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

Article

Published Version

Creative Commons: Attribution 4.0 (CC-BY)

Open Access


It is advisable to refer to the publisher’s version if you intend to cite from the work.

To link to this article DOI: http://dx.doi.org/10.1137/16M1108339

Publisher: Society for Industrial and Applied Mathematics

All outputs in CentAUR are protected by Intellectual Property Rights law, including copyright law. Copyright and IPR is retained by the creators or other copyright holders. Terms and conditions for use of this material are defined in the End User Agreement.

www.reading.ac.uk/centaur
CentAUR

Central Archive at the University of Reading

Reading's research outputs online
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

\[
Cx = A^T b, \quad C = A^T A, \tag{2}
\]

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...
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:

\[
A = \begin{pmatrix} A_s \\ A_d \end{pmatrix}, \quad A_s \in \mathbb{R}^{m_s \times n}, \quad A_d \in \mathbb{R}^{m_d \times n}, \quad b = \begin{pmatrix} b_s \\ b_d \end{pmatrix}, \quad b_s \in \mathbb{R}^{m_s}, \quad b_d \in \mathbb{R}^{m_d},
\]

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

\[
\min_x \left\| \begin{pmatrix} A_s \\ A_d \end{pmatrix} x - \begin{pmatrix} b_s \\ b_d \end{pmatrix} \right\|_2.
\]

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 \( AD^2 A^T \) 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 \( A^T A \) 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

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

The LS solution may then be explicitly expressed as

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

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 As have full column rank. Then with Cs = AsTAs and Cd = AdTAd, the following identity holds:

\[(Cs + Cd)^{-1}A_{d}^{T} = C_s^{-1}A_d^T(I_{m_d} + A_dC_s^{-1}A_d^T)^{-1}.\]

Proof. From (5),

\[(Cs + Cd)^{-1}A_{d}^{T} = |C_s^{-1} - C_s^{-1}A_d^T(I_{m_d} + A_dC_s^{-1}A_d^T)^{-1}A_dC_s^{-1}|A_d^T\]

\[= C_s^{-1}A_d^T - C_s^{-1}A_d^T(I_{m_d} + A_dC_s^{-1}A_d^T)^{-1}A_dC_s^{-1}A_d^T\]

\[= C_s^{-1}A_d^T(I_{m_d} + A_dC_s^{-1}A_d^T)^{-1}(I_{m_d} + A_dC_s^{-1}A_d^T)\]

\[= C_s^{-1}A_d^T(I_{m_d} + A_dC_s^{-1}A_d^T)^{-1}A_dC_s^{-1}A_d^T\]

\[= C_s^{-1}A_d^T(I_{m_d} + A_dC_s^{-1}A_d^T)^{-1}. \]

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

\[(7) \quad \min_{x_s} \| A_s x_s - b_s \|_2 \]

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

THEOREM 2.2. Assume that ˜x is an approximate solution to (7). Define rs = bs - As ˜x and rd = bd - Ad ˜x. Then the LS solution of (4) is equal to x = ˜x + y, where

\[y = C_s^{-1}(A_s^T r_s + A_d^T(I_{m_d} + A_dC_s^{-1}A_d^T)^{-1}(r_d - A_dC_s^{-1}A_d^T r_s)).\]

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

\[C^{-1}A^T (r_s^T \quad r_d^T)^T = (Cs + Cd)^{-1}(A_s^T r_s + A_d^T r_d)\]

\[= (Cs + Cd)^{-1}((Cs + Cd)C_s^{-1}A_s^T r_s - C_dC_s^{-1}A_s^T r_s + A_d^T r_d)\]

\[= C_s^{-1}A_s^T r_s + (Cs + Cd)^{-1}A_d^T (r_d - A_dC_s^{-1}A_d^T r_s)\]

\[= C_s^{-1}A_s^T r_s + C_s^{-1}A_d^T(I_{m_d} + A_dC_s^{-1}A_d^T)^{-1}(r_d - A_dC_s^{-1}A_d^T r_s).\]

Adding the last formula to the approximation ˜x we get the result.

