Science & Technology Facilities Council

Preprint

RAL-P-2017-004

A Schur complement approach to preconditioning sparse linear least-squares problems with some dense rows

J Scott, M Tuma

April 2017

Submitted for publication in Numerical Algorithms

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

A SCHUR COMPLEMENT APPROACH TO PRECONDITIONING SPARSE LINEAR LEAST-SQUARES PROBLEMS WITH SOME DENSE ROWS

JENNIFER SCOTT* AND MIROSLAV TŮMA†

Abstract. The effectiveness of sparse matrix techniques for directly solving large-scale linear least-squares problems is severely limited if the system matrix A has one or more nearly dense rows. In this paper, we partition the rows of A into sparse rows and dense rows (A_s and A_d) and apply the Schur complement approach. A potential difficulty is that the reduced normal matrix $A_s^T A_s$ is often rank-deficient, even if A is of full rank. To overcome this, we propose explicitly removing null columns of A_s and then employing a regularization parameter and using the resulting Cholesky factors as a preconditioner for an iterative solver applied to the symmetric indefinite reduced augmented system. We consider complete factorizations as well as incomplete Cholesky factorizations of the shifted reduced normal matrix. Numerical experiments are performed on a range of large least-squares problems arising from practical applications. These demonstrate the effectiveness of the proposed approach when combined with either a sparse parallel direct solver or a robust incomplete Cholesky factorization algorithm.

Key words. large-scale linear least-squares problems, dense rows, augmented system, Schur complement, iterative solvers, preconditioning, Cholesky factorization, incomplete factorizations.

AMS subject classifications.

1. Introduction.

We are interested in solving the following linear least-squares problem:

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

where $A \in \Re^{m\times n}$ ($m \geq n$) and $b \in \Re^m$. The most commonly-used approach is to work with the mathematically equivalent $n \times n$ *normal equations*

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

where, provided A has full column rank, the *normal matrix* C is symmetric and positive definite. Our focus is on the case where the system matrix A is large and sparse but has a number of "dense" rows (that is, rows that contain significantly more entries than the other rows, although the number of entries in each such row may be less than n). Just a single dense row is sufficient to cause catastrophic fill in C and thus for the factors of a Cholesky or QR factorization to be dense. In practice, for large-scale problems this means that it may not be possible to use a direct solver since the memory demands can be prohibitive. Moreover, if an incomplete factorization is used as a preconditioner for an iterative solver such as LSQR [29, 30] or LSMR [13] applied to the normal equations, the error in the factorization can be so large as to prohibit its effectiveness as a preconditioner; this was recently observed in the study by Gould and Scott [18]. The effects of the presence of dense rows has long been recognised as a fundamental difficulty in the solution of sparse least-squares problems; see, for example, [2, 5, 8, 14, 16, 40, 41, 42].

Let us assume that the rows of A are partitioned into two parts: rows that are sparse and those that are considered dense. We also assume conformal partitioning 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

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

* 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. 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, 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.

In this paper, we exploit the fact that solving (1.4) is equivalent to solving the larger $(m+n)\times(m+n)$ *augmented system*

$$\begin{pmatrix} I_{m_s} & & A_s \\ & I_{m_d} & A_d \\ A_s^T & A_d^T & 0 \end{pmatrix} \begin{pmatrix} r_s \\ r_d \\ x \end{pmatrix} = \begin{pmatrix} b_s \\ b_d \\ 0 \end{pmatrix}, \tag{1.5}$$

where

$$r = \begin{pmatrix} r_s \\ r_d \end{pmatrix} = \begin{pmatrix} b_s \\ b_d \end{pmatrix} - \begin{pmatrix} A_s \\ A_d \end{pmatrix} x$$

is the residual vector. Here and elsewhere I_k denotes the $k \times k$ identity matrix. The system (1.5) is symmetric indefinite and so, if there is sufficient memory available, a sparse direct solver that incorporates the use of numerical pivoting for stability can be used (well-known examples include `MA57` [12] and `HSL_MA97` [20] from the HSL mathematical software library [21], MUMPS [27] and WSMP [43]). Employing a general-purpose sparse solver ignores the block structure, although its use of a sparsity-preserving ordering (such as a variant of minimum degree or nested dissection) will tend to lead to the dense rows being eliminated last [11]. An alternative approach is to perform a block elimination to reduce the problem from a 3-block saddle-point system to a 2-block system. This reduced augmented matrix can be factorized using a sparse indefinite solver, but this would again ignore the structure. Instead, we use the so-called Schur complement method (see, for example, [15, 26, 34]) that exploits the structure by performing a sparse Cholesky factorization of $A_s^T A_s$, forming an $m_d \times m_d$ dense Schur complement matrix and factorizing it using a dense Cholesky factorization; using the sparse and dense factors to solve a number of triangular systems completes the solution process. This has the advantage of using Cholesky factorizations that, because they do not involve numerical pivoting, are more efficient (especially in parallel) than an indefinite factorization. Moreover, it can be easily incorporated into the normal equations approach.

In practice, even if A is of full rank, A_s is often rank deficient (indeed, it may have null columns). In this case, a Cholesky factorization of $A_s^T A_s$ will break down. To overcome this, we propose removing null columns and employing a regularization parameter and using the resulting Cholesky factors as a preconditioner for an iterative solver applied to the reduced augmented system. For large problems, even if A_s is sparse, memory limitations can mean that it is not possible to use a sparse direct solver. Thus we also consider using incomplete Cholesky factorizations combined with an iterative solver.

The outline of the rest of the paper is as follows. In Section 2, we recall the Schur complement approach and, in particular, we look at regularization and propose using the factors of the reduced regularized matrix $A_s^T A_s + \alpha I$ to obtain a block preconditioner. The use of a limited memory incomplete Cholesky factorization is also discussed. Section 3 introduces our numerical experiments. Computational results for complete and incomplete Cholesky factorizations are given in Sections 4 and 5, respectively. Concluding remarks are made in Section 6.

