adlers and Björck [2] (see also Adlers [1]).

In this paper, our focus is on using an iterative solver. We propose a preconditioner that is based on an incomplete Cholesky (IC) factorization of C_s and incorporates the effect of A_d using the factorization of a dense matrix of order m_d. The computational scheme is incorporated within the preconditioned CGLS method that was described in the seminal paper by Hestenes and Stiefel [32] (see also the CGLS1 variant from [11]). CGLS is an extension of the conjugate gradient (CG) method to LS problems and is mathematically equivalent to applying CG to the normal equations, but avoids actually forming them.

The rest of the paper is organized as follows. Section 2 introduces our approach to preconditioning within CGLS that exploits the row splitting (3) of A and combines an IC factorization of C_s with a factorization involving A_d. In section 3, numerical experiments using large-scale problems from practical applications demonstrate the efficiency and robustness of the proposed approach. Section 4 discusses dealing with the case that often occurs in the practice of A_s having null columns. Finally, section 5 presents some concluding remarks and possible future directions.

2. Iterative solution on sparse-dense splitting.

2.1. The use of sparse-dense splitting.

We start our discussion of solving the linear LS problem using the splitting (3) with a slight extension of the result from Sautter [43] (see also [8, 9]).

LEMMA 2.1. *Assume A and A_s have full column rank. Then with $C_s = A_s^T A_s$ and $C_d = A_d^T A_d$, the following identity holds:*

$$(C_s + C_d)^{-1} A_d^T = C_s^{-1} A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1}.$$

Proof. From (5),

$$\begin{aligned}(C_s + C_d)^{-1} A_d^T &= [C_s^{-1} - C_s^{-1} A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1} A_d C_s^{-1}] A_d^T \\ &= C_s^{-1} A_d^T - C_s^{-1} A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1} A_d C_s^{-1} A_d^T \\ &= C_s^{-1} A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1} (I_{m_d} + A_d C_s^{-1} A_d^T) \\ &\quad - C_s^{-1} A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1} A_d C_s^{-1} A_d^T \\ &= C_s^{-1} A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1}. \end{aligned}$$

∎

As discussed in the introduction, a standard strategy is to use a direct solver to compute a factorization of C_s to solve the problem

$$\min_{x_s} \|A_s x_s - b_s\|_2 \tag{7}$$

and to then incorporate the effect of A_d using an updating scheme. For large problems, or if C_s is not sufficiently sparse, this may not be practical. Instead, we may have only an approximate solution $\tilde{x}$ of (7). Let us consider the solution x to (4) as the sum of an approximate solution $\tilde{x}$ to (7) and a correction y. The following theorem shows the form of the correction.

THEOREM 2.2. *Assume that $\tilde{x}$ is an approximate solution to* (7). *Define $r_s = b_s - A_s\tilde{x}$ and $r_d = b_d - A_d\tilde{x}$. Then the LS solution of* (4) *is equal to $x = \tilde{x} + y$, where*

$$y = C_s^{-1}(A_s^T r_s + A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1}(r_d - A_d C_s^{-1} A_s^T r_s)).$$

Proof. The proof follows from straightforward verification with an application of Lemma 2.1.

$$\begin{aligned} C^{-1} A^T \begin{pmatrix} r_s^T & r_d^T \end{pmatrix}^T &= (C_s + C_d)^{-1}(A_s^T r_s + A_d^T r_d) \\ &= (C_s + C_d)^{-1}((C_s + C_d) C_s^{-1} A_s^T r_s - C_d C_s^{-1} A_s^T r_s + A_d^T r_d) \\ &= C_s^{-1} A_s^T r_s + (C_s + C_d)^{-1} A_d^T (r_d - A_d C_s^{-1} A_s^T r_s) \\ &= C_s^{-1} A_s^T r_s + C_s^{-1} A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1}(r_d - A_d C_s^{-1} A_s^T r_s). \end{aligned}$$

Adding the last formula to the approximation $\tilde{x}$ we get the result. ∎

To employ Theorem 2.2, a factorization of the reduced normal matrix

$$C_s = L_s L_s^T \tag{8}$$

must be available (so that C_s^{-1} can be applied by solving with the factors L_s and L_s^T). We can use the factors to consider the problem

$$\min_z \left\| \begin{pmatrix} B_s \\ B_d \end{pmatrix} z - \begin{pmatrix} b_s \\ b_d \end{pmatrix} \right\|_2, \tag{9}$$

with $B_s = A_s L_s^{-T}$, $B_d = A_d L_s^{-T}$, and $z = L_s^T y$.

The following lemma shows that Theorem 2.2 can be simplified provided the factorization (8) is available and we have the exact solution of the transformed LS problem $\|B_s z - b_s\|_2$, as assumed in the description of update strategies [8].

LEMMA 2.3. *The LS solution of problem* (9) *can be written as* $z = \tilde{x}_1 + y_1$, *where* $\tilde{x}_1$ *is an approximate solution to* (9) *and*

$$y_1 = B_s^T \rho_s + B_d^T (I_{m_d} + B_d B_d^T)^{-1} (\rho_d - B_d B_s^T \rho_s), \tag{10}$$

with $\rho_s = b_s - B_s \tilde{x}_1$ *and* $\rho_d = b_d - B_d \tilde{x}_1$.

Proof. Equation (10) follows directly from Theorem 2.2 using $B_s^T B_s = I_n$. ∎

Formula (10) in Lemma 2.3 offers a way to compute an approximate solution to (9). But there is a potential problem: in general, $B_s = A_s L_s^{-T}$ is dense, and so if m_s is large, it is not normally possible to store B_s explicitly. Thus if (10) is used, $t = B_s^T \rho_s$ should be computed by solving the triangular system $L_s t = A_s^T \rho_s$. To derive a practical procedure from (10) that we can use as a preconditioner we need to consider two important points. First, there is the issue of obtaining an initial approximate solution $\tilde{x}_1$; this is discussed in the following remark.

