Science & Technology Facilities Council

Preprint

RAL-P-2016-005

On using Cholesky-based factorizations for solving rank-deficient sparse linear least-squares problems

JA Scott

March 2016

Submitted for publication in SIAM Journal on 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

On using Cholesky-based factorizations for solving rank-deficient sparse linear least-squares problems

Jennifer Scott[1]

ABSTRACT

By examining the performance of modern parallel sparse direct solvers and exploiting our knowledge of the algorithms behind them, we perform numerical experiments to study how they can be used to efficiently solve rank-deficient sparse linear least-squares problems arising from practical applications. We consider both the regularized normal equations and the regularized augmented system. We employ the computed factors of the regularized systems as preconditioners with an iterative solver to obtain the solution of the original (unregularized) problem. Furthermore, we look at using limited-memory incomplete Cholesky-based factorizations and how these can offer the potential to solve very large problems.

Keywords: least-squares problems, normal equations, augmented system, sparse matrices, direct methods, iterative methods, Cholesky factorizations, preconditioning.

AMS(MOS) subject classifications: 65F05, 65F50

---

[1] Scientific Computing Department, STFC Rutherford Appleton Laboratory, Harwell Campus, Oxfordshire, OX11 0PX, UK.
jennifer.scott@stfc.ac.uk
Supported by EPSRC grant EP/M025179/1.

March 11, 2016

1 Introduction

In recent years, a number of methods have been proposed for preconditioning sparse linear least-squares problems; a brief overview with a comprehensive list of references is included in the introduction to the paper by Bru et al. [4]. The recent study of Gould and Scott [20, 21] reviewed many of these methods (specifically those for which software has been made available) and then tested and compared their performance using a range of examples coming from practical applications. One of the outcomes of that study was some insight into which least-squares problems in the widely-used sparse matrix collections CUTEst [19] and UFL [10] currently pose a real challenge for direct methods and/or iterative solvers. In particular, it found that most of the available software packages were not reliable or efficient for rank-deficient least squares problems (at least, not when run with the recommended settings for the input parameters that were employed in the study). In this paper, we look further at such problems and focus on the effectiveness of both sparse direct solvers and incomplete factorization preconditioners for solving them. A key theme is the use of regularization (see, for example, [15, 48, 49]). We propose computing a factorization (either complete or incomplete) of a regularized problem and then using this as a preconditioner for an iterative solver to recover the solution of the original (unregularized) problem.

The problem we are interested in is

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

where $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$ is large and sparse and $b \in \mathbb{R}^m$. Our focus is on the case where A is not of full column rank. Solving (1.1) is mathematically equivalent to solving the $n \times n$ *normal equations*

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

A well-known issue associated with solving (1.2) is that the condition number of the normal matrix C is the square of the condition number of A so that the normal equations can be highly ill-conditioned [3]. Indeed, if A does not have full column rank, C is positive semidefinite and computing a Cholesky factorization breaks down, that is, a zero (or, in practice, a negative) pivot is encountered at some stage of the factorization. In such cases, a black-box sparse Cholesky solver cannot be applied directly to (1.2). It is thus of interest to consider modifying C by adding a regularization term to allow the use of a Cholesky solver; this is explored in Section 3 and compared with using a sparse symmetric indefinite solver for factorizing C. In particular, we look at employing the factors of the regularized normal matrix as a preconditioner for the iterative method LSMR [14].

An alternative approach is to solve the mathematically equivalent $(m+n)\times(m+n)$ *augmented system*

$$Ky = c, \qquad K = \begin{bmatrix} \gamma I_m & A \\ A^T & 0 \end{bmatrix}, \qquad y = \begin{bmatrix} \gamma^{-1} r(x) \\ x \end{bmatrix}, \qquad c = \begin{bmatrix} b \\ 0 \end{bmatrix}, \tag{1.3}$$

where $\gamma > 0$, $r(x) = b - Ax$ is the residual vector, and I_m denotes the $m \times m$ identity matrix. The condition of K depends upon γ and the maximum and minimum singular values of A; it varies significantly with γ but with an appropriate choice (see [1, 3, 48]), K is much better conditioned than C. Important disadvantages of working with (1.3) are that K is indefinite and it is generally significantly larger than the normal matrix. A sparse direct indefinite solver computes a factorization of K of the form $(PL)D(PL)^T$, where P is a permutation matrix, L is unit lower triangular and D is block diagonal with 1×1 and 2×2 blocks on the diagonal corresponding to 1×1 and 2×2 pivots (see, for example, [11, 27]). Using an indefinite solver may result in a more expensive (and certainly more complex) factorization process than a Cholesky solver and, as reported in [20, 21], for large least squares problems, the amount of memory needed may be prohibitive. One reason for this is that the analyse phase of most widely-used sparse direct solvers chooses the pivot order on the basis of the sparsity pattern of the matrix and makes the assumption that the diagonal is non-zero. When (as in the augmented system) this is not the case, it can be necessary during the subsequent numerical factorization to make significant modifications to the pivot order, leading to much higher levels of fill in the factors (entries in the factor L that were zero in the system matrix

K) than was predicted during the analyse phase (see, for example, [28, 30]). Modifications to the pivot order are needed when a candidate pivot is found to be unstable. The conditions for deciding whether a pivot is stable typically depend on a threshold parameter (see Section 2); choosing this parameter is a compromise between retaining sparsity and ensuring stability. In Section 4, we examine the effects of relaxing the threshold parameter. We also look at regularizing the problem by modifying the $(2, 2)$ block of (1.3) before performing the factorization and then using the factors as a preconditioner for an iterative solver to restore the accuracy in the solution of the original system.

