Science & Technology Facilities Council

Preprint

RAL-P-2017-001

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

J Scott, M Tuma

January 2017

Submitted for publication in SIAM Journal of Scientific Computing

RAL Library
STFC Rutherford Appleton Laboratory
R61
Harwell Oxford
Didcot
OX11 0QX

Tel: +44(0)1235 445384
Fax: +44(0)1235 446403
email: libraryral@stfc.ac.uk

ISSN 1361- 4762

SOLVING MIXED SPARSE-DENSE LINEAR LEAST SQUARES 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 rows that have a relatively large number of entries. These dense rows are initially ignored, a factorization of the sparse part is computed using a sparse direct solver and then the solution updated to take account of the omitted dense rows. In some practical applications the number of dense rows can be significant. In this paper, we propose processing such rows separately within a conjugate gradient method using an incomplete factorization preconditioner and the factorization of a dense matrix of size equal to the number of rows identified as dense. Numerical experiments on large-scale problems from real applications are used to illustrate the effectiveness of our approach.

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

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

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

$$\min_x \|Ax - b\|_2, \tag{1.1}$$

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 (1.1) is mathematically equivalent to solving the $n \times n$ *normal equations*

$$Cx = A^T b, \qquad C = A^T A, \tag{1.2}$$

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 resulting 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

$$A = \begin{pmatrix} A_s \\ A_d \end{pmatrix},\ A_s \in R^{m_s\times n},\ A_d \in R^{m_d\times n},\ b = \begin{pmatrix} b_s \\ b_d \end{pmatrix},\ b_s \in R^{m_s},\ b_d \in R^{m_d}, \tag{1.3}$$

with $m = m_s + m_d$, $m_s \geq n$, and $m_d \geq 1$ (in general, $m_s \gg m_d$). Problem (1.1) then becomes

$$\min_x \left\| \begin{pmatrix} A_s \\ A_d \end{pmatrix} x - \begin{pmatrix} b_s \\ b_d \end{pmatrix} \right\|_2. \tag{1.4}$$

We define $C_s = A_s^T A_s$ to be the *reduced normal matrix*.

* STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, Oxfordshire, OX11 0QX, UK. Correspondence to: jennifer.scott@stfc.ac.uk. Supported by EPSRC grant EP/M025179/1.

† Department of Numerical Mathematics, Faculty of Mathematics and Physics, Charles University, Sokolovska 83, 186 75 Praha 8, Czech Republic, (mirektuma@karlin.mff.cuni.cz.) Supported by the project 13-06684S of the Grant Agency of the Czech Republic and by the ERC project MORE LL1202 financed by the MŠMT of the Czech Republic.

Besides splitting A to reflect the differences in the densities of its rows, there are other possible motivations for such a splitting. For example, a set of additional rows, that are not necessarily dense, is obtained by repeatedly adding new data into the least squares estimation of parameters in a linear model, see, for example, the early papers [5, 6], but the history dates back to Gauss [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 least squares, see the seminal paper [34] or for a comprehensive introduction [13, 42]. Furthermore, additional constraints for the least squares represented by A_d and b_d naturally arise when solving rank-deficient least squares problems (for instance, [7, 8, 36]). Note also that a similar problem arises in the solution of underdetermined systems with dense columns representing a set of constraints in interior point methods [23]. 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 [35, 50]; see also the transformation to the positive-definite system in [37] that is effected 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 [20, 21].

Over the last thirty-five years or more, a number of papers have addressed the problem of the system matrix 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 scheme 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 [19]. In [30] Heath considers several other cases, including updating a sparse unconstrained problem of full rank when both equations and constraints are added. A completely general updating algorithm for the constrained least squares problem based on the use of the QR decomposition and direct elimination is given by Björck [8], but without experimental results (see also [9, 45, 46]). Another possibility is to use an implicit transformation of the dense constraints as proposed in [33]. Other approaches to handling dense rows are based on the Schur-complement method, see for example, [4, 22, 40].

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 [28, 48, 49] and the comprehensive discussion in [29]). This formula enables the inverse C^{-1} to be written in the form

$$C^{-1} = (C_s + A_d^T A_d)^{-1} = C_s^{-1} - C_s^{-1}A_d^T(I_{m_d} + A_d C_s^{-1} A_d^T)A_d C_s^{-1}. \tag{1.5}$$

The least squares solution may then be explicitly expressed as

$$x = x_s + C_s^{-1}A_d^T(I_{m_d} + A_d C_s^{-1} A_d^T)(b_d - A_d x_s) \quad \text{with} \quad x_s = (A_s A_s^T)^{-1}A_s^T b_s. \tag{1.6}$$

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, the Peters-Wilkinson algorithm [39] avoids this. Peters-Wilkinson considers an LU factorization of A with pivoting so that L is well-conditioned; a factorization of $C_L = L^TL$ is then computed. The original paper was solely for dense A and was designed to overcome the problem that the normal matrix can be highly 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 extra (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 as dense or denser than the normal matrix C), it is generally more expensive than working directly with the normal equations. Sautter [41] observed 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 least squares problem has the same solution but the associated normal matrix is not dense. Matrix stretching was originally developed by Grcar [27] (see also [3, 14, 47]). The use of matrix stretching combined with a direct QR decomposition-based solver for least squares 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 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 [31] (see also the CGLS1 variant from [11]). CGLS is an extension of the conjugate gradient method (CG) to least squares problems and is mathematically equivalent to applying CG to the normal equations, but avoids actually forming them.