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

\[(8) \quad C_s = L_s L_s^T \]

must be available (so that C−1 s can be applied by solving with the factors Ls and Ls T). We can use the factors to consider the problem

\[(9) \quad \min_z \left\| \begin{bmatrix} B_s \\ B_d \end{bmatrix} z - \begin{bmatrix} b_s \\ b_d \end{bmatrix} \right\|_2, \]

with Bs = AsL−T s, Bd = AdL−T s, and z = Ls 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 \( z = \hat{x}_1 + y_1 \), where \( \hat{x}_1 \) is an approximate solution to (9) and

\[
y_1 = B_s^T \rho_s + B_d^T (I_{md} + B_d B_d^T)^{-1} (\rho_d - B_d B_s^T \rho_s),
\]

with \( \rho_s = b_s - B_s \hat{x}_1 \) and \( \rho_d = b_d - B_d \hat{x}_1 \).

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 problem (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 \( \hat{x}_1 \); this is discussed in the following remark.

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

\[
\min_{x} \| B_s x - b_s \|_2,
\]

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

\[
\hat{x}_1 \approx \hat{x} = (B_s^T B_s)^{-1} B_s^T b_s = L_s^{-1} A_s^T A_s L_s^{-T} L_s^{-1} A_s^T b_s = L_s^{-1} A_s^T b_s.
\]

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

\[
\min_{x} \| B_d x - b_d \|_2.
\]

If we make no assumptions on the dimensions and the rank of \( B_d \), the approximate solution can be expressed using the pseudoinverse as \( \hat{x} \approx \hat{x}_1 \equiv B_d^T b_d \). 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 \( \hat{x}_1 \). A motivation for our choice is formulated in the following lemma.

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

\[
y_1 = B_d^T (I_{md} + B_d B_d^T)^{-1} \rho_d,
\]

with \( \rho_d = b_d - B_d \check{x}_1 \).

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

Clearly, evaluating (13) is significantly more efficient than evaluating (10). Both involve applying \( B_d^T (I_{md} + B_d B_d^T)^{-1} \) 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

\[
F h = g,
\]

\[
F = (B_d \ I_{md}) \in R^{m_d \times (n+m_d)}.
\]
There are several ways to approach this. One possibility is to compute the Cholesky factorization

\[(14) \quad I_{m_d} + B_d B_d^T = L_d L_d^T \]

and obtain the result of

\[(15) \quad B_d^T (I_{m_d} + B_d B_d^T)^{-1} g \]

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, \([9, 19, 39]\)),

\[(16) \quad F = (L_d \quad 0_{m_d}) \tilde{Q}_d^T. \]

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

\[(17) \quad I_{m_d} + \tilde{B}_d \tilde{B}_d^T \approx \tilde{L}_d \tilde{L}_d^T \quad \text{and} \quad F \approx (\tilde{L}_d \quad 0_{m_d}) \tilde{Q}_d^T. \]

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

**Output:** The computed solution \(x\).

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

   2. **for** \(i = 1 : nmax\) **do**
      3. \(q_s(i-1) = A_s p(i-1)\), \(q_d(i-1) = A_d p(i-1)\)
      4. \(\alpha = \frac{(w_s(i-1) + w_d(i-1), z(i-1))}{(q_s(i-1), q_d(i-1)) + (q_d(i-1), q_d(i-1))}\)
      5. \(x(i) = x(i-1) + \alpha q_d(i-1)\)
      6. \(r_s(i) = r_s(i-1) - \alpha q_s(i-1)\), \(r_d(i) = r_d(i-1) - \alpha q_d(i-1)\)

\(\copyright 2017\) SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license.
6. evaluate the stopping criterion; terminate if satisfied with $x = x^{(i)}$ or if $n = n_{\text{max}}$, with a warning.

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

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 $w_s = A^T r_s$ computed in step 7 of Algorithm 2.6. This is needed to compute $\tilde{x}_1$ (which in turn is to compute $\rho_d$). Also note that $B_d$ is not computed explicitly; rather $I + B_dB_d^T$ is computed implicitly as $I + A_d^T A_d^{-1} A_d$. 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 $C_s \approx L_sL_s^T$, and $B_d = A_d^T L_s^{-T}$, chosen mode (Cholesky or LQ). The Cholesky mode needs $a$ (possibly incomplete) Cholesky factorization $I_{m_d} + B_dB_d^T \approx L_dL_d^T$, the LQ mode needs a possibly incomplete factorization $(B_d I_{m_d}) \approx (L_d 0_{m_d}) \tilde{Q}_d^T$.