When memory is an issue for a direct solver, an alternative approach is to use an incomplete factorization preconditioner in conjunction with an iterative solver. Incomplete Cholesky (IC) factorizations have long been widely used as preconditioners for the numerical solution of large sparse, symmetric positive-definite linear systems of equations; for an introduction and overview see, for example, [2, 46, 51] and the many references therein. More recently, a number of authors have considered incomplete LDL^T factorizations of symmetric quasi-definite matrices [39], saddle-point systems [52] and general indefinite systems [22, 53]. The use of a limited memory IC factorization combined with LSMR to solve (1.1) is considered in Section 5, and in Section 6 we explore using incomplete LDL^T factorizations to solve the augmented system. Our conclusions are drawn in Section 7.

1.1 Test environment

We end this introduction by describing our test environment and test problems. The characteristics of the machine used to perform our tests are given in Table 1.1. All software is written in Fortran and all

Table 1.1: Test machine characteristics

CPU	Two Intel Xeon E5620 quadcore processors
Memory	24 GB
Compiler	gfortran version 4.7 with options -O3 -fopenmp
BLAS	open BLAS

reported timings are CPU times in seconds. In our experiments, the direct solvers `HSL_MA87` and `HSL_MA97` (see Section 2) are run in parallel, using 8 processors. We do not attempt to parallelize the sparse matrix-vector products used by the iterative solvers; moreover, the software to compute incomplete factorizations is serial. In each test, we impose a time limit of 600 seconds per problem. For the iterative methods, the number of iterations for each problem is limited to 100,000.

Our test problems are taken from the CUTEst linear programme set [19] and the UFL Sparse Matrix Collection [10]. 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); details of the resulting test problems are summarized in Table 1.2. Here the nullity is computed by running `HSL_MA97` on the augmented system (1.3) with the pivot threshold parameter set to 0.5 (see Section 2); the reported nullity is the difference between $m + n$ and the returned estimate of the matrix rank. Note that this estimate can be sensitive to the choice of ordering and scaling: `HSL_MA97` was used with no scaling and the nested dissection ordering computed by Metis [32]. In our experiments, if a right-hand side b is provided, it is used, otherwise, we take b to be the vector of 1's.

We employ the preconditioned LSMR algorithm of Fong and Saunders [14]. Like the more established LSQR algorithm [41, 42], it is based on Golub-Kahan bidiagonalization of A. However, while in exact arithmetic, LSQR is mathematically equivalent to applying the conjugate gradient method to (1.2), LSMR it is equivalent to applying MINRES [40], so that the quantities $\|A^T r_k\|_2$ and $\|r_k\|_2$ (where x_k and $r_k = b - Ax_k$ are the least squares solution and residual on the k-th step, respectively) are monotonically decreasing. Fong and Saunders report that LSMR may be a preferable solver because of this and because it may be able to terminate significantly earlier. Experiments in [20, 21] confirm this view and support

Table 1.2: Statistics for our test set. 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, density(A) is the largest ratio of number of nonzeros in a row of A to n over all rows, density(C) is the ratio of the number of entries in C to n^2, and $max|C_{ii}|$ and $min|C_{ii}|$ are the largest and smallest diagonal entries in C. A $-$ denotes insufficient memory to compute the statistic.

Problem	m	n	$nnz(A)$	nullity	density(A)	density(C)	$max\|C_{ii}\|$	$min\|C_{ii}\|$
CUTEst examples								
1. BAXTER	30733	27441	111576	2993	0.0017	0.0016	3.2×10^{5}	1.0×10^{-1}
2. DBIR1	45775	18804	1077025	103	0.0119	0.0119	8.5×10^{6}	1.0
3. DBIR2	45877	18906	1158159	101	0.0123	0.0069	3.0×10^{6}	1.0
4. LPL1	129959	39951	386218	44	0.0004	0.0003	5.4×10^{2}	1.0
5. NSCT2	37563	23003	697738	287	0.0273	0.0157	3.8×10^{6}	1.0
6. PDS-100	514577	156016	1096002	227	0.0000	0.0001	1.0	1.0
7. PDS-90	475448	142596	1014136	227	0.0000	0.0001	9.8	1.0
UFL Sparse Matrix Collection examples								
8. beaflw	500	492	53403	4	0.8130	0.8945	2.2×10^{5}	9.5×10^{-1}
9. 162bit	3606	3476	37118	15	0.0040	0.0195	2.5×10^{1}	7.2×10^{-3}
10. 176bit	7441	7150	82270	38	0.0022	0.0103	3.7×10^{1}	3.0×10^{-3}
11. 192bit	13691	13093	154303	81	0.0012	0.0057	5.4×10^{1}	2.5×10^{-4}
12. 208bit	24430	23191	299756	191	0.0008	0.0036	6.6×10^{-1}	1.2×10^{-4}
13. Maragal_6	21251	10144	537694	516	0.5857	0.7491	1.0×10^{1}	1.1×10^{-2}
14. Maragal_7	46845	26525	1200537	2046	0.3604	0.3099	1.3×10^{3}	1.4×10^{-2}
15. Maragal_8	60845	33093	1308415	7107	0.0503	0.0356	1.9	3.6×10^{-2}
16. mri1	114637	65536	589824	603	0.0037	0.0003	1.3	1.3
17. mri2	104597	63240	569160	-	0.0660	0.0078	1.3	1.3
18. tomographic1	59360	45908	647495	3436	0.0003	0.0009	4.4	4.5×10^{-4}

our choice of LSMR.