The rest of the paper is organised as follows. Section 2 introduces our new approach to preconditioning within CGLS that exploits the row splitting (1.3) of A and combines an IC factorization of C_s with a factorization based on A_d. In Section 3, numerical experiments using large-scale problems from practical applications demonstrate the efficiency and robustness of the new approach. Section 4 discusses dealing with the case that often occurs in practice of A_s having null columns. Finally, Section 5 presents some concluding remarks and possible future directions.

2. Iterative solution of the least squares based on sparse-dense splitting.

2.1. The use of sparse-dense splitting. We start our discussion of solving the linear least squares problem using the splitting (1.3) with a slight extension of the result from Sautter [41] (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 (1.5)

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

□

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

$$\min_{x_s} \|A_s x_s - b_s\|_2, \tag{2.1}$$

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 only have only an approximate solution ξ of (2.1). Let us consider the solution x to (1.4) as the sum of an approximate solution ξ to (2.1) and a correction Γ. The following theorem shows the form of the correction.

THEOREM 2.2. *Assume that ξ is an approximate solution to (2.1). Define $r_s = b_s - A_s\xi$ and $r_d = b_d - A_d\xi$. Then the least squares solution of (1.4) is equal to $x = \xi + \Gamma$, where*

$$\Gamma = C_s^{-1} A_s^T r_s + C_s^{-1} A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1} (r_d - A_d C_s^{-1} A_s^T r_s) \tag{2.2}$$

Proof. The proof follows from straightforward verification with an application 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 ξ we get the result. □

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

$$C_s = L_s L_s^T \tag{2.3}$$

needs to 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 following problem

$$\min_z \left\| \begin{pmatrix} B_s \\ B_d \end{pmatrix} z - \begin{pmatrix} b_s \\ b_d \end{pmatrix} \right\|_2, \tag{2.4}$$

for $B_s = A_s L_s^{-T}$, $B_d = A_d L_s^{-T}$ and $z = L_s^T \Gamma$.

The following lemma shows that Theorem 2.2 can be simplified provided the factorization (2.3) is available and we have the exact solution of the scaled least squares problem $\|B_s z - b_s\|_2$, as is assumed in the description of update strategies [8].

LEMMA 2.3. *If $C_s = L_s L_s^T$, the least squares solution of problem (2.4) can be written as $z = \xi_1 + \Gamma_1$, where ξ_1 is an approximate solution to (2.4), $\rho_s = b_s - B_s\xi_1$ and $\rho_d = b_d - B_d\xi_1$ and*

$$\Gamma_1 = B_s^T \rho_s + B_d^T(I_{m_d} + B_d B_d^T)^{-1}(\rho_d - B_d B_s^T \rho_s). \tag{2.5}$$

Proof. (2.5) directly follows from Theorem 2.2 using $B_s^T B_s = I_n$. □

The formula (2.5) in Lemma 2.3 offers a way to compute an approximate solution to (2.4). But there is one 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 (2.5) 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 (2.5) 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 ξ_1; this is discussed in the following remark.

REMARK 2.1. *Theorem 2.2 and Lemma 2.3 depend on a given approximate solution ξ_1 to problem (2.4). There are two basic possibilities for choosing ξ_1. If the sparse part of (2.4) is important for the solution and B_d represents only a few dense rows or a problem update, ξ_1 can be estimated as an approximate solution of the problem*

$$\min_\xi \|B_s\xi - b_s\|_2, \tag{2.6}$$

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

$$\xi_1 \approx (B_s^T B_s)^{-1} B_s^T b_s = L_s^{-1} A_s^T A_s L_s^{-T} L_s^{-1} A_s^T b_s = L_s^{-1} A_s^T b_s. \tag{2.7}$$

However, if B_d represents a significant part of the problem and its effect dominates that of the sparse part, ξ_1 can be estimated as an approximate solution to

$$\min_\xi \|B_d\xi - b_d\|_2.$$

If we make no assumptions on the dimensions and the rank of B_d, the approximate solution can be expressed using the pseudo inverse as $\xi \approx B_d^\dagger b_d$. In our numerical experiments, we use only the approximation based on (2.6).