Remark 2.4. Theorem 2.2 and Lemma 2.3 depend on a given approximate solution $\tilde{x}_1$ to problem (9). There are two basic possibilities for choosing $\tilde{x}_1$. If the sparse part of (9) is important for the solution and B_d represents only a few dense rows or a problem update, $\tilde{x}_1$ can be chosen as an approximate solution of the problem

$$\min_{\tilde{x}} \|B_s \tilde{x} - b_s\|_2, \tag{11}$$

which (assuming A_s is overdetermined and of full rank) is given by

$$\tilde{x}_1 \approx \tilde{x} \equiv (B_s^T B_s)^{-1} B_s^T b_s = L_s^{-1} A_s^T A_s L_s^{-T} L_s^{-1} A_s^T b_s = L_s^{-1} A_s^T b_s. \tag{12}$$

However, if B_d represents a significant part of the problem and its effect dominates that of the sparse part, $\tilde{x}_1$ can be chosen as an approximate solution to

$$\min_{\tilde{x}} \|B_d \tilde{x} - b_d\|_2.$$

If we make no assumptions on the dimensions and the rank of B_d, the approximate solution can be expressed using the pseudoinverse as $\tilde{x} \approx \tilde{x}_1 \equiv B_d^{\dagger} b_d$. In our numerical experiments, we use only the approximation based on (11).

The second important point is the cost of the preconditioner, which directly relates to the problem of choosing $\tilde{x}_1$. A motivation for our choice is formulated in the following lemma.

LEMMA 2.5. *The LS solution of problem* (9) *can be written as* $z = \tilde{x}_1 + y_1$, *where* $\tilde{x}_1$ *minimizes* $\|B_s z - b_s\|_2$ *exactly and*

$$y_1 = B_d^T (I_{m_d} + B_d B_d^T)^{-1} \rho_d, \tag{13}$$

with $\rho_d = b_d - B_d \tilde{x}_1$.

Proof. If $\tilde{x}_1$ minimizes $\|B_s z - b_s\|_2$, then $B_s^T B_s \tilde{x}_1 = B_s^T b_s$. That is, $B_s^T \rho_s = 0$ with $\rho_s = b_s - B_s \tilde{x}_1$ and (10) reduces to (13). ∎

Clearly, evaluating (13) is significantly more efficient than evaluating (10). Both involve applying $B_d^T (I_{m_d} + B_d B_d^T)^{-1}$ to a vector g, where for (10) $g = \rho_d - B_d B_s^T \rho_s$ and for (13) $g = \rho_d$. This is equivalent to finding the first n components of the minimum norm solution h of the "fat" system

$$Fh = g, \qquad F = \begin{pmatrix} B_d & I_{m_d} \end{pmatrix} \in R^{m_d \times (n+m_d)}.$$

There are several ways to approach this. One possibility is to compute the Cholesky factorization

$$I_{m_d} + B_d B_d^T = L_d L_d^T \tag{14}$$

and obtain the result of

$$B_d^T (I_{m_d} + B_d B_d^T)^{-1} g \tag{15}$$

as the first n components of $F^T (L_d L_d^T)^{-1} g$. This approach applied as a direct method can be found in [9]. An alternative is to use an LQ factorization (see, for example, [9, 19, 39]),

$$F = \begin{pmatrix} L_d & 0_{m_d} \end{pmatrix} Q_d^T. \tag{16}$$

Then (15) can be evaluated as $Q_d(1:n, 1:m_d) L_d^{-1} g$.

For large problems, it may not be possible to compute a complete factorization of the reduced normal matrix; instead, only an IC factorization of the form $C_s \approx \tilde{L}_s \tilde{L}_s^T$ may be available. Similarly, $\tilde{B}_d = A_d^T \tilde{L}_s^{-T}$ and (14) and (16) may be replaced by incomplete factorizations

$$I_{m_d} + \tilde{B}_d \tilde{B}_d^T \approx \tilde{L}_d \tilde{L}_d^T \qquad \text{and} \qquad F \approx \begin{pmatrix} \tilde{L}_d & 0_{m_d} \end{pmatrix} \tilde{Q}_d^T. \tag{17}$$

2.2. Preconditioned iterative method. We now propose incorporating preconditioning into solving the mixed sparse-dense linear LS problem (4) by the conjugate gradient method (CGLS). This combines incomplete factorizations of the reduced normal matrix C_s and of the transformed dense part (17). We use a conformal splitting of each of the vectors of length m within the CGLS algorithm because our preconditioners exploit this splitting. In particular, the residual is explicitly split into r_s and r_d.

ALGORITHM 2.6. **Preconditioned CGLS algorithm**

Input: *$A \in R^{m \times n}$ with $m \geq n$ and of full column rank with its rows split into two parts A_s and A_d with $A_s \in R^{m_s \times n}$ with $m_s \geq n$ and of full column rank; an incomplete factorization $A_s^T A_s \approx \tilde{L}_s \tilde{L}_s^T$; a right-hand side vector $b \in R^m$ split into b_s and b_d conformally with the splitting of A; the initial solution $x^{(0)} \in R^n$; preconditioning operation ($z = M^{-1}s$) given in Algorithm 2.7; the maximum number of iterations $nmax$.*

Output: *The computed solution x.*

0. **Initialization:** $r_s^{(0)} = b_s - A_s x^{(0)}$, $r_d^{(0)} = b_d - A_d x^{(0)}$, $w_s^{(0)} = A_s^T r_s^{(0)}$, $w_d^{(0)} = A_d^T r_d^{(0)}$, $z^{(0)} = M^{-1}(w_s^{(0)} + w_d^{(0)})$, $p^{(0)} = z^{(0)}$

1. **for** $i = 1 : nmax$ **do**

2. $q_s^{(i-1)} = A_s p^{(i-1)}$, $q_d^{(i-1)} = A_d p^{(i-1)}$

3. $$\alpha = \frac{(w_s^{(i-1)} + w_d^{(i-1)}, z^{(i-1)})}{(q_s^{(i-1)}, q_s^{(i-1)}) + (q_d^{(i-1)}, q_d^{(i-1)})}$$

4. $x^{(i)} = x^{(i-1)} + \alpha p^{(i-1)}$

5. $r_s^{(i)} = r_s^{(i-1)} - \alpha q_s^{(i-1)}$, $r_d^{(i)} = r_d^{(i-1)} - \alpha q_d^{(i-1)}$

6. *evaluate the stopping criterion; terminate if satisfied with* $x = x^{(i)}$ *or if* $n =$ *nmax, with a warning*

7. $w_s^{(i)} = A_s^T r_s^{(i)}$, $w_d^{(i)} = A_d^T r_d^{(i)}$

8. $z^{(i)} = M^{-1}(w_s^{(i)} + w_d^{(i)})$

9. $$\beta = \frac{(w_s^{(i)} + w_d^{(i)}, z^{(i)})}{(w_s^{(i-1)} + w_d^{(i-1)}, z^{(i-1)})}$$