2. Schur complement method.

2.1. Schur complement method with direct solvers. An alternative approach to applying a direct solver to either the (dense) normal equations or the augmented system (1.5) is to eliminate the first m_s rows and columns of (1.5) to obtain a reduced 2-block system of order $(m_d+n)\times(m_d+n)$ that can be written in the form

$$K \begin{pmatrix} x \\ r_d \end{pmatrix} = \begin{pmatrix} -A_s^T b_s \\ b_d \end{pmatrix}, \qquad K = \begin{pmatrix} -C_s & A_d^T \\ A_d & I_{m_d} \end{pmatrix}. \tag{2.1}$$

Here the $n \times n$ matrix $C_s = A_s^T A_s$ is termed the *reduced normal matrix*. We will refer to (2.1) as the *reduced augmented system*. Provided A_s has full column rank, C_s is symmetric positive definite and, if the partitioning (1.3) is such that all the rows of A_s are sparse, C_s is generally significantly sparser than the

original normal matrix C. Let $C_s = L_s L_s^T$ be the Cholesky factorization of C_s. Using this yields a block factorization

$$K = \begin{pmatrix} L_s & \\ B_d & I_{m_d} \end{pmatrix} \begin{pmatrix} -I_n & \\ & S_d \end{pmatrix} \begin{pmatrix} L_s^T & B_d^T \\ & I_{m_d} \end{pmatrix}, \tag{2.2}$$

where B_d and the Schur complement matrix S_d are given by

$$L_s B_d^T = -A_d^T \tag{2.3}$$

$$S_d = I_{m_d} + B_d B_d^T. \tag{2.4}$$

Since L_s is lower triangular, solving (2.3) for B_d^T is straightforward. B_d will normally be dense and hence S_d will be dense and symmetric positive definite. Once we have L_s, B_d and S_d, we can solve (2.1) by solving

$$\begin{pmatrix} -L_s & \\ -B_d & S_d \end{pmatrix} \begin{pmatrix} y_s \\ y_d \end{pmatrix} = \begin{pmatrix} -A_s^T b_s \\ b_d \end{pmatrix}, \tag{2.5}$$

followed by

$$\begin{pmatrix} L_s^T & B_d^T \\ & I_{m_d} \end{pmatrix} \begin{pmatrix} x \\ r_d \end{pmatrix} = \begin{pmatrix} y_s \\ y_d \end{pmatrix}. \tag{2.6}$$

Using $r_d = y_d$, this requires us to solve

$$L_s y_s = A_s^T b_s, \tag{2.7}$$

$$S_d r_d = b_d + B_d y_s, \tag{2.8}$$

$$L_s^T x = y_s - B_d^T r_d. \tag{2.9}$$

Again, (2.7) and (2.9) are triangular systems and hence straightforward to solve. Solving (2.8) requires the Cholesky factorization of the $m_d \times m_d$ dense matrix S_d. Observe that in the case of a single dense row ($m_d = 1$) this is trivial. In general, the LAPACK routine `_potrf` can be used to factorize S_d and then routine `_potrs` employed to solve (2.8).

2.2. Scaling and stability. Poorly scaled entries in A_s may result in the block elimination of the first m_s rows and columns of (1.5) being unstable. To overcome this, 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. The elimination of the first m_s rows and columns is thus stable (the pivots can be chosen in order from the main diagonal and they satisfy the criteria for complete pivoting). However, this does not guarantee stability of the next step. The entries of the factor L_s of the positive definite C_s can be small leading to large entries in B_d and hence large entries in S_d.

If instability is detected then a general-purpose symmetric indefinite sparse solver that incorporates numerical pivoting can be applied to solve the reduced augmented system (2.1). Whilst this offers a robust approach, it has the disadvantage that the block structure within (2.1) is not exploited. Furthermore, if A_s is fixed and the interest lies in adding new rows A_d, the factorization must be redone in its entirety for each A_d. Thus, in the next subsection, we propose an alternative approach to maintain stability.

We assume throughout the remainder of our discussion and in all our numerical experiments that A has been prescaled but omit D to simplify the notation.

2.3. Removal of null columns. In practice, when A is partitioned, the sparse part A_s often contains a (small) number of null columns. It is possible to explicitly remove these columns, as we now show. Let

A have full column rank and assume A_s has n_2 null columns with $n_2 \ll n$. Assuming these columns are permuted to the end, we can split A into the form

$$A = \begin{pmatrix} A_1 & A_2 \end{pmatrix} \equiv \begin{pmatrix} A_{s_1} & 0 \\ A_{d_1} & A_{d_2} \end{pmatrix} \tag{2.10}$$

with $A_1 \in R^{m \times n_1}$ and $A_2 \in R^{m \times n_2}$ ($n = n_1 + n_2$). The following result from [39] shows that the solution of the least-squares problem can be expressed as a combination of partial solutions.

LEMMA 2.1. *Let the columns of A be split as in (2.10) and let $z \in R^{n_1}$ and $W \in R^{n_1 \times n_2}$ be the solutions to the problems*

$$\min_z \|A_1 z - b\|_2 \tag{2.11}$$

and

$$\min_W \|A_1 W - A_2\|_F , \tag{2.12}$$

respectively. Then the solution $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ of the least-squares problem (1.1) is given by

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} z - W x_2 \\ x_2 \end{pmatrix} \tag{2.13}$$

with

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

The least-squares problem (2.11) again has m_d dense rows and thus can be solved using the Schur complement method. The factorizations of the reduced normal matrix $C_{s_1} = A_{s_1}^T A_{s_1}$ and the Schur complement matrix S_{d_1} can then be used to solve (2.12). Since $n_2 \ll n$, this step is inexpensive.

2.4. Regularization and preconditioning. While Lemma 2.1 provides a way of removing null columns, it is possible that even if A is of full column rank, A_s (or A_{s_1} after the removal of null columns) is rank deficient (or is close to rank deficient). To simplify notation, in our discussion we use A_s but this may be replaced by A_{s_1} if A_s contains null columns. If A_s is rank deficient, the reduced normal matrix C_s is positive semi definite and a Cholesky factorization breaks down (that is, a very small or a non positive pivot is encountered). If break down occurs, we employ a shift $\alpha > 0$ and compute a Cholesky factorization of the shifted matrix

$$C_s(\alpha) = A_s^T A_s + \alpha I_n. \tag{2.14}$$

The shift α is also referred to as a *Tikhonov regularization* parameter. The choice of α should be related to the smallest eigenvalue of $A_s^T A_s$, but this information is not readily available. Clearly, it is always possible to find an α so that $C_s(\alpha)$ is positive definite; if the initial choice α is too small (that is, C_α is positive semi definite), it may be necessary to restart the factorization more than once, increasing α on each restart until breakdown is avoided. Use of a shift was discussed by Lustig, Marsten and Shanno [24] (see also [1, 2]). It was observed that careful and often substantial use of iterative refinement to compute each column of B_d^T was required. However, we adopt a different approach in which we use the factorization of (2.14) to obtain a preconditioner for the system (2.1).