Following Gould and Scott [21], in our experiments with LSMR we use the stopping rule

$$ratio(r_k) < \delta \quad \text{with} \quad ratio(r_k) = \frac{\|A^T r_k\|_2/\|r_k\|_2}{\|A^T r_0\|_2/\|r_0\|_2}. \tag{1.4}$$

Unless indicated otherwise, we set the convergence tolerance δ to 10^{-6}. Note that (1.4) is independent of the choice of preconditioner.

When solving the augmented system (1.3) using an indefinite preconditioner, we use right preconditioned restarted GMRES [47]. Since GMRES is applied to the 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 the stopping condition (1.4) (which is based on A) is satisfied; it can, of course, be checked once GMRES has terminated. Instead, in our experiments involving (1.3), we use the scaled backward error

$$\frac{\|Ky_k - c\|_2}{\|c\|_2} < \epsilon, \tag{1.5}$$

where y_k is the computed solution of the augmented system on the k-th step. We set the tolerance ϵ to 10^{-7}. This stopping criterion is also applied in experiments involving MINRES.

2 Sparse direct solvers

Sparse direct solvers have long been used to solve both the normal equations and the augmented system. Here we briefly introduce such solvers, highlighting a number of features that are particularly relevant to understanding their performance when used for rank-deficient least squares problems.

Sparse direct methods are designed to solve symmetric linear systems $\mathcal{A}z = f$ ($\mathcal{A} = \{\mathcal{A}_{i,j}\}$) by performing a factorization

$$\mathcal{A} = LDL^T,$$

where L is a unit lower triangular matrix and D is a block diagonal matrix with non-singular 1×1 and 2×2 blocks. In practice, a more general factorization of the form

$$S\mathcal{A}S = (PL)D(PL)^T$$

is computed, where S is a diagonal scaling matrix and P is a permutation matrix that holds the pivot order. If $\mathcal{A}$ is positive definite, D is diagonal with positive diagonal entries and, in this case, L may be redefined to be the lower triangular matrix $L \Leftarrow LD^{1/2}$, giving the Cholesky factorization

$$S\mathcal{A}S = (PL)(PL)^T.$$

For efficiency in terms of both time and memory it is essential to choose P to exploit the sparsity of $\mathcal{A}$. The structure of the factor L is the union of the structure of the permuted matrix $P^T\mathcal{A}P$ and new entries known as *fill*. The amount of fill is highly dependent on the choice of P. Direct solvers choose P in an analyse phase that precedes the numerical factorization and generally works solely with the sparsity pattern of $\mathcal{A}$.

For positive-definite systems, the chosen pivot order can be used unaltered by the numerical factorization. However, for indefinite systems, it may be necessary to make modifications to maintain numerical stability. This is done by delaying the elimination of variables that could cause instability until later in the factorization when the associated pivot (that is, the 1×1 or 2×2 block used to eliminate one, respectively, two variables) can be safely used. The exact method used to select pivots during the numerical factorization varies from solver to solver, but essentially each seeks to avoid dividing a large off-diagonal entry by a small diagonal one. If the elimination of variable k is delayed, either an update from another elimination will increase the magnitude of the diagonal entry $\mathcal{A}_{k,k}$, or column k will become adjacent to column $k+1$ with the property that $\mathcal{A}_{k,k+1}$ is large and hence can be incorporated into a stable 2×2 pivot.

There are several modern sparse direct solvers available for solving positive-definite problems. Some are designed exclusively for such systems (including CHOLMOD [6] and `HSL_MA87` [25]) while others can also be used to solve indefinite systems (notably, `MA57` [11], `HSL_MA97` [27], MUMPS [38], WSMP [23], PARDISO [43] and `SPRAL_SSIDS` [24]). We employ the packages `HSL_MA87` and `HSL_MA97` from the HSL mathematical software library [31]; an overview of both packages together with a numerical comparison is provided by Hogg and Scott [29]. `HSL_MA87` is designed to run on multicore architectures. It splits each part of the computation into tasks of modest size but sufficiently large that good level-3 BLAS performance can be achieved. The dependencies between the tasks are implicitly represented by a directed acyclic graph (DAG). By contrast, `HSL_MA97` is a parallel multifrontal code that is able to solve both positive-definite and indefinite systems (although its performance on positive-definite systems is generally not competitive with that of `HSL_MA87`). In a multifrontal method, the factorization of $\mathcal{A}$ proceeds using a succession of assembly operations of small dense matrices, interleaved with partial factorizations of these matrices. The assembly operations can be recorded as a tree, known as an assembly tree. The assembly proceeds from the leaf nodes up the tree to the root node(s). Typically, most of the flops are performed at the root node and the final few levels of the tree. For the efficient and stable partial factorization of the dense submatrices, `HSL_MA97` uses separate computational kernels for the positive-definite and indefinite cases. For the latter, `HSL_MA97` employs the sufficient (but not necessary) conditions given by Duff et al. [12] for threshold partial pivoting to be stable. Let $\mathcal{A}^{(k)}$ denote the Schur complement after columns $1, \ldots, k-1$ of $\mathcal{A}$ have been eliminated, and let $u \in [0, 0.5]$ be the pivot threshold. The criteria for stability are:

- a 1×1 pivot on column k is stable if

$$\max_{i>k} |\mathcal{A}^{(k)}_{i,k}| < u^{-1}|\mathcal{A}^{(k)}_{k,k}| \tag{2.1}$$

- a 2×2 pivot on columns k and $k+1$ is stable if

$$\left| \begin{pmatrix} \mathcal{A}_{k,k}^{(k)} & \mathcal{A}_{k,k+1}^{(k)} \\ \mathcal{A}_{k+1,k}^{(k)} & \mathcal{A}_{k+1,k+1}^{(k)} \end{pmatrix}^{-1} \right| \begin{pmatrix} \max_{i>k+1} |\mathcal{A}_{i,k}^{(k)}| \\ \max_{i>k+1} |\mathcal{A}_{i,k+1}^{(k)}| \end{pmatrix} \leq u^{-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \tag{2.2}$$

  where the modulus of the matrix is interpreted element-wise. Additionally, it is required that the pivot can be stably inverted.

In the case where u is zero, this is interpreted as requiring that the pivot be nonsingular. Observe that these conditions imply that each entry in L is bounded in modulus by u^{-1}. The choice of u is a compromise between stability and sparsity: the larger u is, the more stable the factorize will be but the fill in L may increase. The default value within `HSL_MA97` is 0.01 and, unless stated otherwise, this value is used in our experiments. Note that including pivoting in an indefinite sparse direct solver is necessary for stability but it has the major disadvantage of hindering parallelism.

Both `HSL_MA87` and `HSL_MA97` employ a technique known as node amalgamation. This has become well established as a means of improving the factorization speed at the expense of the number of entries in the factor L and the operation counts during the factorization and subsequent solve phase. During the analyse phase, a child node in the tree is merged with its parent if either both parent and child have fewer than a prescribed number `nemin` of variables that are eliminated or merging parent and child generates no additional nonzeros in L. The value of the parameter `nemin` determines the level of node amalgamation, with a value in the range 8 to 32 typically recommended as providing a good balance between sparsity and efficiency in the factorize and solve phases (see [25, 44]). In our experiments, we set `nemin` equal to 32.

3 Direct solver as a preconditioner for LSMR

If A does not have full column rank, the normal matrix $C = A^T A$ is symmetric and positive semi definite and thus attempting to compute a Cholesky factorization will suffer breakdown. Breakdown occurs when a zero (or negative) pivot is encountered; if this happens, a Cholesky solver will terminate the computation with an error flag. A possibly remedy is to employ a global shift $\alpha > 0$ and to then compute a Cholesky factorization of the shifted and scaled matrix

$$C_\alpha = SA^T AS + \alpha I. \tag{3.1}$$

The shift α is also referred to as a *Tikhonov regularization* parameter. The choice of α should be related to the smallest eigenvalue of $SA^T AS$, but this information is not readily available. Clearly, α wants to be small; if the initial choice α is too small, it may be necessary to restart the factorization more than once, increasing α on each restart until breakdown is avoided. If the regularized scaled normal equations

$$C_\alpha x_\alpha = S(C + S^{-1}\alpha I S^{-1})Sx_\alpha = SA^T b$$

are solved, the computed value of the least squares objective $\|r_\alpha\|_2 = \|b - ASx_\alpha\|_2$ may differ from the optimum for the original problem. We can seek to recover the solution x to (1.1) by applying a refinement process. The standard approach for linear systems is to employ a small number of steps of iterative refinement. However, iterative refinement is not effective when applied to the normal equations if the normal matrix is ill conditioned [3]. We thus propose using the Cholesky factors $L_\alpha L_\alpha^T$ of C_α as a preconditioner for LSMR applied to the original (unshifted) problem.

In our experiments, we use l_2-norm scaling of the normal matrix, that is, $S_{ii}^2 = 1/\|Ce_i\|_2$ (where e_i is the i-th unit vector) so that each column of C is normalised by its 2-norm. In Figure 3.1, we plot the number of iterations required by preconditioned LSMR and the total CPU time for problem `Maragal_6` using values of the shift α in the range 10^{-14} to 10^{-3}; here L_α is computed using the positive-definite solver `HSL_MA87` with Metis [32] nested dissection ordering. In Table 3.1, we report results for our test set with $\alpha = 10^{-14}$. With the l_2-norm scaling, it was not necessary to use a larger α for any of our test