The second important point is the cost of the preconditioner that directly relates to the problem of the choice of ξ_1 and a motivation for this choice is formulated in the following Lemma 2.4.

LEMMA 2.4. *If $C_s = L_s L_s^T$, the least squares solution of problem (2.4) can be written as $z = \xi_1 + \Gamma_1$, where ξ_1 minimizes $\|B_s z - b_s\|_2$ exactly, $\rho_s = b_s - B_s\xi_1$ and $\rho_d = b_d - B_d\xi_1$ and*

$$\Gamma_1 = B_d^T (I_{m_d} + B_d B_d^T)^{-1} \rho_d. \tag{2.8}$$

Proof. If ξ_1 minimizes $\|B_s z - b_s\|_2$, then $B_s^T \rho_s = 0$ and (2.5) reduces to (2.8). □

Clearly, evaluating (2.8) is significantly more efficient than evaluating (2.5). Both involve applying $B_d^T(I_{m_d} + B_d B_d^T)^{-1}$ to a vector g, where for (2.5) $g = \rho_d - B_d B_s^T \rho_s$ and for (2.8) $g = \rho_d$. 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 a number of ways to perform this. One possibility is to compute the Cholesky factorization

$$I_{m_d} + B_d B_d^T = L_d L_d^T \tag{2.9}$$

and then obtain the result of

$$B_d^T (I_{m_d} + B_d B_d^T)^{-1} g \tag{2.10}$$

as the first n components of $F^T (L_d L_d^T)^{-1} g$. An alternative is to use an LQ factorization

$$F = \begin{pmatrix} L_d & 0_{m_d} \end{pmatrix} Q_d^T. \tag{2.11}$$

Then (2.10) can be evaluated using the LQ decomposition 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, (2.9) and (2.11) may be replaced by incomplete factorizations

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

2.2. Preconditioned iterative method. We now propose a new iterative approach for solving mixed sparse-dense systems (1.4) preconditioned by a combination of an IC factorization of the reduced normal matrix C_s and of the transformed A_d. The split nature of the preconditioner is embedded into the preconditioned CGLS. In our algorithm outline, the preconditioning is formulated using the notation B_d, $\tilde{L}_d$ introduced in the previous section. We use a conformal splitting of each of the vectors of length m that are used in CGLS because our preconditioners exploit this splitting. In particular, the preconditioning procedure is able to avoid recomputing $w_s = A_s^T r_s$ and it explicitly uses the splitting of the residual into r_s and r_d.

ALGORITHM 2.1. **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 2.2; 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. $\alpha = \dfrac{(w_s^{(i-1)} + w_d^{(i-1)}, z^{(i-1)})}{(q_s^{(i-1)}, q_s^{(i-1)}) + (q_d^{(i-1)}, q_d^{(i-1)})}$
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. $\beta = \dfrac{(w_s^{(i)} + w_d^{(i)}, z^{(i)})}{(w_s^{(i-1)} + w_d^{(i-1)}, z^{(i-1)})}$
10. $p^{(i)} = z^{(i)} + \beta p^{(i-1)}$
11. **end do**

Algorithm 2.2 presents the preconditioning algorithm. Note that the preconditioner is applied to the residual parts 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 at step 7 of Algorithm 2.1. We present two possible modes: the mode *Cholesky* corresponds to (2.8) and mode *LQ* uses an approximate LQ factorization based on (2.11).

ALGORITHM 2.2. **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* $B_d = A_d^T \tilde{L}_s^{-T}$, *chosen mode (Cholesky or LQ). The Cholesky mode needs a (possibly incomplete) Cholesky factorization* $I_{m_d} + B_d B_d^T \approx \tilde{L}_d \tilde{L}_d^T$, *the LQ mode needs a possibly incomplete factorization* $\begin{pmatrix} B_d & I_{m_d} \end{pmatrix} \approx \begin{pmatrix} \tilde{L}_d & 0_{m_d} \end{pmatrix} \tilde{Q}_d^T$.

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

1. *Solve* $\tilde{L}_s \xi_1 = w$ *for* ξ_1
2. $\rho_d = r_d - B_d \xi_1$
3. **if** *mode == Cholesky* **then**
4. $u = B_d^T (\tilde{L}_d \tilde{L}_d^T)^{-1} \rho_d$
5. **else if** *mode == LQ* **then**
6. $\rho_s = r_s - A_s \tilde{L}_s^{-T} \xi_1$
7. $u = \tilde{L}_s^{-1} A_s^T \rho_s + \tilde{Q}_d(1 : n, 1 : m_d) * \tilde{L}_d^{-1} * (\rho_d - B_d \tilde{L}_s^{-1} A_s^T \rho_s)$

8. **end if**

9. *Solve* $\tilde{L}_s^T z = (\xi_1 + u)$ *for* z

---

3. Numerical experiments. In this section, we present numerical results to illustrate the potential of our proposed approach for handling dense rows of A. We use Algorithm 2.1 with the Cholesky mode employed in the preconditioning step (Algorithm 2.2). 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 `_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` that is available as part of the HSL mathematical software library [32]. `HSL_MI35` implements a limited memory IC algorithm (see [43, 44] 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 3.1. 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 [24]. 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 [25, 26].