If $\alpha > 0$ then using (2.14) the computed solution is the solution of (2.1) with K replaced by

$$K(\alpha) = \begin{pmatrix} -C_s(\alpha) & A_d^T \\ A_d & I_{m_d} \end{pmatrix}.$$

Thus the computed value of the least-squares objective may differ from the optimum for the original problem. Having solved the regularized problem we want to recover the solution of the original problem. Following Scott [36], we propose doing this by using the factors of $K(\alpha)$ as a preconditioner for an iterative method applied to (2.1).

Let the Cholesky factorization of $C_s(\alpha)$ be $L_s(\alpha)L_s(\alpha)^T$. For $\alpha > 0$, this is an approximate factorization of C_s, that is, $C_s \approx L_s(\alpha)L_s(\alpha)^T$. More generally, let

$$C_s \approx \tilde{L}_s \tilde{L}_s^T, \tag{2.15}$$

where $\tilde{L}_s$ is lower triangular. We are interested in the case $\tilde{L}_s = L_s(\alpha)$ but our main focus is where $\tilde{L}_s$ is an incomplete Cholesky (IC) factor, that is, one that contains fewer entries than occur in a complete factorization. For very large systems, computing and factorizing C_s (or $C_s(\alpha)$) is prohibitively expensive in terms of memory and/or computational time. Over the last fifty or more years, IC factorizations have represented an important tool in the armoury of preconditioners for the numerical solution of large sparse symmetric positive-definite linear systems of equations; for an introduction and overview see, for example, [6, 32, 38] and the long lists of references therein. Here we consider preconditioning the symmetric indefinite system (2.1) using an factorization of the form (2.15) and exploiting the block structure of (2.1).

The right-preconditioned reduced augmented system is

$$KM^{-1}\begin{pmatrix} w_s \\ w_d \end{pmatrix} = \begin{pmatrix} -A_s^T b_s \\ b_d \end{pmatrix}, \qquad M\begin{pmatrix} x \\ r_d \end{pmatrix} = \begin{pmatrix} w_s \\ w_d \end{pmatrix}, \tag{2.16}$$

where M is the chosen preconditioner. Using (2.15), we obtain a preconditioner M given by

$$M = \begin{pmatrix} \tilde{L}_s & \\ \tilde{B}_d & I_{m_d} \end{pmatrix} \begin{pmatrix} -I_n & \\ & \tilde{S}_d \end{pmatrix} \begin{pmatrix} \tilde{L}_s^T & \tilde{B}_d^T \\ & I_{m_d} \end{pmatrix}, \tag{2.17}$$

where $\tilde{B}_d$ and $\tilde{S}_d$ are given by (2.3) and (2.4) with the complete factor L_s replaced by the incomplete one $\tilde{L}_s$, that is,

$$\tilde{L}_s \tilde{B}_d^T = -A_d^T \tag{2.18}$$

$$\tilde{S}_d = I_{m_d} + \tilde{B}_d \tilde{B}_d^T. \tag{2.19}$$

Applying this preconditioner requires a number of steps that are analogous to (2.7)–(2.9). In particular, a dense $m_d \times m_d$ symmetric positive-definite system of the form $\tilde{S}_d y_d = u_d$ must be solved. We again assume that LAPACK may be used. If m_d is so large that this is too expensive, an incomplete factorization of $\tilde{S}_d$ could be used. Algorithm 1 outlines the steps required for each application of the preconditioner. We see that it involves a triangular solve with $\tilde{L}_s$ and with $\tilde{L}_s^T$, calls to the BLAS routine `_gemv` for steps 2 and 4, and triangular solves using the factors of $\tilde{S}_d$.

Algorithm 1 Application of the block factorization preconditioner, that is, compute $M^{-1}z = y$.

Input: $\tilde{L}_s$, $\tilde{B}_d$, the Cholesky factors of $\tilde{S}_d$, and the vector $z = \begin{pmatrix} z_s \\ z_d \end{pmatrix}$.

Output: $y = \begin{pmatrix} y_s \\ y_d \end{pmatrix} = M^{-1}z$.

1: Solve $\tilde{L}_s u_s = -z_s$.
2: Compute $u_d = z_d + \tilde{B}_d u_s$.
3: Use the Cholesky factors of $\tilde{S}_d$ to solve $\tilde{S}_d y_d = u_d$.
4: Form $u_s = u_s - \tilde{B}_d^T y_d$.
5: Solve $\tilde{L}_s^T y_s = u_s$.

Many different IC factorizations have been proposed. Although they may be considered to be general purpose, most are best suited to solving particular classes of problems. For example, level-based methods are often most appropriate for systems with underlying structure, such as from finite element or finite difference applications. Here we use the limited memory based approach of Scott and Tůma [37, 38], that has been shown in [18] to result in effective preconditioners for a wide range of least-squares problems. The basic scheme employs a matrix factorization of the form

$$C_s \approx (\tilde{L}_s + R)(\tilde{L}_s + R)^T, \tag{2.20}$$

where $\tilde{L}_s$ is the lower triangular matrix with positive diagonal entries that is used for preconditioning and R is a strictly lower triangular matrix with small entries that is used to stabilize the factorization process but is then discarded (it is not used as part of the preconditioner). The user specifies the maximum number of entries in each column of $\tilde{L}_s$ and R. At each step j of the incomplete factorization process, the largest entries are kept in column j of $\tilde{L}_s$, the next largest are kept in column j of R, and the remainder (the smallest entries) are dropped. In practice, C_s is optionally preordered and scaled and, if necessary, shifted to avoid breakdown of the factorization (which occurs if a non positive pivot is encountered) [25].

3. Numerical experiments. In this section, we present numerical results to illustrate potential of the Schur complement approach and, in particular, demonstrate that it allows us to solve some problems that are intractable if dense rows are ignored. Results are included for direct solvers and for iterative solvers that can be used to solve very large problems.