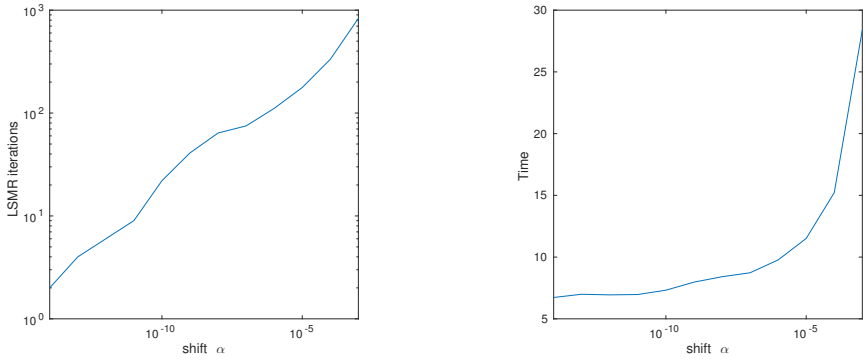

Figure 3.1: The effect of the shift α on the number of LSMR iterations (left) and the CPU time (right) for problem `Maragal_6`. `HSL_MA87` is used to compute a preconditioner for LSMR.

Table 3.1: Results for solving the least squares problem using the Cholesky solver `HSL_MA87` to compute a preconditioner for LSMR. The shift is $\alpha = 10^{-14}$. $nnz(L_\alpha)$ denotes the number of entries in the `HSL_MA87` factor, *time* is the total solution CPU time (in seconds), and *itn* is the number of LSMR iterations. The value of the least squares objective before and after applying LSMR is $\|r_\alpha\|_2$ and $\|r\|_2$, respectively. Results are also given for `HSL_MA97` run in indefinite mode (with no shift).

Problem	`HSL_MA87` (positive definite)					`HSL_MA97` (indefinite)		
	$nnz(L_\alpha)$	*time*	$\|r_\alpha\|_2$	$\|r\|_2$	*itn*	$nnz(L)$	*time*	$\|r\|_2$
`BAXTER`	6.83×10^6	0.32	6.683×10^1	5.929×10^1	24	6.95×10^6	0.44	5.929×10^1
`DBIR1`	4.27×10^6	0.42	1.667×10^2	1.667×10^2	1	4.61×10^6	0.87	1.667×10^2
`DBIR2`	4.94×10^6	0.46	1.665×10^2	1.665×10^2	1	4.85×10^6	0.96	1.665×10^2
`LPL1`	7.44×10^6	0.17	7.088×10^1	7.088×10^1	1	7.94×10^6	0.39	7.088×10^1
`NSCT2`	8.81×10^6	0.79	1.838×10^2	1.838×10^2	1	8.44×10^6	1.96	1.838×10^2
`PDS-100`	5.91×10^7	1.15	2.849×10^2	2.849×10^2	1	5.79×10^7	2.87	2.849×10^2
`PDS-90`	5.23×10^7	0.98	2.685×10^2	2.685×10^2	1	5.14×10^7	2.47	2.685×10^2
`beaflw`	1.17×10^5	0.05	4.346	4.162	7	1.15×10^5	0.05	4.200
`162bit`	2.82×10^6	0.08	1.177×10^1	1.177×10^1	1	2.85×10^6	0.25	1.177×10^1
`176bit`	1.02×10^7	0.33	1.842×10^1	1.842×10^1	1	1.03×10^7	1.37	1.842×10^1
`192bit`	2.85×10^7	1.29	2.485×10^1	2.485×10^1	1	2.98×10^7	5.68	2.485×10^1
`208bit`	8.60×10^7	6.62	3.850×10^1	3.850×10^1	1	8.57×10^7	25.5	3.850×10^1
`Maragal_6`	4.96×10^7	6.79	1.069×10^1	1.069×10^1	2	5.06×10^7	18.1	1.069×10^1
`Maragal_7`	1.43×10^8	22.4	1.369×10^1	1.369×10^1	2	1.39×10^8	44.3	1.369×10^1
`Maragal_8`	8.85×10^7	7.42	2.379×10^2	2.378×10^2	26	9.18×10^7	23.8	2.379×10^2
`mri1`	8.27×10^6	0.17	2.674×10^1	2.674×10^1	1	8.79×10^6	0.65	2.674×10^1
`mri2`	3.43×10^7	1.99	1.413×10^2	1.413×10^2	1	3.79×10^7	5.90	1.413×10^2
`tomographic1`	2.96×10^7	0.70	4.185×10^1	4.185×10^1	2	3.27×10^7	2.58	4.185×10^1

examples. We see that, in a many of cases, the requested accuracy is achieved with a single step of LSMR. We did experiment with running HSL_MA97 in positive-definite mode but, as reported in [29], it is generally slower than HSL_MA87.

An alternative to computing a Cholesky factorization and using the factor to precondition LSMR is to solve the normal equations using a sparse symmetric indefinite direct solver that allows the system matrix to be singular provided the equations to be solved are consistent. This has the advantage of not requiring the use of a shift α but because of pivoting, it may be more expensive than performing a Cholesky factorization. Results for our test problems run with HSL_MA97 in indefinite mode are included in Table 3.1 (using the l_2-norm scaling, Metis ordering and threshold parameter $u = 0.01$). Observe that the positive-definite solver applied to the scaled and shifted normal equations is faster than the indefinite solver; in many cases, the former is faster by more than a factor of 2.