10. $p^{(i)} = z^{(i)} + \beta p^{(i-1)}$

11. **end do**

Algorithm 2.7 presents the preconditioning algorithm. Note that the preconditioner is applied to r_s and r_d that play the role of the right-hand sides b_s and b_d in the formulae explained in the previous section. In addition, we use $w_s = A_s^T r_s$ computed in step 7 of Algorithm 2.6. This is needed to compute $\tilde{x}_1$ (which in turn is used to compute ρ_d). Also note that $\tilde{B}_d$ is not computed explicitly; rather $I_{m_d} + \tilde{B}_d \tilde{B}_d^T$ is computed implicitly as $I_{m_d} + A_d^T \tilde{L}_s^{-T} \tilde{L}_s^{-1} A_d$. We present two possible modes: the mode *Cholesky* corresponds to (13), and mode *LQ* uses an approximate LQ factorization based on (16).

ALGORITHM 2.7. **Preconditioning procedure**

Input: *Residual vector components* r_s, r_d, *transformed residual component* w, *an incomplete factorization* $C_s \approx \tilde{L}_s \tilde{L}_s^T$, *and* $\tilde{B}_d = A_d^T \tilde{L}_s^{-T}$, *chosen mode (Cholesky or LQ). The Cholesky mode needs a (possibly incomplete) Cholesky factorization* $I_{m_d} + \tilde{B}_d \tilde{B}_d^T \approx \tilde{L}_d \tilde{L}_d^T$, *the LQ mode needs a possibly incomplete factorization* $\begin{pmatrix} \tilde{B}_d & I_{m_d} \end{pmatrix} \approx \begin{pmatrix} \tilde{L}_d & 0_{m_d} \end{pmatrix} \tilde{Q}_d^T$.

Output: *Computed* $z = M^{-1} w$

1. *Solve* $\tilde{L}_s \tilde{x}_1 = w_s$ *for* $\tilde{x}_1$

2. *Solve* $\tilde{L}_s^T v = \tilde{x}_1$ *for* v

3. $\rho_d = r_d - A_d^T v$

4. **if** *mode == Cholesky* **then**

5. *Solve* $\tilde{L}_d u_1 = \rho_d$ *for* u_1

6. *Solve* $\tilde{L}_d^T u_2 = u_1$ *for* u_2

7. *Solve* $\tilde{L}_s u = A_d u_2$ *for* u

8. **else if** *mode == LQ* **then**

9. $\rho_s = r_s - A_s v$

10. *Solve* $\tilde{L}_s t = A_s^T \rho_s$ *for* t

11. *Solve* $\tilde{L}_s^T t_1 = t$ *for* t_1

12. *Solve* $\tilde{L}_d t_2 = \rho_d - A_d^T t_1$ *for* t_2

13. $u = t + \tilde{Q}_d(1 : n, 1 : m_d) t_2$

14. **end if**

15. *Solve* $\tilde{L}_s^T z = (\tilde{x}_1 + u)$ *for* z

3. Numerical experiments. We present numerical results to illustrate the potential of our proposed approach for handling dense rows of A. We use Algorithm 2.6 with the Cholesky mode employed in the preconditioning step (Algorithm 2.7). Any sparsity in the dense part A_d is ignored, and the Cholesky factorization of $I_{m_d} + \tilde{B}_d \tilde{B}_d^T$ uses the LAPACK routine `_potrf`. To perform the IC factorization $A_s^T A_s \approx \tilde{L}_s \tilde{L}_s^T$, we use the package `HSL_MI35` from the HSL mathematical software library [33]. `HSL_MI35` implements a limited memory IC algorithm (see [46, 45] for details). Note that it handles ordering for sparsity and scaling. In our tests, we use default settings. `HSL_MI35` requires the user to set the parameters $lsize$ and $rsize$ that respectively control the number of entries in each column of the IC factor and the memory required to compute the factorization. In general, increasing these parameters improves the quality of the preconditioner (so that the number of iterations of the preconditioned iterative solver is reduced) at the cost of more time and memory to compute the factorization. In our experiments, unless stated otherwise, we use $lsize = rsize = 5$ so that the number of entries in $\tilde{L}_s$ is at most $5n$.

3.1. Test environment. Our test set is described in Table 1. With the exception of PDE1, all the examples are part of the University of Florida Sparse Matrix Collection [12]. Problem PDE1 is taken from the CUTEst linear programming set [25]. Note that the University of Florida examples used here can also be found in the CUTEst set, but with slightly different identifiers. All the test problems were included in the study by Gould and Scott [26, 27].

We scale A by normalizing each of its columns. That is, we replace A by AD, where D is the diagonal matrix with entries D_{ii} satisfying $D_{ii}^2 = 1/\|Ae_i\|_2$ (e_i denotes the ith unit vector). The entries of AD are all less than one in absolute value.

In our experiments, we define a row of A to be dense if the number of entries either exceeds 100 times the average number of entries per row or is more than 4 times greater than the maximum number of entries in a row in the sparse part A_s. For most of our test cases, this choice is not critical, although for the Mittelmann/neos examples, we found the results can be improved by relaxing these conditions so that fewer rows are classified as dense. Removing dense rows can leave A_s rank-deficient. In section 4, we discuss how we can develop a practical procedure to deal with this case. However, here for simplicity we modify the problem by removing any columns of A that correspond to null columns of A_s.

To terminate the preconditioned CGLS algorithm, we employ the following stopping rules taken from [27]:

C1: Stop if $\|r\|_2 < \delta_1$.

C2: Stop if

$$\frac{\|A^T r\|_2}{\|r\|_2} < \frac{\|A^T r^{(0)}\|_2}{\|r^{(0)}\|_2} * \delta_2,$$

where $r = b - Ax$ is the residual, $r^{(0)} = b - Ax^{(0)}$ is the initial residual, and δ_1 and δ_2 are convergence tolerances that we set to 10^{-8} and 10^{-6}, respectively. In all our experiments, b is taken to be the vector of all 1's and we take the initial solution guess to be $x^{(0)} = 0$ so that C2 reduces to

$$\frac{\|A^T r\|_2}{\|r\|_2} < \frac{\|A^T b\|_2}{\|b\|_2} * \delta_2.$$

TABLE 1

Statistics for our test set. m, n, and $nnz(A)$ are the row and column counts and the number of entries in A. $nnz(C)$ is the number of entries in the lower triangle of $C = A^T A$, and density(C) is the ratio of the number of entries in C to n^2.