3.1. Test environment. The characteristics of the machine used to perform our tests are given in Table 3.1. All software is written in Fortran and all reported timings are elapsed times in seconds. In

TABLE 3.1
Test machine characteristics

CPU	Two Intel Xeon E5620 quadcore processors
Memory	24 GB
Compiler	gfortran version 4.8.4 with options -O3 -fopenmp
BLAS	Intel MKL

our experiments, we employ the Cholesky sparse direct solver `HSL_MA87` [19] for positive-definite systems and `HSL_MA97` [20] for general sparse symmetric indefinite systems; both employ OpenMP and are run in parallel, using 4 processors. Both solvers are run with a nested dissection ordering [23]. Sparse matrix-vector products required by the iterative solvers are performed in parallel using the Intel Mathematics Kernel Library (MKL) routines; no attempt is made to parallelize the iterative methods themselves. In each test, we impose a time limit of 600 seconds per problem and for the iterative methods, the number of iterations is limited to 100,000.

Following Gould and Scott [18], we want the computed residual r to satisfy

$$ratio(r) < \delta \quad \text{with} \quad ratio(r) = \frac{\|A^T r\|_2/\|r\|_2}{\|A^T b\|_2/\|b\|_2}. \tag{3.1}$$

We set the tolerance δ to 10^{-6} and in our experiments, the right-hand side b is taken to be the vector of 1's. As the preconditioner (2.17) is indefinite, it needs to be used with a general non symmetric iterative method such as GMRES [33]; we use right preconditioned restarted GMRES. Since GMRES is applied to the reduced augmented system matrix K, the stopping criteria is applied to K. With the available implementations of GMRES, it is not possible during the computation to check whether (3.1) is satisfied; this can only be checked once GMRES has terminated. Instead, we use the scaled backward error

$$\frac{\|K \begin{pmatrix} x^{(k)} \\ r_d^{(k)} \end{pmatrix} - \begin{pmatrix} -A_s^T b_s \\ b_d \end{pmatrix} \|_2}{\| \begin{pmatrix} -A_s^T b_s \\ b_d \end{pmatrix} \|_2} < \tilde{\delta}, \tag{3.2}$$

where $\begin{pmatrix} x^{(k)} \\ r_d^{(k)} \end{pmatrix}$ is the computed solution of (2.1) on the kth step. In our experiments, we set $\tilde{\delta} = 10^{-7}$. With this choice, in most of our experiments (3.1) is satisfied with $\delta = 10^{-6}$.

3.2. Test set 1. Our test problems are taken from the CUTEst linear programme set [17] and the UFL Sparse Matrix Collection [9]. In each case, the matrix is "cleaned" (duplicates are summed, out-of-range entries and explicit zeros are removed along with any null rows or columns). In our experiments, we use the following definition for a dense row of A: given ρ ($0 < \rho \leq 1$), row i of A is defined to be dense if the percentage of entries in row i is at least ρ.

Our first test set is given in Table 3.2. The problems were chosen because they have at least one row that is more than 10% dense. They are also difficult problems to solve (see [18]); at least three of the problems are rank deficient. An estimate of the rank was computed by running the sparse symmetric indefinite solver `HSL_MA97` on the augmented system (1.5) (with the pivot threshold parameter set to 0.5); for problems `12month1` and `PDE1` there was insufficient memory to do this.

TABLE 3.2

Statistics for Test Set 1. m, n and $nnz(A)$ are the row and column counts and the number of nonzeros in A. nullity is the estimated deficiency in the rank of A, rdensity(A) is the largest ratio of number of nonzeros in a row of A to n over all rows, m_j ($j = 10, 20, 30, 40, 50$) is the number of rows of A with at least j% entries, and density(C) is the ratio of the number of entries in C to n^2. − denotes insufficient memory to compute the statistic.

Problem	m	n	$nnz(A)$	nullity	rdensity(A)	m_{10}	m_{20}	m_{30}	m_{40}	m_{50}	density(C)
`Trec14`	15904	3159	2872265	0	0.791	2664	1232	649	346	150	9.32×10^{-1}
`Maragal_6`	21251	10144	537694	516	0.586	68	68	30	21	0	7.49×10^{-1}
`Maragal_7`	46845	26525	1200537	2046	0.360	85	43	21	0	0	3.10×10^{-1}
`scsd8-2r`	60550	8650	190210	0	0.100	40	0	0	0	0	5.22×10^{-2}
`PDE1`	271792	270595	990587	-	0.670	1	1	1	1	1	-
`12month1`	872622	12471	22624727	-	0.274	284	4	0	0	0	6.87×10^{-1}

TABLE 3.3

The effects of varying the row density parameter ρ on the number m_d of rows that are classed as dense and the density of C_s (the ratio of the number of entries in C_s to n^2).

Identifier	m	n	ρ	m_d	$density(C_s)$
`Trec14`	15904	3159	0.005	12643	2.38×10^{-2}
			0.010	9676	8.52×10^{-2}
			0.050	4467	6.17×10^{-1}
			0.100	2664	8.31×10^{-1}
`Maragal_6`	21251	10144	0.005	2923	6.22×10^{-4}
			0.010	823	1.93×10^{-2}
			0.100	68	5.49×10^{-2}
`Maragal_7`	46845	26525	0.001	4668	2.15×10^{-4}
			0.005	687	9.02×10^{-3}
			0.010	108	1.70×10^{-2}
			0.100	85	1.78×10^{-2}
`scsd8-2r`	60550	8650	0.050	50	1.44×10^{-3}
			0.100	40	1.39×10^{-2}
`PDE1`	271792	270595	0.660	1	4.52×10^{-5}
`12month1`	872622	12471	0.010	43951	1.10×10^{-1}
			0.050	3641	5.66×10^{-1}
			0.100	284	6.56×10^{-1}

In Table 3.3, we report the effects of varying the parameter ρ that controls which rows are classified as dense. Increasing ρ reduces the number m_d of dense rows but increases the density of the reduced normal matrix C_s. Problem `PDE1` has only one row that is classified as dense for $\rho \in [0.001, 0.66]$. We see that for `12month1` and `Trec14`, ρ has to be very small for C_s to be sparse but, in this case, m_d is large compared

to m. For the `Maragal` problems, C_s is highly sparse if approximately 10% of the rows are classified as dense.