4 Solving the augmented system with a direct solver

4.1 Using a direct solver for $Ky = c$

We now consider applying HSL_MA97 to the augmented system (1.3). The computed factorization is

$$SKS = (PL)D(PL)^T,$$

with S a diagonal scaling matrix, P a permutation matrix, and D is block diagonal (with 1×1 and 2×2 blocks). During the factorization of the augmented system, the zero $(2,2)$ block can lead to many modifications being made to the pivot order that was chosen by the analyse phase in order to preserve stability; this is reflected in the number of delayed pivots reported by HSL_MA97 ($ndelay$). It is well known that for some sparse indefinite problems, the choice of the scaling S can have a significant impact on reducing the number of delayed pivots and hence the fill in L and overall performance (see, for example, [26]). When developing HSL_MA97, Hogg and Scott studied pivoting strategies for tough sparse indefinite systems [30]. As no single method works best on all problems, HSL_MA97 offers a number of in-built scaling algorithms that are implemented using auxiliary HSL packages [31]. These options (1) generate a scaling using a weighted bipartite matching (MC64) [13]; or (2) generate an equilibration-based scaling (MC77) [45]; or (3) use a matching-based ordering and the scaling that is generated as a side-effect of this process (MC80); or (4) generate a scaling by minimising the absolute sum of log values in the scaled matrix (MC30). Note that, with the exception of MC80, these scalings do not exploit the block structure of K but treat K as a general indefinite sparse symmetric matrix. In addition to the HSL_MA97 options, we compute the l_2-norm scaling of K and the l_2-norm scaling of A. In the latter case, we set

$$S = \begin{bmatrix} I_m & \\ & \tilde{S} \end{bmatrix}, \tag{4.1}$$

where $\tilde{S}_{ii} = 1/\|Ae_i\|_2$. We illustrate the effects of scaling on a subset of our problems in Table 4.1. Here we use the default threshold parameter $u = 0.01$ and Metis ordering and set the parameter $\gamma = 1$ in (1.3). These examples were chosen because they were found to be sensitive to the scaling; for some of our other test examples, it had a much smaller effect. Closer examination shows that, if we compute the diagonal entries of C, the ratio of the largest to the smallest entry for these examples is large (see Table 1.2), indicating poor initial scaling. As expected, no single scaling gives the best results on all problems. The matching-based MC64 scaling can lead to sparse factors but it can be slow to compute. Based on our findings, we use l_2-norm scaling of A for our remaining experiments with HSL_MA97 applied to the augmented system and we set $\gamma = 1$ ([1]).

Results for solving the augmented system are given in Table 4.2. Compared to employing a direct solver to factorize the normal equations (Table 3.1), we see that, in general, factorizing the augmented system with the default threshold parameter $u = 0.01$ results in significantly more entries in the factors plus a slower computation time. In some optimization applications (see, for example, [17, 49]), it is common

Table 4.1: The effects of scaling on the performance of HSL_MA97 for solving the augmented system (1.3) ($u = 0.01$, Metis ordering, and $\gamma = 1$). $nnz(L)$ denotes the number of entries in the factor, *ndelay* is the number of delayed pivots, *time* is the total solution CPU time (in seconds). $l_2(K)$ and $l_2(A)$ denote l_4-norm scaling of K and A, respectively. For each problem, the sparsest factors and fastest times are in bold.

Problem	Scaling	$nnz(L)$	*ndelay*	*time*
BAXTER	none	8.36×10^6	2.74×10^4	0.50
	MC30	1.62×10^6	5.66×10^3	**0.24**
	MC64	$\mathbf{1.13}\times10^6$	1.08×10^3	0.26
	MC77	2.12×10^6	6.00×10^3	0.29
	MC80	1.49×10^6	1.37×10^2	0.27
	$l_2(K)$	5.72×10^6	2.00×10^4	0.38
	$l_2(A)$	1.38×10^7	3.41×10^4	0.73
DBIR1	none	9.79×10^7	1.73×10^5	47.6
	MC30	1.16×10^8	1.88×10^5	52.6
	MC64	9.02×10^6	5.02×10^3	1.44
	MC77	1.58×10^8	2.33×10^5	140
	MC80	2.03×10^7	3.93×10^4	3.62
	$l_2(K)$	1.05×10^8	1.80×10^5	62.1
	$l_2(A)$	$\mathbf{8.76}\times10^6$	1.12×10^3	**1.15**
DBIR2	none	1.23×10^8	1.23×10^5	26.6
	MC30	2.68×10^7	3.37×10^4	3.79
	MC64	$\mathbf{8.54}\times10^6$	4.00×10^2	1.41
	MC77	1.33×10^8	1.27×10^5	61.2
	MC80	1.96×10^7	1.12×10^4	3.17
	$l_2(K)$	1.00×10^8	1.04×10^5	18.5
	$l_2(A)$	8.85×10^6	9.62×10^2	**1.14**

Table 4.2: Results for solving the augmented system (1.3) using the direct solver HSL_MA97 with threshold parameter u set to 0.01 and 10^{-8}. $nnz(L)$ denotes the number of entries in the factor, *ndelay* is the number of delayed pivots, *time* is the total solution CPU time (in seconds), and the value of the least squares objective is $\|r\|_2$.