Identifier	m	n	$nnz(A)$	nnz(C)	density(C)
Meszaros/aircraft	7,517	3,754	20,267	1.4×10^6	0.200
Meszaros/lp_fit2p	13,525	3,000	50,284	4.5×10^6	1.000
Meszaros/scrs8-2r	27,691	14,364	58,439	6.2×10^6	0.143
Meszaros/sctap1-2b	33,858	15,390	99,454	2.6×10^6	0.050
Meszaros/scsd8-2r	60,550	8,650	190,210	2.0×10^6	0.100
Meszaros/scagr7-2r	62,423	35,213	123,239	2.2×10^7	0.036
Meszaros/sc205-2r	62,423	35,213	123,239	6.5×10^6	0.010
Meszaros/sctap1-2r	63,426	28,830	186,366	9.1×10^6	0.050
Meszaros/scfxm1-2r	65,943	37,980	221,388	8.3×10^5	0.014
Mittelmann/neos1	133,743	131,581	599,590	1.7×10^8	0.027
Mittelmann/neos2	134,128	132,568	685,087	2.3×10^8	0.033
Meszaros/stormg2-125	172,431	66,185	433,256	1.0×10^6	0.002
PDE1	270,595	271,792	990,587	1.6×10^{10}	0.670
Mittelmann/neos	515,905	479,119	1,526,794	5.3×10^8	0.034
Mittelmann/stormg2_1000	1,377,306	528,185	3,459,881	4.2×10^7	0.002
Mittelmann/cont1_l	1,921,596	1,918,399	7,031,999	8.2×10^{11}	0.667

Our experiments are performed on an Intel(R) Core(TM) i5-4590 CPU running at 3.30 GHz with 12 GB of internal memory. All software is written in Fortran, and the Visual Fortran Intel(R) 64 XE compiler (version 14.0.3.202) was used. Reported timings are elapsed times in seconds.

3.2. Summary results for our test set. In Table 2, we summarize our findings for the complete test set, with and without exploiting dense rows. Without exploiting dense rows, we are unable to solve many of the largest problems, either because the IC factorization time exceeds 1000 seconds or because convergence is not achieved within 2000 CGLS iterations. Along with the storage for the incomplete factors, when dense rows are exploited, we need to store the $m_d \times n$ entries of A_d together with the $m_d(m_d+1)/2$ entries of the dense Cholesky factor of $I_{m_d} + \tilde{B}_d \tilde{B}_d^T$.

We see that our preconditioning strategy that exploits dense rows significantly reduces the iteration count and computation times. Furthermore, it is able to solve problems that `HSL_MI35` applied to the whole of A did not solve. A more detailed look at the behavior of our strategy on some of our test problems is now presented.

3.3. Results for Meszaros/scsd8-2r. In the following, we look at (i) the number $nnz(C_s)$ of entries in the reduced normal matrix (lower triangle only), (ii) the CGLS iteration counts, and (iii) the ratio *ratio* of the number of entries in the matrices that are factorized to the number *size_ps* of entries in the preconditioner, that is,

$$ratio = (nnz(C_s) + m_d(m_d+1)/2) \,/\, (size\_ps + m_d(m_d+1)/2)\,.$$

The first example that we consider is problem Meszaros/scsd8-2r. Figure 1 illustrates the size of the reduced normal matrix C_s as the number of rows that are classified as dense increases. Then in Figures 2 and 3, we present the CGLS iteration counts, *ratio*, and the times to compute the preconditioner and run CGLS. As expected, increasing the number m_d of rows that are classified as dense from 0 to the value 50 that was specified in Table 2 significantly reduces the size of C_s; *ratio* also decreases. We also see that the iteration counts can decrease before m_d reaches 50.

TABLE 2

Results with and without exploiting dense rows. size_p and size_ps denote the number of entries in the incomplete factors $\tilde{L}$ and $\tilde{L}_s$ of C and C_s, respectively. Its is the number of CGLS iterations; T_p and T_its denote the times (in seconds) to compute the preconditioner and for convergence of CGLS, respectively. † indicates time to compute the preconditioner exceeds 1000 seconds; ‡ denotes convergence of CGLS not achieved.

	Dense rows not exploited				Dense rows exploited				
Identifier	*size_p*	*T_p*	*Its*	*T_its*	m_d	*size_ps*	*T_p*	*Its*	*T_its*
Meszaros/aircraft	22,509	0.09	44	0.02	17	3,750	0.01	1	0.01
Meszaros/lp_fit2p	17,985	0.26	‡	‡	25	4,940	0.09	1	0.01
Meszaros/scrs8-2r	86,169	0.94	380	0.50	22	36,385	0.01	1	0.02
Meszaros/sctap1-2b	92,325	0.39	639	0.69	34	68,644	0.01	1	0.01
Meszaros/scsd8-2r	51,885	0.25	90	0.11	50	51,855	0.05	7	0.02
Meszaros/scagr7-2r	197,067	3,34	244	0.53	7	152,977	0.06	1	0.01
Meszaros/sc205-2r	211,257	1.56	72	0.19	8	104,022	0.08	1	0.01
Meszaros/sctap1-2r	172,965	1.47	673	1.90	34	127,712	0.03	1	0.01
Meszaros/scfxm1-2r	227,835	0.59	187	0.51	58	227,823	0.14	33	0.23
Mittelmann/neos1	789,471	†	†	†	74	789,471	5.27	132	3.71
Mittelmann/neos2	†	†	†	†	90	795,323	5.46	157	4.84
Meszaros/stormg2-125	395,595	0.27	‡	‡	121	7,978,135	0.22	16	0.29
PDE1	†	†	†	†	1	1,623,531	12.7	696	1.28
Mittelmann/neos	†	†	†	†	20	2,874,699	4.93	232	15.0
Mittelmann/stormg2_1000	3,157,095	19.1	‡	‡	121	3,125,987	19.1	18	2.92
Mittelmann/cont1_l	†	†	†	†	1	11,510,370	4.82	1	0.33