3.3. Test set 2. For our second test set, we take some of the CUTEst and UFL examples that do not initially contain dense rows and append some rows. This allows us to explore the effect of varying the number of dense rows as well as the density of these rows. The problems are listed in Table 3.4; these problems are all of full rank. When appending rows, the pattern of each such row is generated randomly with the requested density and the values of the entries are random numbers in $[-1, 1]$.

For our solvers, the number of entries $nnz(C)$ in the normal matrix C can be at most `huge(1)` ($\approx 2 \times 10^9$) where `huge` is the Fortran intrinsic function. If we add a single row with density $\rho \geq 0.1$ to each of the matrices A_s in the lower part of Table 3.4 then $nnz(C)$ exceeds this limit. Thus for these examples and our current software, we cannot use any approach that requires the normal matrix to be computed.

TABLE 3.4

Statistics for Test Set 2. m_s, n and $nnz(A_s)$ are the row and column counts and the number of nonzeros in A_s. rdensity(A_s) is the largest ratio of number of nonzeros in a row of A_s to n over all rows, and density(C_s) is the ratio of the number of entries in C_s to n^2.

Problem	m_s	n	$nnz(A_s)$	rdensity(A_s)	density(C_s)
`IG5-15`	11369	6146	323509	1.95×10^{-2}	1.52×10^{-1}
`psse0`	26722	11028	102432	3.63×10^{-4}	5.88×10^{-4}
`graphics`	29493	11822	117954	3.38×10^{-4}	5.91×10^{-4}
`WORLD`	67147	34506	198883	4.64×10^{-4}	4.89×10^{-4}
`STAT96V3`	1113780	33841	3317736	3.55×10^{-4}	3.58×10^{-4}
`STORMG21K`	1377306	526185	3459881	1.93×10^{-3}	3.00×10^{-4}
`GL7d20`	1911124	1437546	29893084	2.99×10^{-5}	2.23×10^{-4}
`CONT11_L`	1961394	1468599	5382999	4.77×10^{-6}	8.38×10^{-6}
`LargeRegFile`	2111154	801374	4944201	4.99×10^{-6}	9.93×10^{-6}
`relat9`	9746232	274667	38955420	1.46×10^{-5}	5.09×10^{-4}

4. Direct solver results. Our first experiments look at the effectiveness of the Schur complement approach using the Cholesky direct solver `HSL_MA87` to factorize the reduced normal matrix C_s. We compare this with using `HSL_MA87` to solve the original normal matrix C (1.2) without partitioning A into sparse and dense parts. If the Cholesky factorization of C breaks down because of a non positive pivot, we factorize the shifted normal matrix $C + \alpha I_n = L(\alpha)L(\alpha)^T$ and use the factors as a preconditioner for the iterative method LSMR [13] (see [36]). In our tests, we set $\alpha = 10^{-12}$.

Results are given in Tables 4.1 and 4.2 for the normal equations and Schur complement approaches, respectively. For problem `PDE1`, the number of entries in the normal matrix C exceeds `huge(1)` so we cannot form C and use the direct solver `HSL_MA87`. The reported times T_f and T_p for computing the Cholesky factorization of C (Table 4.1) and the block factorization preconditioner (Table 4.2) include the time to form C and C_s, respectively. For problem `12month1`, forming C (or C_s) accounts for approximately half the total time. Note we could try to employ a solver that avoids storing C in main memory. The out-of-core solver `HSL_MA77` [31] only requires one column of C at a time and both matrix and factor data are written to files on disk, thus minimising memory requirements. However, `HSL_MA77` is for sparse matrices and when n is large and C is dense the amount of in-core memory available is still exceeded.

The results reported in Table 4.2 illustrate that the Schur complement approach is successful but to achieve savings compared to using the normal equations in terms of the size of the factors and/or the computation time, m_d must be small compared to n and the reduced normal matrix C_s must be sparse. For the `Maragal` problems, we are able to choose the density ρ to achieve this. In Table 4.2, we include the time T_K to form the reduced normal matrix K and then factorize and solve (2.1) using the symmetric indefinite solver `HSL_MA97`; we also report the number $nnz(L_K)$ of entries in the `HSL_MA97` factors of K. A comparison of the times in the T_{total} and T_K columns illustrates the savings offered by

TABLE 4.1

Results for Test Set 1 of running the Cholesky direct solver HSL_MA87 *on the normal equations (without exploiting dense rows), using LSMR for refinement. $nnz(L)$ denotes the number of entries in the Cholesky factor L of C and Its is the number of LSMR iterations. T_f, T_s and T_{total} denote the times (in seconds) to compute the normal matrix and factorize it, to run LSMR and the total time. − denotes unable to form normal matrix C.*

Identifier	m	n	$nnz(L)$	Its	T_f	T_s	T_{total}
Trec14	15904	3159	4.85×10^6	1	4.79	0.03	4.82
Maragal_6	21251	10144	4.96×10^7	3	13.1	0.12	13.2
Maragal_7	46845	26525	1.43×10^8	4	37.8	0.40	38.2
scsd8-2r	60550	8650	1.20×10^7	0	0.88	0.00	0.88
PDE1	271792	270595	-	-	-	-	-
12month1	872622	12471	7.27×10^7	1	42.5	0.35	42.9

TABLE 4.2

Results for Test Set 1 of solving the reduced augmented system (2.1) using the Schur complement approach and the Cholesky direct solver HSL_MA87*. ρ is the row density parameter. density(C_s) is the ratio of the number of entries in the reduced normal matrix C_s to n^2, $nnz(L)$ is total number of entries in the factors (that is, $nnz(L_s) + m_d(m_d+1)/2$), Its is the number of GMRES iterations. T_p, T_s and T_{total} denote the times (in seconds) to compute the preconditioner, to run GMRES and the total time. T_K is the time to form the reduced augmented matrix and solve using the sparse symmetric indefinite direct solver* HSL_MA97 *and $nnz(L_K)$ is the number of entries in* HSL_MA97 *factors.*