Problem	$u = 0.01$				$u = 10^{-8}$			
	$nnz(L)$	*ndelay*	*time*	$\|r\|_2$	$nnz(L)$	*ndelay*	*time*	$\|r\|_2$
BAXTER	1.38×10^7	3.41×10^4	0.73	5.929×10^1	1.49×10^6	4.75×10^3	0.22	5.929×10^1
DBIR1	8.76×10^6	1.12×10^3	1.15	1.667×10^2	8.75×10^6	1.08×10^3	1.13	1.667×10^2
DBIR2	8.85×10^6	9.62×10^2	1.13	1.665×10^2	8.46×10^6	1.11×10^2	1.12	1.665×10^2
LPL1	1.45×10^7	4.34×10^3	1.02	7.088×10^1	1.38×10^7	3.0	0.97	7.088×10^1
NSCT2	1.07×10^7	5.73×10^3	0.79	1.838×10^2	7.60×10^6	1.05×10^3	0.69	1.838×10^2
PDS-100	1.05×10^8	0.0	6.67	2.849×10^2	1.05×10^8	0.0	6.61	2.849×10^2
PDS-90	1.00×10^8	0.0	8.06	2.685×10^2	1.00×10^8	0.0	8.07	2.685×10^2
beaflw	2.65×10^5	2.95×10^2	0.05	4.162	2.51×10^5	6.70×10^1	0.03	4.162
162bit	4.25×10^6	3.42×10^2	0.33	1.177×10^1	4.23×10^6	1.59×10^2	0.31	1.177×10^1
176bit	1.52×10^7	2.33×10^3	2.07	1.842×10^1	1.50×10^7	1.07×10^3	1.98	1.842×10^1
192bit	4.51×10^7	9.00×10^3	7.85	2.485×10^1	4.44×10^7	3.80×10^3	7.59	2.485×10^1
208bit	1.30×10^8	2.45×10^4	44.1	3.850×10^1	1.28×10^8	1.09×10^4	42.1	3.850×10^1
Maragal_6	2.30×10^7	4.18×10^4	3.02	1.069×10^1	2.30×10^7	4.17×10^4	2.75	1.069×10^1
Maragal_7	7.03×10^7	3.48×10^4	9.32	1.369×10^1	7.03×10^7	3.47×10^4	8.41	1.369×10^1
Maragal_8	1.75×10^8	2.92×10^5	89.2	2.378×10^2	1.77×10^8	2.89×10^5	58.0	2.379×10^2
mri1	1.71×10^7	1.70×10^4	1.42	2.674×10^1	1.65×10^7	1.48×10^4	1.33	2.674×10^1
mri2	1.48×10^8	1.79×10^5	74.7	1.413×10^2	1.44×10^8	1.75×10^5	38.7	1.413×10^2
tomographic1	5.10×10^7	7.62×10^4	5.98	4.185×10^1	4.68×10^7	6.00×10^4	4.78	4.185×10^1

practice to try and make the factorization of indefinite systems more efficient in terms of time and memory by employing a relaxed threshold parameter u and to only increase u if the linear system is not solved with sufficient accuracy. Thus in Table 4.2 we also include results for $u = 10^{-8}$. With the exception of problems `BAXTER` and `NSCT2`, this gives only a modest reduction in the number of entries in the factors and in the number of delayed pivots but there can be a significant reduction in the time (including problems `Maragal_8` and `mri2`). Further examination reveals that pivots are still being rejected at the non-root nodes and passed up the assembly tree to the root node, where a dense factorization is performed. The delayed pivots result in the root node being much larger than was predicted by the analyse phase and, in this case, the factorization of the root node accounts for most of the operations and time. Using a small value of the threshold parameter significantly reduces the time for the root node factorization and it is this that leads to the overall reduction in the time.

4.2 Regularized augmented system

To attempt to reduce the number of entries in the factors of the augmented system resulting from delayed pivots, we consider the regularized system

$$K_\beta y_\beta = c, \qquad K_\beta = \begin{bmatrix} I_m & A \\ A^T & -I_n\beta \end{bmatrix}, \qquad y_\beta = \begin{bmatrix} r(x_\beta) \\ x_\beta \end{bmatrix}, \qquad c = \begin{bmatrix} b \\ 0 \end{bmatrix}, \tag{4.2}$$

where $r(x_\beta) = b - Ax_\beta$ and $\beta > 0$ (see, for example, [16, 48]). This a symmetric quasi-definite (SQD) system. Vanderbei [55] shows that, in exact arithmetic, SQD systems are strongly factorizable, i.e., a signed Cholesky factorization of the form LDL^T (with D diagonal having both positive and negative entries) exists for any symmetric permutation P. Thus P can be chosen to maintain sparsity. However, the signed Cholesky factorization may be unstable. A stability analysis is given by Gill et al. [17] (see also [15, 18]), which shows the importance of the effective condition number of K_β for the stability of the factorization.

We note that other regularizations of the augmented system have been proposed. In particular, Saunders [48, 49] and George and Saunders [15] use the SQD matrix

$$K_{\beta_1\beta_2} = \begin{bmatrix} I_m\beta_1 & A \\ A^T & -I_n\beta_2 \end{bmatrix},$$

with β_1, $\beta_2 > 0$ and in their experiments they set $\beta_1 = \beta_2 = 10^{-6}$. Saunders suggests that this may be favorable when A is ill-conditioned.