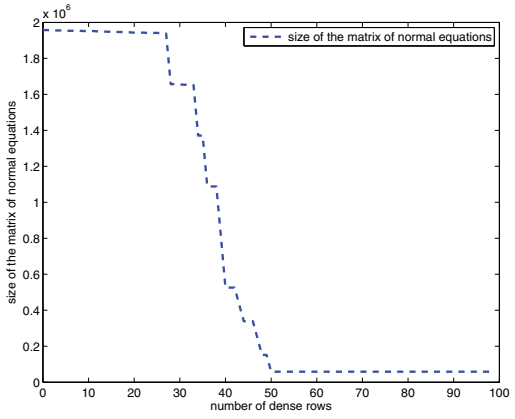

FIG. 1. *The number of entries in C_s for problem Meszaros/scsd8-2r as the number m_d of dense rows that are exploited increases.*

This demonstrates not only that dense rows lead to high memory demands, but that their presence can reduce the IC factorization quality as well. The timings reflect the same general dependence on m_d. The fluctuations in the times needed to compute the preconditioner given in the left-hand plot of Figure 3 are a result of the IC factorization code performing a number of restarts. If the IC factorization breaks down (which happens if a zero or negative pivot is encountered), `HSL_MI35` introduces a positive shift α and restarts the factorization with C_s replaced by $C_s + \alpha I$. Since an appropriate choice of α is not known a priori, the factorization may need to restart more than once with α increased on each occasion (`HSL_MI35` takes care of this automatically), and this is reflected in the computation time. To investigate whether the problem can be solved more efficiently by applying a higher quality preconditioner, Figures 4

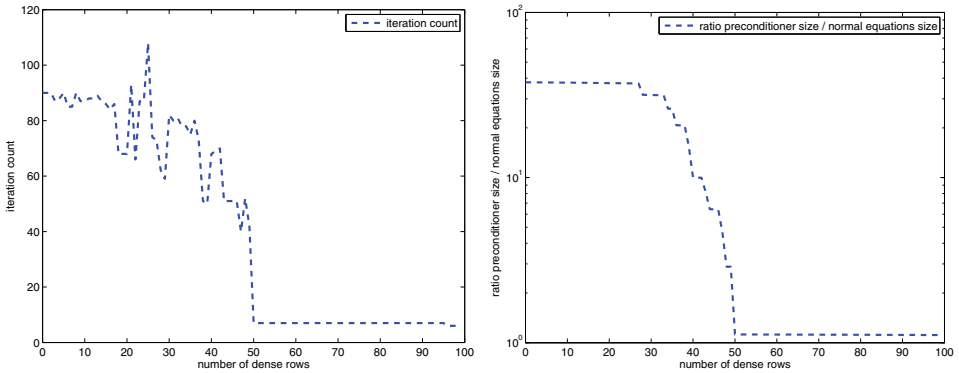

FIG. 2. *Problem Meszaros/scsd8-2r. Iteration counts (left) and ratio (right) as the number m_d of dense rows that are exploited increases.*

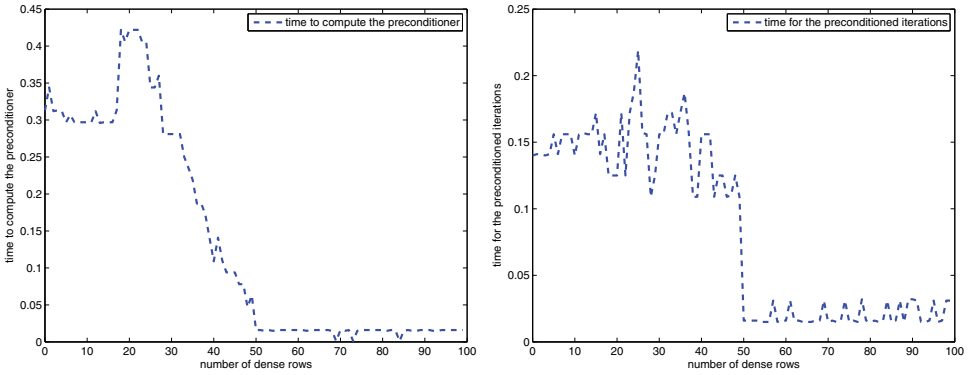

FIG. 3. *Problem Meszaros/scsd8-2r. Time to compute the preconditioner (left) and time for CGLS (right) as the number m_d of dense rows that are exploited increases.*

and 5 report results for `HSL_MI35` with the parameters *lsize* and *rsize* that control the number of entries in the incomplete factor $\tilde{L}_s$ increased from 5 to 20. We see that the qualitative behavior remains the same and is consistent with our assumptions.

3.4. Results for Mittelmann/stormg2_1000. Next we consider the large example Mittelmann/stormg2_1000; results are given in Figures 6–8. Again, the time for computing the preconditioner is influenced by the number of restarts needed during the incomplete factorization, and this is not a smooth function of the number of dense rows. The CGLS time drops dramatically when all 121 dense rows reported in Table 2 are exploited.

4. Dealing with null columns. As already observed, even if A has full column rank, A_s may not have full column rank. Indeed, in practice, A_s can contain null columns. In this section, we explain how this can be overcome. Let A have full rank, and assume A_s has n_2 null columns with $n_2 \ll n$. Assuming these columns are permuted to the end, we have the splitting

$$A = \begin{pmatrix} A_1 & A_2 \end{pmatrix} \equiv \begin{pmatrix} A_{s_1} & 0 \\ A_{d_1} & A_{d_2} \end{pmatrix}.$$

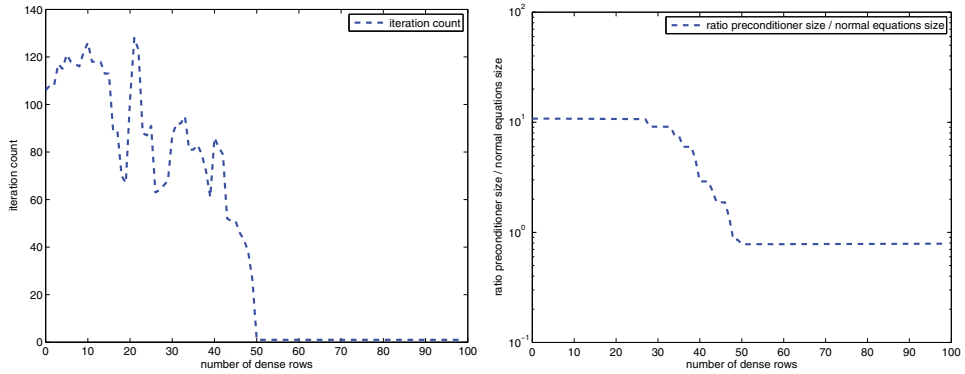

FIG. 4. *Problem Meszaros/scsd8-2r with lsize = rsize = 20. Iteration counts (left) and ratio (right) as the number m_d of dense rows that are exploited increases.*