Identifier	ρ	m_d	$density(C_s)$	$nnz(L)$	Its	T_p	T_s	T_{total}	$nnz(L_K)$	T_K
Trec14	0.050	4467	6.17×10^{-1}	1.48×10^7	1	4.09	0.06	4.14	7.97×10^6	7.18
	0.100	2664	8.31×10^{-1}	8.50×10^6	1	2.48	0.04	2.52	7.13×10^6	6.59
	0.200	1234	9.09×10^{-1}	5.73×10^6	1	2.41	0.02	2.43	6.76×10^6	5.63
Maragal_6	0.001	2923	6.22×10^{-4}	4.39×10^6	3	1.78	0.16	1.94	1.78×10^7	2.54
	0.010	823	1.93×10^{-2}	2.46×10^7	3	2.77	0.12	2.88	2.95×10^7	5.58
	0.100	68	5.49×10^{-2}	4.30×10^7	3	4.61	0.15	4.76	4.15×10^7	18.3
Maragal_7	0.001	4668	2.15×10^{-4}	1.12×10^7	4	9.51	0.69	10.2	6.15×10^7	12.9
	0.005	687	9.02×10^{-3}	9.04×10^7	3	11.7	0.36	12.1	1.10×10^8	26.5
scsd8-2r	0.050	50	1.44×10^{-3}	9.20×10^4	3	0.03	0.00	0.03	5.77×10^5	0.03
	0.100	40	1.39×10^{-2}	5.40×10^6	3	0.30	0.04	0.34	5.25×10^6	0.29
PDE1	0.100	1	4.52×10^{-5}	2.04×10^7	0	1.24	0.03	1.27	2.07×10^7	5.02
12month1	0.050	3641	5.66×10^{-1}	7.74×10^7	3	49.3	0.70	50.0	8.91×10^7	61.4
	0.100	284	6.56×10^{-1}	7.23×10^7	3	42.1	0.54	42.7	7.53×10^7	70.0

the Schur complement approach that result from being able to exploit a Cholesky solver. Observe that although the symmetric indefinite solver HSL_MA97 ignores the block structure of K, as already noted, the sparsity-preserving nested dissection ordering it computes prior to the numerical factorization orders the dense rows last and thus the difference between $nnz(L)$ and $nnz(L_K)$ is generally relatively small. Furthermore, HSL_MA97 is able to take advantage of any zeros in the "dense" rows. If the number m_d of dense rows is not small, $nnz(L)$ is dominated by the storage needed for the dense factors of S_d (2.4) and $nnz(L)$ can then exceed $nnz(L_K)$; this is illustrated by problem Trec14.

5. Iterative method results. A software package HSL_MI35 that implements the limited memory IC algorithm outlined in Section 2.4 for the normal equations has been developed for the HSL library. We employ this package in our experiments. Note that it handles ordering for sparsity and scaling and also automatically selects the shift α. We use the default settings and set the parameters $lsize$ and $rsize$ that control the maximum number of entries in each column of the factors to 20; see [37] for more details of the parameters.

5.1. Results for Test Set 1. Table 5.1 presents results for running LSMR on the normal equations (1.2) using the IC preconditioner. Here the density of the rows is ignored. In Table 5.2 results are given for running GMRES on the reduced augmented system using the block IC factorization preconditioner. The time T_p includes the time for forming the reduced normal matrix C_s and computing its IC factorization, for

solving (2.18), and for forming and factorizing the Schur complement matrix (2.19). For problems `Trec14`, `scsd8-2r` and `12month1`, results are given for more than one value of the parameter ρ that controls which rows are classified as dense. As the density of C_s increases, a larger shift α is needed to prevent breakdown of the IC factorization and this has the effect of decreasing the quality of the preconditioner. However, for small ρ, for examples `12month1` and `Trec14`, m_d is large. Consequently, the factorization of the dense Schur complement $\tilde{S}$ is expensive and although the GMRES iteration count is much less than the LSMR count, for these two problems the Schur complement approach offers no significant benefit in terms of total time. For the other problems, exploiting the dense rows is advantageous. In particular, `PDE1` could not be solved via the normal equations but the reduced augmented system approach performs well. We observe that for the rank deficient `Maragal` problems, we found it was necessary to use a very small ρ to obtain a preconditioner that gave rapid convergence of GMRES (larger values of ρ led to unacceptably slow convergence). Finally, we remark that the size of the incomplete factors for the normal equations approach is approximately $lsize*n$ while for the Schur complement approach it is $lsize*n+m_d(m_d+1)/2$ (recall in our experiments the `HSL_MI35` memory parameter $lsize$ is set to 20).

TABLE 5.1

Results for Test Set 1 of running preconditioned LMSR on the normal equations using the IC factorization preconditioner `HSL_MI35`. *α denotes the global shift, Its is the number of LSMR iterations. T_p, T_s and T_{total} denote the times (in seconds) to compute the form the normal equations and compute the IC preconditioner, to run LSMR and the total time.*

Identifier	m	n	α	Its	T_p	T_s	T_{total}
`Trec14`	15904	3159	1.638×10^{1}	1050	4.52	3.19	7.71
`Maragal_6`	21251	10144	5.120×10^{-1}	1130	6.41	1.59	8.00
`Maragal_7`	46845	26525	2.048	410	19.9	1.47	21.4
`scsd8-2r`	60550	8650	3.277×10^{1}	140	0.46	0.19	0.64
`PDE1`	271792	270595	-	-	-	-	-
`12month1`	872622	12471	1.024	200	32.6	10.0	42.7

TABLE 5.2

Results for Test Set 1 of running GMRES on the reduced augmented system using the block IC factorization preconditioner. density(C_s) is the ratio of the number of entries in the reduced normal matrix C_s to n^2, α denotes the global shift, Its is the number of GMRES iterations. T_p, T_s and T_{total} denote the times (in seconds) to compute the preconditioner, to run GMRES and the total time.