TABLE 3.1

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

We prescale A so that the entries of the scaled A are small relative to 1. Thus we scale A by normalising each column by its 2-norm. 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 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 in the row either exceeds 100 times the average number of entries in a row or is more than 4 times greater than the number of entries in any 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 that are taken from [26]:

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 $r = Ax - b$ is the residual, r_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

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

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 3.2, we summarize our findings for the complete test set. We present results with and without exploiting dense rows. For the former, 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. As well as the storage for the incomplete factors, when dense rows are exploited, we need 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

TABLE 3.2

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

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 $ratio_s$ of the number of entries in the matrices that are factorized to the number of entries in the preconditioner, that is,

$$ratio_s = (nnz(C_s) + md(md+1)/2) \,/\, (size\_ps + md(md+1)/2)\,.$$

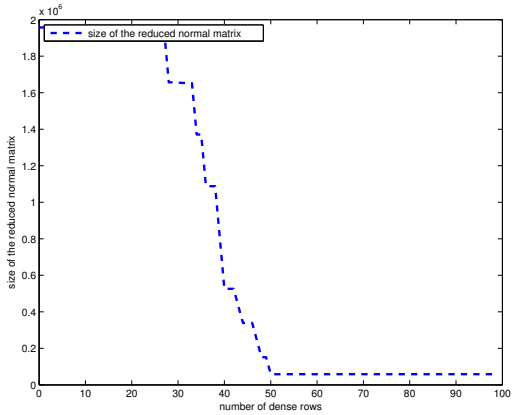

FIG. 3.1. *The number of entries in C_s for problem Meszaros/scsd8-2r as the number of dense rows that are removed from A increases.*

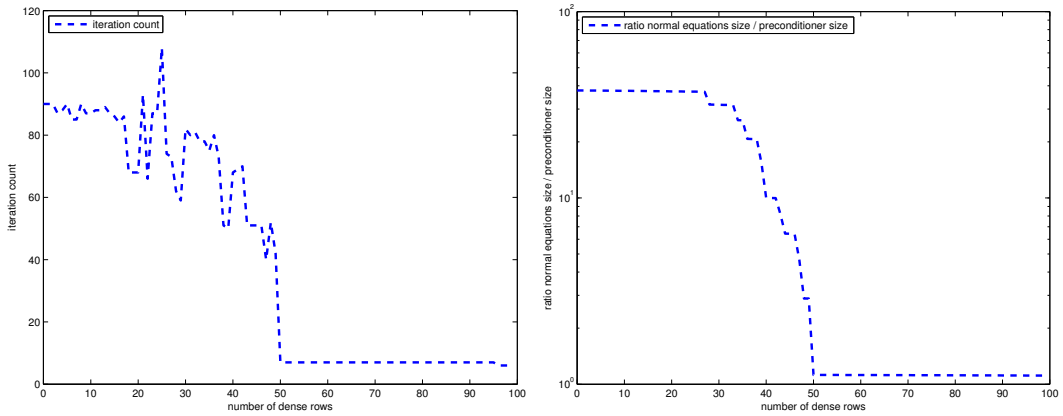

FIG. 3.2. *Problem Meszaros/scsd8-2r. Iteration counts (left), and $ratio_s$ (right) as the number of dense rows that are removed from A increases.*

The first example that we consider is problem Meszaros/scsd8-2r. Figure 3.1 illustrates the size of the reduced normal matrix C_s as the number of rows that are classified as dense increases. Then in Figures 3.2 and 3.3, we present the CGLS iteration counts, $ratio_s$, 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 3.2 significantly reduces the size of C_s; $ratio_s$ also decreases. We also see that the iteration counts can decrease before m_d reaches 50. This demonstrates that dense rows not only lead to high memory demands but their presence can reduce the IC factorization quality. 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 3.3 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