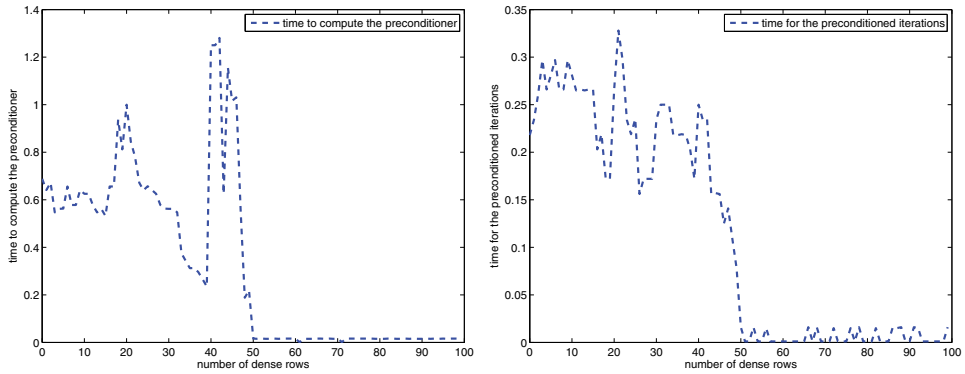

FIG. 5. *Problem Meszaros/scsd8-2r with lsize = rsize = 20. Time to compute the preconditioner (left) and time for CGLS (right) as the number m_d of dense rows that are exploited increases.*

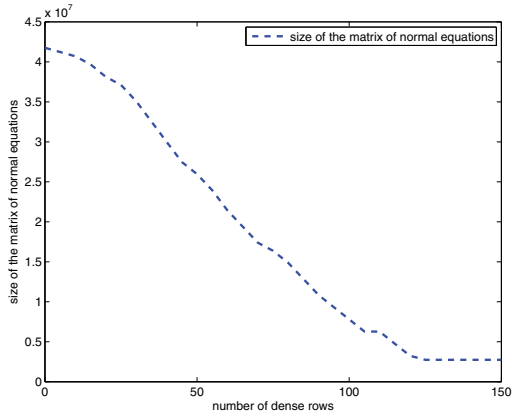

FIG. 6. *The number of entries in C_s for problem Mittelmann/stormg2_1000 as the number m_d of dense rows that are exploited increases.*

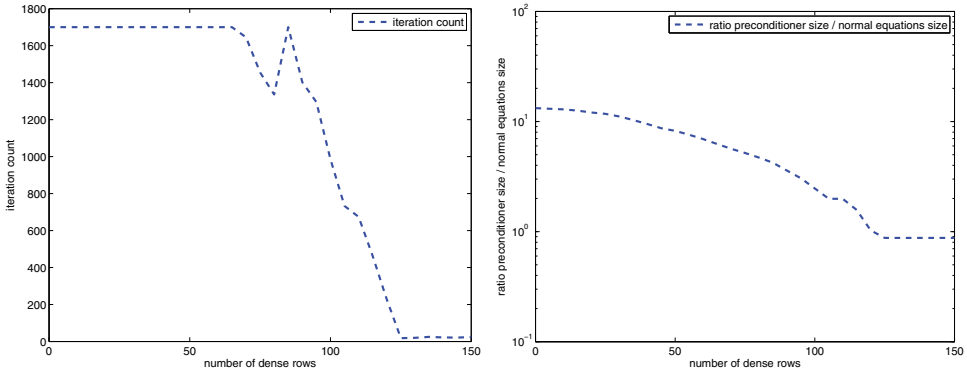

FIG. 7. *Problem Mittelmann/stormg2_1000. Iteration counts (left) and ratio (right) as the number m_d of dense rows that are exploited increases.*

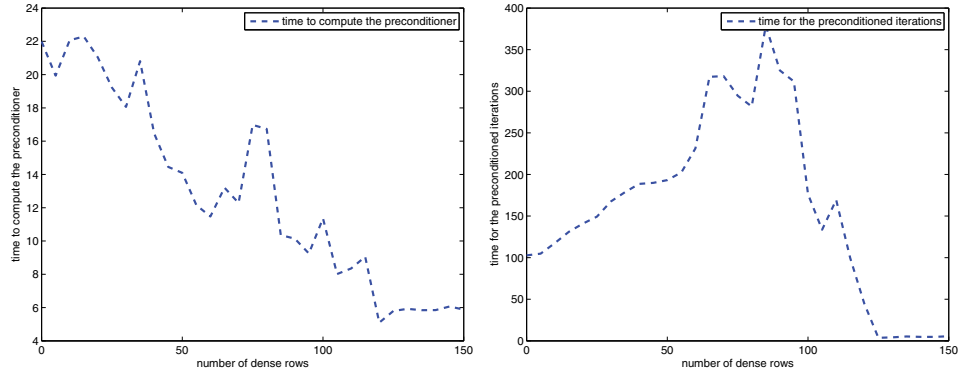

FIG. 8. *Problem Mittelmann/stormg2_1000. Time to compute the preconditioner (left) and time for CGLS (right) as the number m_d of dense rows that are exploited increases.*

The following result shows that the solution of the LS problem (1) can be expressed as a combination of partial solutions.