Identifier	m	n	ρ	m_d	$density(C_s)$	α	Its	T_p	T_s	T_{total}
`Trec14`	15904	3159	0.050	4467	6.17×10^{-1}	6.400×10^{-2}	163	2.72	4.17	6.89
			0.100	2664	8.31×10^{-1}	2.560×10^{-1}	245	1.41	3.49	4.90
			0.200	1234	9.09×10^{-1}	1.024	352	1.71	2.74	4.45
`Maragal_6`	21251	10144	0.001	2923	6.22×10^{-4}	1.562×10^{-5}	62	1.46	1.86	3.33
`Maragal_7`	46845	26525	0.001	4668	2.15×10^{-4}	2.500×10^{-4}	15	8.95	1.74	10.7
`scsd8-2r`	60550	8650	0.050	50	1.44×10^{-3}	9.766×10^{-7}	2	0.03	0.00	0.03
			0.100	40	1.39×10^{-2}	3.227×10^{1}	68	0.32	0.13	0.44
`PDE1`	271792	270595	0.100	1	4.52×10^{-5}	8.000×10^{-3}	174	1.25	4.19	5.44
`12month1`	872622	12471	0.050	3641	5.66×10^{-1}	1.024	127	32.8	13.0	45.8
			0.100	284	6.56×10^{-1}	1.024	151	36.1	10.3	46.4

5.2. Results for Test Set 2. We now look at adding rows to the examples in Test Set 2. We first append a single row ($m_d = 1$) of increasing density and solving the normal equations using preconditioned LSMR with the `HSL_MI35` IC preconditioner. In Table 5.3, we report results for $\rho = 0.01, 0.1, 0.5$. Problem `CONT11_L` is omitted since the time to compute the IC factorization exceeds 600 seconds. In Table 5.4, results are given for $\rho = 1$; results are also given for running GMRES on the reduced augmented system using the block IC factorization preconditioner. We see that, if the normal equations are used, as ρ and hence the density of C increases, so too do the shift α needed to prevent breakdown, the time to compute the IC factorization, and the iterations for convergence. Indeed, for the large examples, the time exceeds our

limit of 600 seconds. By contrast, for preconditioned GMRES on the reduced augmented system, the shift and the times to compute the incomplete factorization and achieve convergence are essentially independent of ρ (and for this reason only results for $\rho = 1.0$ are included in Table 5.4). Furthermore, this approach uses a smaller shift than for the normal equations and produces a much higher quality preconditioner, leading to significantly faster times. With more than one added row, the density of C often increases further making the normal equation approach even less feasible. For the augmented approach, adding more than one row does not affect C_s or the time to compute the incomplete factorization but does result in the dense factorization of the Schur complement matrix becoming more expensive. For most of our test problems, the number of iterations decreases as the number of added rows increases (for example, `psse0` and `graphics` but for others (including `relat9`), the converse is true (see Table 5.5).

TABLE 5.3

Results for Test Set 2 with a single dense row of density ρ appended. Results are for preconditioned LMSR on the normal equations using the IC factorization preconditioner. α denotes the global shift, Its is the number of iterations. T_p, T_s and T_{total} denote the times (in seconds) to compute the IC preconditioner, to run the iterative solver and the total time. – indicates statistic unavailable.

Problem	*density*(C)	α	Its	T_p	T_s	T_{total}
$\rho = 0.01$						
IG5-15	1.52×10^{-1}	5.120×10^{-1}	280	0.18	0.29	0.47
psse0	5.33×10^{-4}	2.500×10^{-4}	1030	0.09	0.92	1.00
graphics	6.56×10^{-4}	2.500×10^{-4}	6350	0.03	6.30	6.33
WORLD	5.88×10^{-4}	1.638×10^{1}	1340	0.28	3.57	3.85
STAT96V3	4.57×10^{-4}	1.024	560	0.27	12.1	12.4
STORMG21K	4.00×10^{-4}	1.311×10^{2}	1620	51.9	101	153
GL7d20	3.23×10^{-4}	5.243×10^{2}	40	166	19.1	185
LargeRegFile	1.10×10^{-4}	1.311×10^{2}	70	127	7.68	134
relat9	6.09×10^{-4}	1.311×10^{2}	90	19.8	29.1	48.9
$\rho = 0.1$						
IG5-15	1.61×10^{-1}	3.277×10^{1}	810	0.23	0.83	1.07
psse0	1.06×10^{-2}	3.277×10^{1}	38200	0.22	40.2	40.4
graphics	1.06×10^{-2}	3.277×10^{1}	>100000	0.23	–	–
WORLD	1.05×10^{-2}	1.311×10^{2}	1840	0.88	4.91	5.79
STAT96V3	1.04×10^{-2}	1.311×10^{2}	880	1.19	19.86	21.0
STORMG21K	1.03×10^{-2}	1.049×10^{3}	1470	192	97.8	290
GL7d20	–	–	–	>600	–	>600
LargeRegFile	–	–	–	>600	–	>600
relat9	1.05×10^{-2}	5.243×10^{2}	90	63.7	28.6	92.3
$\rho = 0.5$						
IG5-15	3.64×10^{-1}	6.554×10^{1}	880	0.50	0.91	1.41
psse0	2.50×10^{-1}	2.621×10^{2}	34570	1.31	40.64	41.96
graphics	2.50×10^{-1}	2.621×10^{2}	>100000	1.38	–	–
WORLD	2.50×10^{-1}	5.243×10^{2}	2020	13.10	6.42	19.52
STAT96V3	2.50×10^{-1}	5.243×10^{2}	760	10.91	17.03	27.94
STORMG21K	–	–	–	>600	–	>600
GL7d20	–	–	–	>600	–	>600
LargeRegFile	–	–	–	>600	–	>600
relat9	–	–	–	>600	–	>600

6. Concluding remarks. In this paper, we have focused on using the Schur complement approach to solve large-scale linear least-squares problems in which the system matrix A contains a number of nearly dense rows. Our proposed approach involves using a regularization parameter and then applying a Cholesky solver to the shifted reduced normal equations. A small number of steps of the iterative solver GMRES applied to the reduced augmented system are then employed to recover the solution of the original (unshifted) problem. We have considered some hard-to-solve problems (including some rank deficient examples) from practical applications and shown that this approach offers savings (in terms of

TABLE 5.4

Results for Test Set 2 with a single dense row ($\rho = 1.0$) appended. Results are for preconditioned LMSR on the normal equations using the IC factorization preconditioner and for running GMRES on the reduced augmented system using the block IC factorization preconditioner. α denotes the global shift, Its is the number of iterations. T_p, T_s and T_{total} denote the times (in seconds) to compute the IC preconditioner, to run the iterative solver and the total time. – indicates statistic unavailable.