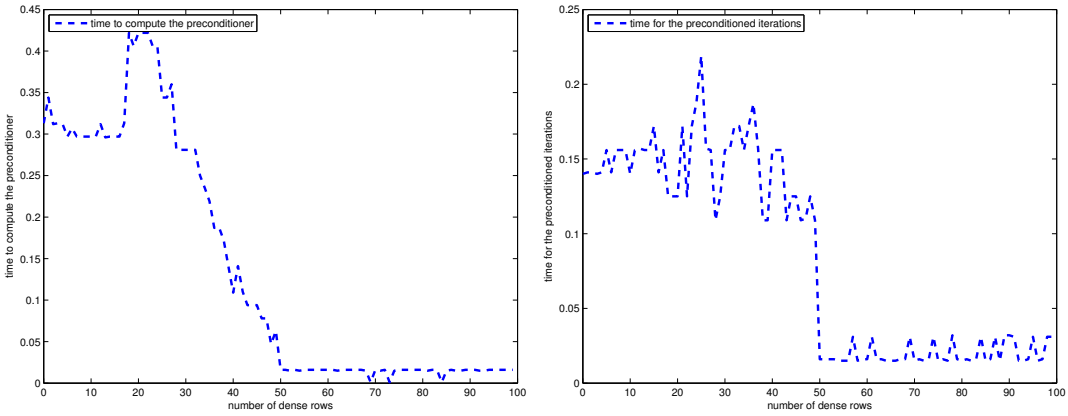

FIG. 3.3. *Problem Meszaros/scsd8-2r. Time to compute the preconditioner (left) and time for CGLS (right) as the number of dense rows that are removed from A is increased.*

than once with α increased each time (`HSL_MI35` takes care of this automatically) and this is reflected in the time. To investigate whether the problem can be solved more efficiently by applying a higher quality preconditioner, Figures 3.4 and 3.5 report results for `HSL_MI35` with the parameters $lsize$ and $rsize$ that control the number of entries in the incomplete factor $\tilde{L}_s$ from 5 to increased to 20. We see that the qualitative behavior remains the same and is consistent with our assumptions.

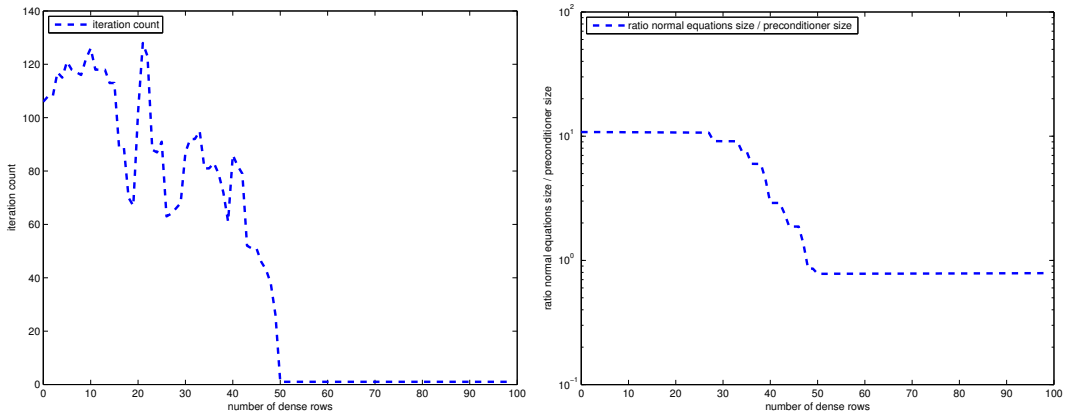

FIG. 3.4. *Problem Meszaros/scsd8-2r with $lsize = rsize = 20$. Iteration counts (left) and and $ratio_s$ (right) as the number of dense rows that are removed from A increases.*

3.4. Results for Mittelmann/stormg2_1000. Next we consider the large example Mittelmann/stormg2_1000; results are given in Figures 3.6 to 3.8. Again, the time for computing the preconditioner is influenced by the number of times the incomplete factorization is restarted 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 3.2 are removed.

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 following splitting

$$A = \begin{pmatrix} A_1 & A_2 \end{pmatrix} \equiv \begin{pmatrix} A_{s_1} & A_{s_2} \\ A_{d_1} & A_{d_2} \end{pmatrix}, \tag{4.1}$$

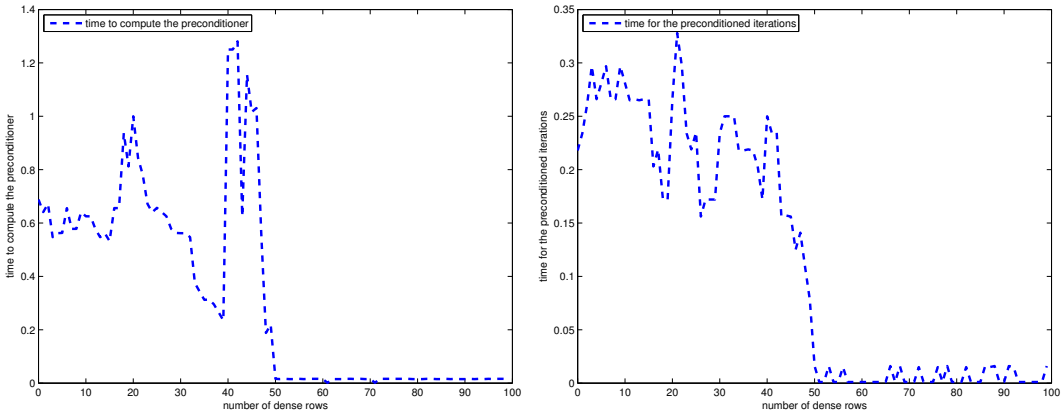

FIG. 3.5. *Problem Meszaros/scsd8-2r with lsize = rsize = 20. Time to compute the preconditioner (left), time for CGLS (right) as the number of dense rows that are removed from A increases.*