THEOREM 4.1. *Let $z \in R^{n_1}$ and $W \in R^{n_1 \times n_2}$ be the solutions to the problems*

$$\min_z \|A_1 z - b\|_2 \quad \text{and} \quad \min_W \|A_1 W - A_2\|_F, \tag{18}$$

respectively. Then the solution $x = \binom{x_1}{x_2}$ of problem (1) with its splitting consistent with (18) (that is, $x_1 \in R^{n_1}$, $x_2 \in R^{n_2}$) is given by

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} z - W x_2 \\ x_2 \end{pmatrix}, \quad x_2 = (A_2^T (A_2 - A_1 W))^{-1} A_2^T (b - A_1 z).$$

Proof. We have

$$A^T A x = \begin{pmatrix} A_1^T A_1 & A_1^T A_2 \\ A_2^T A_1 & A_2^T A_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} A_1^T b \\ A_2^T b \end{pmatrix}. \tag{19}$$

Furthermore, $z = (A_1^T A_1)^{-1} A_1^T b$ and $W = (A_1^T A_1)^{-1} A_1^T A_2$. Premultiplying the first block row of (19) by $(A_1^T A_1)^{-1}$ gives the result. ∎

Theorem 4.1 suggests a practical procedure for solving LS problems with some dense rows: compute partial solutions z and W corresponding to the nonnull column block A_1 using Algorithm 2.6 with $n_2 + 1$ right-hand sides, and then use the result of Theorem 4.1. This requires forming and then solving with $A_2^T(A_2 - A_1W)$. Provided n_2 is small, this is inexpensive.

5. Concluding remarks. We have looked at the problem of solving linear LS problems in which the system matrix A has a number of dense rows. We have proposed an approach that processes the dense rows separately within a CG method, using an incomplete factorization preconditioner for the sparse rows and a complete Cholesky factorization of the small dense subproblem arising from the dense rows. We note that other iterative methods, including LSQR [40] and LSMR [16], could be employed. Likewise, other preconditioners could be used. The reported numerical experiments on practical problems that each have between 1 and 121 dense rows demonstrate the potential advantages of our proposed approach. In particular, we efficiently solved large problems that we were not able to solve using preconditioned conjugate gradients without exploiting the dense rows. As expected, it is important for efficiency to exploit all the dense rows (that is, to move them all into A_d). We remark that in our experiments we found that the strategy based upon (13) gave better results than using the more complex scheme (10). We conjecture that this conclusion would be different if the preconditioner were based on a perturbed direct solver rather than on an incomplete factorization; this requires further investigation that is beyond the scope of this paper.

There remain other issues that need to be addressed before we have fully reliable and robust preconditioned iterative solvers for general LS problems. In particular, more work needs to be done to address rank-deficiency. In the future, we plan to look at other possible splittings and transformations of A based on the problem structure to improve the preconditioner quality further.

Finally, we remark that Avron, Ng, and Toledo [7] look at using a QR factorization of A to solve linear LS problems. If A has a few rows that are identified as dense, they recommend that these rows be dropped before the QR factorization starts. They use the resulting R factor as a preconditioner for LSQR and show that if m_d dense rows are dropped, then LSQR is expected to converge in at most m_d+1 iterations. We experimented with simply dropping the dense rows (that is, we discarded A_d and just used the IC factorization of C_s), but we found that in general the incomplete factorization of C_s gave very poor convergence, confirming the importance of incorporating the dense rows within the preconditioned iterative solver.

Acknowledgments. We would like to thank Michael Saunders and an anonymous reviewer for their careful reading of our manuscript and for their constructive comments that led to improvements in the presentation of this paper.

REFERENCES

[1] M. Adlers, *Topics in Sparse Least Squares Problems*, Technical Report, Department of Mathematics, Linköping University, Linköping, Sweden, 2000.

[2] M. Adlers and Å. Björck, *Matrix stretching for sparse least squares problems*, Numer. Linear Algebra Appl., 7 (2000), pp. 51–65.

[3] F. L. Alvarado, *Matrix enlarging methods and their application*, BIT, 37 (1997), pp. 473–505.

[4] K. D. Andersen, *A modified Schur complement method for handling dense columns in interior point methods for linear programming*, ACM Trans. Math. Software, 22 (1996), pp. 348–356.

[5] O. D. Anderson, *An improved approach to inverting the autocovariance matrix of a general mixed autoregressive moving average time process*, Austral. J. Statist., 18 (1976), pp. 73–75.
[6] O. D. Anderson, *On the inverse of the autocovariance matrix for a general moving average process*, Biometrika, 63 (1976), pp. 391–394.
[7] H. Avron, E. Ng, and S. Toledo, *Using perturbed QR factorizations to solve linear least-squares problems*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 674–693, https://doi.org/10.1137/070698725.
[8] Å. Björck, *A general updating algorithm for constrained linear least squares problems*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 394–402, https://doi.org/10.1137/0905029.
[9] Å. Björck, *Numerical Methods for Least Squares Problems*, SIAM, Philadelphia, 1996, https://doi.org/10.1137/1.9781611971484.
[10] Å. Björck and I. S. Duff, *A direct method for the solution of sparse linear least squares problems*, Linear Algebra Appl., 34 (1980), pp. 43–67.
[11] Å. Björck, T. Elfving, and Z. Strakoš, *Stability of conjugate gradient and Lanczos methods for linear least squares problems*, SIAM J. Matrix Anal. Appl., 19 (1998), pp. 720–736, https://doi.org/10.1137/S089547989631202X.
[12] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), pp. 1–28.
[13] P. S. R. Diniz, *Adaptive Filtering: Algorithms and Practical Implementation*, 4th ed., Springer, New York, 2013.
[14] I. S. Duff and J. A. Scott, *Stabilized bordered block diagonal forms for parallel sparse solvers*, Parallel Comput., 31 (2005), pp. 275–289.
[15] R. W. Farebrother, *Fitting Linear Relationships: A History of the Calculus of Observations* 1750–1900, Springer-Verlag New York, 1999.
[16] D. C.-L. Fong and M. Saunders, *LSMR: An iterative algorithm for sparse least-squares problems*, SIAM J. Sci. Comput., 33 (2011), pp. 2950–2971, https://doi.org/10.1137/10079687X.
[17] C. F. Gauss and G. W. Stewart, *Theory of the Combination of Observations Least Subject to Errors Part One, Part Two, Supplement*, Classics Appl. Math. 11, SIAM, Philadelphia, 1995, https://doi.org/10.1137/1.9781611971248.
[18] A. George and M. T. Heath, *Solution of sparse linear least squares problems using Givens rotations*, Linear Algebra Appl., 34 (1980), pp. 69–83.
[19] A. George, M. T. Heath, and E. Ng, *Solution of sparse underdetermined systems of linear equations*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 988–997, https://doi.org/10.1137/0905068.
[20] A. George and E. Ng, *SPARSPAK: Waterloo Sparse Matrix Package User's Guide for SPARSPAK-B*, Research Report CS-84-37, Department of Computer Science, University of Waterloo, Waterloo, ON, Canada, 1984.
[21] A. George and E. Ng, *An implementation of Gaussian elimination with partial pivoting for sparse systems*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 390–409, https://doi.org/10.1137/0906028.
[22] A. George and E. Ng, *Symbolic factorization for sparse Gaussian elimination with partial pivoting*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 877–898, https://doi.org/10.1137/0908072.
[23] P. E. Gill, W. Murray, D. B. Ponceleon, and M. A. Saunders, *Solving Reduced KKT Systems in Barrier Methods for Linear and Quadratic Programming*, Technical Report SOL 91-7, Department of Operations Research, Stanford University, Stanford, CA, 1991.
[24] D. Goldfarb and K. Scheinberg, *A product-form Cholesky factorization method for handling dense columns in interior point methods for linear programming*, Math. Program., 99 (2004), pp. 1–34.
[25] N. I. M. Gould, D. Orban, and Ph. L. Toint, *CUTEst: A constrained and unconstrained testing environment with safe threads for mathematical optimization*, Comput. Optim. Appl., 60 (2015), pp. 545–557.
[26] N. I. M. Gould and J. A. Scott, *The State-of-the-Art of Preconditioners for Sparse Linear Least Squares Problems: The Complete Results*, Technical Report RAL-TR-2015-009, Rutherford Appleton Laboratory, Didcot, Oxfordshire, England, 2015.
[27] N. I. M. Gould and J. A. Scott, *The state-of-the-art of preconditioners for sparse linear least-squares problems*, ACM Trans. Math. Software, 43 (2017), 36.
[28] J. F. Grcar, *Matrix Stretching for Linear Equations*, Technical Report SAND90-8723, Sandia National Laboratories, Albuquerque, NM, 1990.
[29] L. Guttman, *Enlargement methods for computing the inverse matrix*, Ann. Math. Statistics,