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

1. Solve $L_s\tilde{x}_1 = w_s$ for $\tilde{x}_1$
2. Solve $L_d^Tv = \tilde{x}_1$ for $v$
3. $\rho_d = r_d - A_d^Tv$
4. if mode == Cholesky then
5. Solve $L_d u_1 = \rho_d$ for $u_1$
6. Solve $L_d^T u_2 = u_1$ for $u_2$
7. Solve $L_s u = A_d u_2$ for $u$
8. else if mode == LQ then
9. $\rho_s = r_s - A_s v$
10. Solve $L_s t = A_s^T \rho_s$ for $t$
11. Solve $L_d^T t_1 = t$ for $t_1$
12. Solve $L_d t_2 = \rho_d - A_d^T t_1$ for $t_2$
13. $u = t + \tilde{Q}_d (1: n, 1: m_d) t_2$

© 2017 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license
14. \textbf{end if}
15. \textbf{Solve} \( \tilde{L}^T \tilde{s} = (\tilde{x}_1 + u) \) 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 \( I_{m_d} + B_d B_d^T \) uses the LAPACK routine \texttt{potrf}. To perform the IC factorization \( A_d^T A_d \approx \tilde{L}_d \tilde{L}_d^T \), we use the package \texttt{HSL_MI35} from the HSL mathematical software library [33]. \texttt{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. \texttt{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 \( \tilde{L}_d \) 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 \( D_{ii} = 1/\|Ae_i\|_2 \) (where \( e_i \) denotes the \( i \)th 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]:
\begin{itemize}
  \item C1: Stop if \( \|r\|_2 < \delta_1 \).
  \item C2: Stop if \( \frac{\|A^T r\|_2}{\|r\|_2} < \frac{\|A^T r(0)\|_2}{\|r(0)\|_2} \times \delta_2 \),
\end{itemize}
where \( r = b - Ax \) is the residual, \( r(0) = b - A x(0) \) is the initial residual, and \( \delta_1 \) and \( \delta_2 \) 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
\[ \frac{\|A^T r\|_2}{\|r\|_2} < \frac{\|A^T b\|_2}{\|b\|_2} \times \delta_2. \]
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 \(I_{m_d} + B_d B_d^T\).

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 \(\text{ratio}\) of the number of entries in the matrices that are factorized to the number \(\text{size.ps}\) of entries in the preconditioner, that is,

\[
\text{ratio} = \frac{(nnz(C_s) + m_d(m_d + 1)/2)}{(\text{size.ps} + m_d(m_d + 1)/2)}.
\]