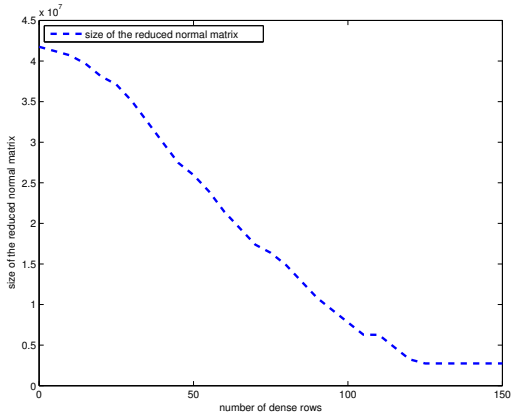

FIG. 3.6. *The number of entries in C_s for problem Mittelmann/stormg2_1000 as the number of dense rows that are removed from A increases.*

We have the following result that shows that the solution of the least squares problem can be expressed as a combination of partial solutions.

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

$$\min_z \|A_1 z - b\|_2 \tag{4.2}$$

and

$$\min_W \|A_1 W - A_2\|_F , \tag{4.3}$$

respectively. Then the solution $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ of problem (1.1) with its splitting consistent with (4.2) (that is, $x_1 \in R^{n_1}$, $x_2 \in R^{n_2}$) is given by

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} \xi - \Gamma x_2 \\ x_2 \end{pmatrix} \tag{4.4}$$

with

$$x_2 = (A_2^T A_2 - A_2^T A_1 \Gamma)^{-1} (A_2^T b - A_2^T A_1 \xi).$$

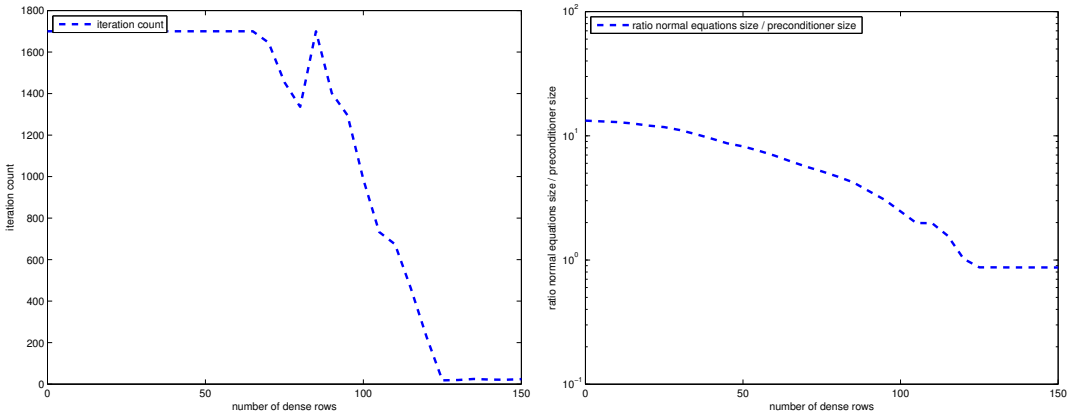

FIG. 3.7. *Problem Mittelmann/stormg2_1000. Iteration counts (left) and ratio$_s$ (right) as the number of dense rows that are removed from A increases.*