17 (1946), pp. 336–343.
[30] W. W. Hager, *Updating the inverse of a matrix*, SIAM Rev., 31 (1989), pp. 221–239, https://doi.org/10.1137/1031049.
[31] M. T. Heath, *Some extensions of an algorithm for sparse linear least squares problems*, SIAM J. Sci. Statist. Comput., 3 (1982), pp. 223–237, https://doi.org/10.1137/0903014.
[32] M. R. Hestenes and E. Stiefel, *Methods of conjugate gradients for solving linear systems*, J. Research Nat. Bur. Standards, 49 (1952), pp. 409–436.
[33] HSL, *A Collection of Fortran Codes for Large-Scale Scientific Computation*, http://www.hsl.rl.ac.uk, 2016.
[34] T.-M. Hwang, W.-W. Lin, and D. Pierce, *A Robust Method for Handling Dense Rows in a Sparse QR or RRQR Factorization*, Technical Report ISSTECH-96-021, Boeing Information and Support Services, Plano, TX, 1996.
[35] R. E. Kalman, *A new approach to linear filtering and prediction problems*, Trans. ASME J. Basic Eng. Ser. D, 82 (1960), pp. 35–45.
[36] I. J. Lustig, R. E. Marsten, and D. F. Shanno, *On implementing Mehrotra's predictor–corrector interior-point method for linear programming*, SIAM J. Optim, 2 (1992), pp. 435–449, https://doi.org/10.1137/0802022.
[37] E. Ng, *A scheme for handling rank-deficiency in the solution of sparse linear least squares problems*, SIAM J. Sci. Statist. Comput., 12 (1991), pp. 1173–1183, https://doi.org/10.1137/0912062.
[38] A. R. L. Oliveira and D. C. Sorensen, *A new class of preconditioners for large-scale linear systems from interior point methods for linear programming*, Linear Algebra Appl., 394 (2005), pp. 1–24.
[39] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. Numer. Anal., 12 (1975), pp. 617–629, https://doi.org/10.1137/0712047.
[40] C. C. Paige and M. A. Saunders, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982), pp. 43–71.
[41] G. Peters and J. H. Wilkinson, *The least squares problem and pseudo-inverse*, Comput. J., 131 (1970), pp. 309–316.
[42] M. A. Saunders, *Cholesky-based methods for sparse least squares: The benefits of regularization*, in Linear and Nonlinear Conjugate Gradient-Related Methods, L. Adams and J. L. Nazareth, eds., SIAM, Philadelphia, 1996, pp. 92–100.
[43] W. Sautter, *Fehleranalyse für die Gauss-Elimination zur Berechnung der Lösung minimaler Länge*, Numer. Math., 30 (1978), pp. 165–184.
[44] A. H. Sayed, *Fundamentals of Adaptive Filtering*, John Wiley and Sons, Hoboken, NJ, 2003.
[45] J. Scott and M. Tůma, *On positive semidefinite modification schemes for incomplete Cholesky factorization*, SIAM J. Sci. Comput., 36 (2014), pp. A609–A633, https://doi.org/10.1137/130917582.
[46] J. A. Scott and M. Tůma, `HSL_MI28`*: An efficient and robust limited-memory incomplete Cholesky factorization code*, ACM Trans. Math. Software, 40 (2014), 24.
[47] J. A. Scott and M. Tůma, *A Schur Complement Approach to Preconditioning Sparse Linear Least-Squares Problems with Some Dense Rows*, Technical Report RAL-TR-2017-P-004, Rutherford Appleton Laboratory, Didcot, Oxfordshire, England, 2017.
[48] C. Sun, *Dealing with Dense Rows in the Solution of Sparse Linear Least Squares Problems*, Research Report CTC95TR227, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, Ithaca, NY, 1995.
[49] C. Sun, *Parallel solution of sparse linear least squares problems on distributed-memory multiprocessors*, Parallel Comput., 23 (1997), pp. 2075–2093.
[50] R. J. Vanderbei, *Splitting dense columns in sparse linear systems*, Linear Algebra Appl., 152 (1991), pp. 107–117.
[51] M. A. Woodbury, *The Stability of Out-Input Matrices*, University of Chicago Press, Chicago, IL, 1949.
[52] M. A. Woodbury, *Inverting Modified Matrices*, Statistical Research Group, Memo. Rep. 42, Princeton University, Princeton, NJ, 1950.
[53] M. H. Wright, *Interior methods for constrained optimization*, in Acta Numerica, Vol. 1, Cambridge University Press, Cambridge, UK, 1992, pp. 341–407.