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, \(\text{ratio}\), 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 \(C_s\); \(\text{ratio}\) also decreases. We also see that the iteration counts can decrease before \(m_d\) reaches 50.
Results with and without exploiting dense rows. \( \text{size}_p \) and \( \text{size}_{ps} \) denote the number of entries in the incomplete factors \( \tilde{L} \) and \( \tilde{L}_s \) of \( C \) and \( C_s \), respectively. \( \text{Its} \) 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. \( \dagger \) indicates time to compute the preconditioner exceeds 1000 seconds; \( \ddagger \) denotes convergence of CGLS not achieved.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>( \text{size}_p )</th>
<th>( p )</th>
<th>( T_p )</th>
<th>( \text{Its} )</th>
<th>( T_{Its} )</th>
<th>( m_d )</th>
<th>( \text{size}_{ps} )</th>
<th>( p )</th>
<th>( T_p )</th>
<th>( \text{Its} )</th>
<th>( T_{Its} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Meszaros/aircraft</td>
<td>22,509</td>
<td>0.09</td>
<td>44</td>
<td>0.02</td>
<td>17</td>
<td>3,750</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meszaros/lpfit2p</td>
<td>17,985</td>
<td>0.26</td>
<td>‡</td>
<td>‡</td>
<td>25</td>
<td>4,940</td>
<td>0.09</td>
<td>1</td>
<td>0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meszaros/scrs8-2r</td>
<td>86,169</td>
<td>0.94</td>
<td>380</td>
<td>0.50</td>
<td>22</td>
<td>36,385</td>
<td>0.01</td>
<td>1</td>
<td>0.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meszaros/scap1-2b</td>
<td>92,325</td>
<td>0.39</td>
<td>639</td>
<td>0.69</td>
<td>34</td>
<td>68,644</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meszaros/scsd8-2r</td>
<td>51,885</td>
<td>0.25</td>
<td>90</td>
<td>0.11</td>
<td>50</td>
<td>51,855</td>
<td>0.05</td>
<td>7</td>
<td>0.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meszaros/scag7-2r</td>
<td>197,067</td>
<td>3.34</td>
<td>244</td>
<td>0.53</td>
<td>7</td>
<td>152,977</td>
<td>0.06</td>
<td>1</td>
<td>0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meszaros/sc205-2r</td>
<td>211,257</td>
<td>1.56</td>
<td>72</td>
<td>0.19</td>
<td>8</td>
<td>104,022</td>
<td>0.08</td>
<td>1</td>
<td>0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meszaros/scap1-2r</td>
<td>172,965</td>
<td>1.47</td>
<td>673</td>
<td>1.90</td>
<td>34</td>
<td>127,712</td>
<td>0.03</td>
<td>1</td>
<td>0.01</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meszaros/scfcm1-2r</td>
<td>227,835</td>
<td>0.59</td>
<td>187</td>
<td>0.51</td>
<td>58</td>
<td>227,823</td>
<td>0.14</td>
<td>33</td>
<td>0.23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mittelmann/neos1</td>
<td>789,471</td>
<td>‡</td>
<td>‡</td>
<td>‡</td>
<td>74</td>
<td>789,471</td>
<td>5.27</td>
<td>132</td>
<td>3.71</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mittelmann/neos2</td>
<td>‡</td>
<td>‡</td>
<td>‡</td>
<td>‡</td>
<td>90</td>
<td>795,323</td>
<td>5.46</td>
<td>157</td>
<td>4.84</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meszaros/stormg2-125</td>
<td>395,595</td>
<td>0.27</td>
<td>‡</td>
<td>‡</td>
<td>121</td>
<td>7,978,155</td>
<td>0.22</td>
<td>16</td>
<td>0.29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PDE1</td>
<td>‡</td>
<td>‡</td>
<td>‡</td>
<td>‡</td>
<td>1</td>
<td>1,623,531</td>
<td>12.7</td>
<td>696</td>
<td>1.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mittelmann/neos</td>
<td>‡</td>
<td>‡</td>
<td>‡</td>
<td>‡</td>
<td>20</td>
<td>2,874,099</td>
<td>4.95</td>
<td>232</td>
<td>15.0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mittelmann/stormg2-1000</td>
<td>3,157,095</td>
<td>19.1</td>
<td>‡</td>
<td>‡</td>
<td>121</td>
<td>3,125,987</td>
<td>19.1</td>
<td>18</td>
<td>2.92</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mittelmann/cont1</td>
<td>‡</td>
<td>‡</td>
<td>‡</td>
<td>‡</td>
<td>1</td>
<td>11,510,370</td>
<td>4.82</td>
<td>1</td>
<td>0.33</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 \( \alpha \) and restarts the factorization with \( C_s \) replaced by \( C_s + \alpha I \). Since an appropriate choice of \( \alpha \) is not known a priori, the factorization may need to restart more than once with \( \alpha \) 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...
and 5 report results for HSL_MI35 with the parameters lsize and rsize that control the number of entries in the incomplete factor $L_s$ 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

$$A = (A_1 \ A_2) \equiv \begin{pmatrix} A_{s1} & 0 \\ A_{d1} & A_{d2} \end{pmatrix}. $$
Fig. 4. Problem Meszaros/scsd8-2r with isize = rsize = 20. Iteration counts (left) and ratio (right) as the number md of dense rows that are exploited increases.

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

Fig. 6. The number of entries in Cs for problem Mittelmann/stormg2_1000 as the number md 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 $z \in \mathbb{R}^{n_1}$ and $W \in \mathbb{R}^{n_1 \times n_2}$ be the solutions to the problems

\begin{equation}
\min_{x} \| A_1 x - b \|_2 \quad \text{and} \quad \min_{W} \| A_1 W - A_2 \|_F,
\end{equation}

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

\[
\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} z - Wx_2 \\ x_2 \end{pmatrix}, \quad x_2 = (A_2^T (A_2 - A_1 W))^{-1} A_2^T (b - A_1 z).
\]

**Proof.** We have

\begin{equation}
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}.
\end{equation}

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.

© 2017 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license.
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 $A_2^T(A_2 - A_1 W)$. 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


© 2017 SIAM. Published by SIAM under the terms of the Creative Commons 4.0 license


MIXED SPARSE-DENSE LINEAR LEAST-SQUARES PROBLEMS