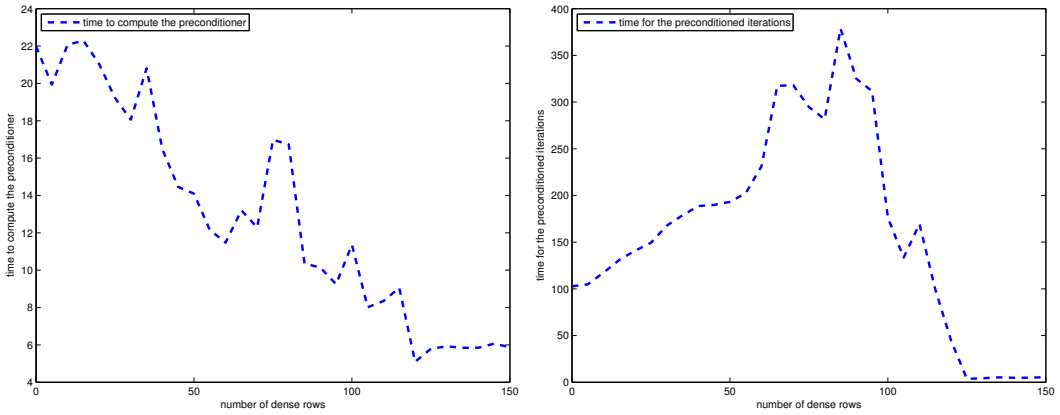

FIG. 3.8. *Problem Mittelmann/stormg2_1000. Time to compute the preconditioner (left) and time for CGLS (right) as the number of dense rows that are removed from A increases.*

Proof. From (4.2), we have

$$A^T A x = \begin{pmatrix} A_1^T A_1 & A_1^T A_2 \\ A_2^T A_1 & A_2^T A_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} A_1^T b \\ A_2^T b \end{pmatrix}. \tag{4.5}$$

Furthermore, $\xi = (A_1^T A_1)^{-1} A_1^T b$ and $\Gamma = (A_1^T A_1)^{-1} A_1^T A_2$. Premultiplying the first block row of (4.5) by $(A_1^T A_1)^{-1}$, the result follows. □

Theorem 4.1 implies a practical procedure for solving least squares problems with some dense rows: compute partial solutions ξ and Γ corresponding to the non-null column block A_1 using Algorithm 2.1 with $n_2 + 1$ right-hand sides and then use the result of Theorem 4.1. This requires forming and then solving with $(A_2^T A_2 - A_2^T A_1 \Gamma)$. Provided n_2 is small, this is inexpensive.

5. Concluding remarks. In this paper, we have looked at the problem of solving linear least squares problems in which the system matrix A has a number of dense rows. We propose a new approach that processes the dense rows separately within the CGLS 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 [38] and LSMR [16], could be employed. Likewise, other preconditioners could be used within our approach. Using problems from practical applications that each has between 1 and 121 rows that we classify as dense, our numerical experiments demonstrate the potential advantages of our proposed approach. In particular, we are able

to efficiently solve large problems that we were not able to solve using preconditioned CGLS without exploiting the dense rows. We remark that, in our experiments, we found the strategy based upon (2.8) gave better results than using the more complex scheme (2.5). We conjecture that this conclusion may be different if the preconditioner was based on a perturbed sparse direct solver rather 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 least squares 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 least squares problems. If A has a few rows that are identified as dense, they recommend that these rows are dropped before the QR factorization starts. They then 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 for our test examples that in general this gave very poor (or no) convergence, confirming the importance of incorporating the dense rows within the iterative solver.

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. *Numerical Linear Algebra with Applications*, 7(2):51–65, 2000.

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

[4] K. D. Andersen. A modified Schur complement method for handling dense columns in interior point methods for linear programming, 1996.

[5] O. D. Anderson. An improved approach to inverting the autocovariance matrix of a general mixed autoregressive moving average time process. *Australian J. Statistics*, 18(1-2):73–75, 1976.

[6] O. D. Anderson. On the inverse of the autocovariance matrix for a general moving average process. *Biometrika*, 63(2):391–394, 1976.

[7] H. Avron, E. Ng, and S. Toledo. Using perturbed QR factorizations to solve linear least-squares problems. *SIAM J. on Matrix Analysis and Applications*, 31(2):674–693, 2009.

[8] Å. Björck. A general updating algorithm for constrained linear least squares problems. *SIAM J. on Scientific and Statistical Computing*, 5(2):394–402, 1984.

[9] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[10] Å. Björck and I. S. Duff. A direct method for the solution of sparse linear least squares problems. *Linear Algebra and its Applications*, 34:43–67, 1980.

[11] Å. Björck, T. Elfving, and Z. Strakoš. Stability of conjugate gradient and Lanczos methods for linear least squares problems. *SIAM J. on Matrix Analysis and Applications*, 19(3):720–736, 1998.

[12] T. A. Davis and Y. Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1–28, 2011.

[13] P. S. R. Diniz. *Adaptive Filtering: Algorithms and Practical Implementation*. Springer, 4th ed. 2013 edition, 2012.

[14] I. S. Duff and J. A. Scott. Stabilized bordered block diagonal forms for parallel sparse solvers. *Parallel Computing*, 31:275–289, 2005.

[15] R. W. Farebrother. *Fitting Linear Relationships: A History of the Calculus of Observations 1750–1900*. Springer-Verlag New York, 1999.

[16] D. Fong and M. Saunders. LSMR: an iterative algorithm for sparse least-squares problems. *SIAM J. on Scientific Computing*, 33(5):2950–2971, 2011.