Problem	Normal equations with LSMR					Reduced augmented system with GMRES				
	α	Its	T_p	T_s	T_{total}	α	Its	T_p	T_s	T_{total}
IG5-15	6.5536×10^{1}	810	0.92	0.82	1.73	1.024	337	0.47	1.08	1.55
psse0	2.6214×10^{2}	33690	2.06	39.8	41.9	0.0	81	0.01	0.05	0.06
graphics	2.6214×10^{2}	>100000	3.56	>134	>138	9.766×10^{-7}	900	0.02	1.49	1.51
WORLD	5.2429×10^{2}	2040	39.9	7.16	47.1	1.280×10^{-1}	294	1.32	1.31	2.63
STAT96V3	5.2429×10^{2}	750	23.9	16.8	40.7	0.0	20	0.20	0.04	0.23
STORMG21K	–	–	>600	–	>600	2.621×10^{2}	1320	19.5	211	230
GL7d20	–	–	>600	–	>600	5.120×10^{-1}	32	187	39.0	226
CONT11_L	–	–	>600	–	>600	8.000×10^{-3}	142	9.23	22.8	32.0
LargeRegFile	–	–	>600	–	>600	0.0	11	1.61	0.53	2.14
relat9	–	–	>600	–	>600	9.766×10^{-7}	41	32.3	3.18	35.5

TABLE 5.5

Results for Test Set 2 of running GMRES on the reduced augmented system using the block IC factorization preconditioner when m_d =1, 50 and 100 rows are appended. Its is the number of iterations and T_{total} is the total time.

Problem	$m_d = 1$		$m_d = 50$		$m_d = 100$	
	Its	T_{total}	Its	T_{total}	Its	T_{total}
psse0	81	0.06	40	0.05	30	0.07
graphics	900	1.51	107	0.12	68	0.11
WORLD	294	2.63	158	2.13	116	2.21
IG5-15	337	1.55	151	0.95	129	0.91
STAT96V3	20	0.23	7	0.25	7	0.32
STORMG21K	1320	230	906	194	959	228
LargeRegFile	11	2.14	9	3.28	9	4.83
CONT11_L	142	32.0	12	13.0	13	15.9
GL7d20	32	226	22	222	22	231
relat9	41	35.5	160	33.3	279	65.1

time and the size of the factors) compared to using a general sparse symmetric indefinite solver. The approach can be used with an incomplete Cholesky factorization preconditioner. In this case, a larger shift is required to prevent breakdown of the factorization, and this increases with the density of the reduced normal matrix and number of iterations needed for convergence. We have also considered adding a number of dense rows to the matrix A. For some examples, if the dense rows are not explicitly exploited, we were unable to solve the least-squares problems using an IC preconditioner for the normal equations. However, the use of the reduced normal equations reduces the size of the shift needed, which gives a higher quality preconditioner that successfully solved the test problems when one or more dense rows were added.

Finally, we remark that although our main motivation for partitioning A is the presence of one or more dense rows, there are other possible reasons for employing a partitioning of the form (1.3). 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, [3, 4]. Nowadays, there exist important applications based on this motivation related to Kalman filtering or solving recursive least-squares problems, see the seminal paper [22] or for a comprehensive introduction [10, 35]. Furthermore, additional constraints for the least-squares problem represented by A_d and b_d naturally arise when solving rank-deficient least-squares problems (for instance, [5, 7, 28]). If extra rows are added, the sparse (incomplete) Cholesky factorization within the Schur complement approach can be reused and so the only work needed to solve the updated system (or, in the incomplete case, to apply a preconditioner for the enlarged system), is the solution of triangular systems.

REFERENCES

[1] E. D Andersen, J. Gondzio, C. Mészáros, and X. Xu. *Implementation of interior point methods for large scale linear programming*. HEC/Université de Genève, 1996.

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

[3] 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.

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

[5] 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.

[6] M. Benzi. Preconditioning techniques for large linear systems: a survey. *J. of Computational Physics*, 182(2):418–477, 2002.

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

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

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

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

[11] H. S. Dollar and J. A. Scott. A note on fast approximate minimum degree orderings for matrices with some dense rows. *Numerical Linear Algebra with Applications*, 17:43–55, 2010.

[12] I. S. Duff. MA57– a new code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, 30:118–154, 2004.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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*, 43(6), 2017. Article 36, 35 pages.

[19] J. D. Hogg, J. K. Reid, and J. A. Scott. Design of a multicore sparse Cholesky factorization using DAGs. *SIAM J. on Scientific Computing*, 32:3627–3649, 2010.

[20] J. D. Hogg and J. A. Scott. HSL_MA97: a bit-compatible multifrontal code for sparse symmetric systems. Technical Report RAL-TR-2011-024, Rutherford Appleton Laboratory, 2011.

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

[22] 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.

[23] G. Karypis and V. Kumar. METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices (version 3.0). Technical report, University of Minnesota, Department of Computer Science and Army HPC Research Center, October 1997.

[24] I. J. Lustig, R. E. Marsten, and D. F. Shanno. Computational experience with a primal-dual interior point method for linear programming. *Linear Algebra and its Applications*, 152:191–222, 1991.

[25] T. A. Manteuffel. An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation*, 34:473–497, 1980.

[26] A. Marxen. Primal barrier methods for linear programming. Technical Report SOL 89-6, Department of Operations Research, Stanford University, 1989.

[27] MUMPS. A MUltifrontal Massively Parallel sparse direct Solver, 2016. `http://graal.ens-lyon.fr/MUMPS/`.

[28] 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.

[29] C. C. Paige and M. A. Saunders. Algorithm 583; LSQR: Sparse linear equations and least-squares problems. *ACM Transactions on Mathematical Software*, 8(2):195–209, 1982.

[30] 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.

[31] J. K. Reid and J. A. Scott. An out-of-core sparse Cholesky solver. *ACM Transactions on Mathematical Software*, 36(2):9:1–9:33, 2009.

[32] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, second edition, 2003.

[33] Y. Saad and M. H. Schulz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 7:856–869, 1986.

[34] 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).

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

[36] J. A. Scott. On using Cholesky-based factorizations for solving rank-deficient sparse linear least-squares problems. Technical Report RAL-P-2016-005, Rutherford Appleton Laboratory, 2016.

[37] 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.

[38] 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.

[39] J. A. Scott and M. Tůma. Solving mixed sparse-dense linear least squares by preconditioned iterative methods. Technical Report RAL-TR-2017-P-001, Rutherford Appleton Laboratory, 2017.

[40] 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.

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

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

[43] WSMP. Watson sparse matrix package (WSMP), 2016. `http://researcher.watson.ibm.com/researcher/view_group.php?id=1426`.