[17] C. F. Gauss and G. W. Stewart. *Theory of the Combination of Observations Least Subject to Errors*, volume 11 of *Classics in Applied Mathematics*. SIAM, Philadelphia, PA, 1995.

[18] A. George and M. T. Heath. Solution of sparse linear least squares problems using Givens rotations. *Linear Algebra and its Applications*, 34:69–83, 1980.

[19] A. George and E. Ng. SPARSPAK: Waterloo sparse matrix package user's guide for SPARSPAK-B. Research Report CS-84-37, Dept. of Computer Science, University of Waterloo, 1984.

[20] A. George and E. Ng. An implementation of Gaussian elimination with partial pivoting for sparse systems. *SIAM J. on Scientific and Statistical Computing*, 6(2):390–409, 1985.

[21] A. George and E. Ng. Symbolic factorization for sparse Gaussian elimination with partial pivoting. *SIAM J. on Scientific and Statistical Computing*, 8(6):877–898, 1987.

[22] 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, 1991.

[23] D. Goldfarb and K. Scheinberg. A product-form Cholesky factorization method for handling dense columns in interior point methods for linear programming. *Mathematical Programming, Series A*, 99:1–34, 2004.

[24] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications*, 60:545–557, 2015.

[25] 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, 2015.

[26] N. I. M. Gould and J. A. Scott. The state-of-the-art of preconditioners for sparse linear least squares problems. *ACM Transactions on Mathematical Software*, 2017. To appear.

[27] J. F. Grcar. Matrix stretching for linear equations. Technical Report SAND90-8723, Sandia National Laboratories, 1990.

[28] L. Guttman. Enlargement methods for computing the inverse matrix. *Annals of Mathematical Statistics*, 17:336–343, 1946.

[29] W. W. Hager. Updating the inverse of a matrix. *SIAM Review*, 31(2):221–239, 1989.

[30] M. T. Heath. Some extensions of an algorithm for sparse linear least squares problems. *SIAM J. on Scientific and Statistical Computing*, 3(2):223–237, 1982.

[31] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. of Research of the National Bureau of Standards*, 49:409–435, 1952.

[32] HSL. A collection of Fortran codes for large-scale scientific computation, 2016. `http://www.hsl.rl.ac.uk`.

[33] 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, October 1996.

[34] R. E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[35] I. J. Lustig, R. E. Marsten, and D. F. Shanno. On implementing Mehrotra's predictor-corrector interior-point method for linear programming. *SIAM J. on Optimization*, 2(3):435–449, 1992.

[36] E. Ng. A scheme for handling rank-deficiency in the solution of sparse linear least squares problems. *SIAM J. on Scientific and Statistical Computing*, 12(5):1173–1183, 1991.

[37] 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 and its Applications*, 394:1–24, 2005.

[38] C. C. Paige and M. A. Saunders. LSQR: an algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71, 1982.

[39] G. Peters and J. H. Wilkinson. The least squares problem and pseudo-inverse. *The Computer Journal*, 131:309–316, 1970.

[40] M. A. Saunders. Cholesky-based methods for sparse least squares: The benefits of regularization. Technical Report SOL 95-1, Department of Operations Research, Stanford University, 1995. In L. Adams and J. L. Nazareth (eds.), Linear and Nonlinear Conjugate Gradient-Related Methods, SIAM, Philadelphia, 92–100 (1996).

[41] W. Sautter. Fehleranalyse für die Gauss-Elimination zur Berechnung der Lösung minimaler Länge. *Numerische Mathematik*, 30(2):165–184, 1978.

[42] A. H. Sayed. *Fundamentals of adaptive filtering*. Wiley, 2003.

[43] J. A. Scott and M. Tůma. `HSL_MI28`: an efficient and robust limited-memory incomplete Cholesky factorization code. *ACM Transactions on Mathematical Software*, 40(4):Art. 24, 19 pages, 2014.

[44] J. A. Scott and M. Tůma. On positive semidefinite modification schemes for incomplete Cholesky factorization. *SIAM J. on Scientific Computing*, 36(2):A609–A633, 2014.

[45] 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, 1995.

[46] C. Sun. Parallel solution of sparse linear least squares problems on distributed-memory multiprocessors. *Parallel Computing*, 23(13):2075–2093, 1997.

[47] R. J. Vanderbei. Splitting dense columns in sparse linear systems. *Linear Algebra and its Applications*, 152:107–117, 1991.

[48] M. A. Woodbury. *The Stability of Out-Input Matrices*. Chicago, Ill., 1949.

[49] M. A. Woodbury. *Inverting modified matrices*. Statistical Research Group, Memo. Rep. no. 42. Princeton University, Princeton, N. J., 1950.

[50] M. H. Wright. Interior methods for constrained optimization. In *Acta Numerica, Volume 1*, pages 341–407. Cambridge University Press, 1992.