We apply our sparse symmetric indefinite solver `HSL_MA97` to the scaled regularized augmented matrix $SK_\beta S$ with the threshold parameter u set to 0.0 and S given by (4.1). With this choice, there may still be a small number of delayed pivots. This is because if the candidate pivot is zero or very small with a "large" off-diagonal entry, the pivot must be delayed (thus the permutation P from the analyse phase may change during the factorization). With $\beta > 0$, the computed value of the least squares objective $\|r_\beta\|_2 = \|b - A\tilde{S}x_\beta\|_2$ may differ from the optimum for the original problem and if the stopping criterion (1.4) is not satisfied, we propose using the factors as a preconditioner for an iterative method [15]. Here we use right preconditioned GMRES applied to the original augmented system (1.3). An alternative would be to use the symmetric solver MINRES [40]. This requires a positive definite preconditioner and so we could employ the method presented by Gill et al. [16] to modify D_β and L_β (this approach has been used recently by Greif et al. [22]).

Our results using GMRES are given in Figure 4.1 and Table 4.3. The figure looks at the effects of varying the regularization parameter β on the number of iterations and the solution time for problem `Maragal_6`. As β increases, so too do the iteration count and time. For the results in Table 4.3, we set $\beta = 10^{-8}$. In our experiments, this gives no delayed pivots. While the precise choice of β is not important, if β is "too small", some pivots may get delayed and the factorization is less stable, resulting in more GMRES iterations being needed for the requested accuracy than for a larger β. Comparing Tables 4.2

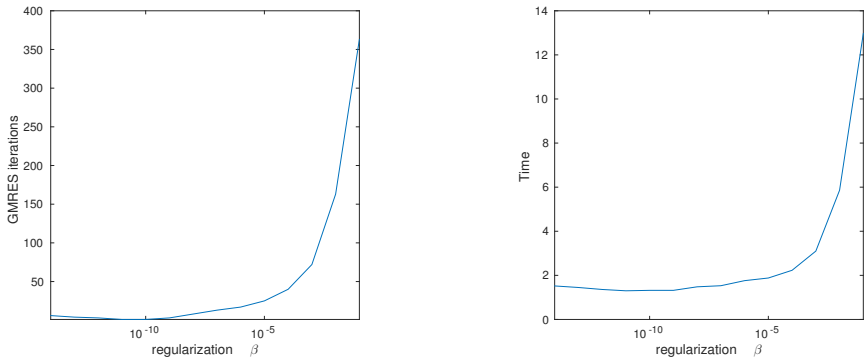

Figure 4.1: The effect of the regularization parameter β on the number of GMRES iterations (left) and the CPU time (right) for problem `Maragal_6` (threshold $u = 0.0$).

Table 4.3: Results for solving the regularized augmented (SQD) system with $\beta = 10^{-8}$ using the direct solver `HSL_MA97` with threshold parameter $u = 0.0$. $nnz(L_\beta)$ denotes the number of entries in the factor, $time_f$ and $time_t$ are the `HSL_MA97` and total solution CPU times (in seconds), the value of the least squares objective before and after applying preconditioned GMRES is $\|r_\beta\|_2$ and $\|r\|_2$, respectively, and itn is the number of GMRES iterations.

Problem	$nnz(L_\beta)$	$time_f$	$time_t$	$\|r_\beta\|_2$	$\|r\|_2$	itn
`BAXTER`	1.07×10^6	0.22	1.84	7.496×10^1	5.929×10^1	234
`DBIR1`	8.38×10^6	1.12	1.16	1.667×10^2	1.667×10^2	1
`DBIR2`	8.44×10^6	1.11	1.12	1.665×10^2	1.665×10^2	0
`LPL1`	1.38×10^7	0.97	0.97	7.088×10^1	7.088×10^1	0
`NSCT2`	7.38×10^6	0.67	0.67	1.838×10^2	1.838×10^2	0
`PDS-100`	1.05×10^8	6.64	6.66	2.849×10^2	2.849×10^2	0
`PDS-90`	1.00×10^8	8.18	8.20	2.685×10^2	2.685×10^2	0
`beaflw`	2.32×10^5	0.04	0.04	4.180	4.162	4
`162bit`	4.16×10^6	0.29	0.31	1.179×10^1	1.177×10^1	2
`176bit`	1.46×10^7	1.95	2.07	1.844×10^1	1.842×10^1	4
`192bit`	4.32×10^7	7.30	7.54	2.489×10^1	2.485×10^1	3
`208bit`	1.25×10^8	41.2	42.7	3.865×10^1	3.850×10^1	8
`Maragal_6`	1.50×10^7	1.28	1.50	1.069×10^1	1.069×10^1	8
`Maragal_7`	2.29×10^7	2.27	2.27	1.369×10^1	1.369×10^1	0
`Maragal_8`	2.31×10^7	3.74	4.62	2.388×10^2	2.386×10^2	17
`mri1`	1.34×10^7	1.00	1.01	2.674×10^1	2.674×10^1	0
`mri2`	8.72×10^6	1.06	1.08	1.413×10^2	1.413×10^2	0
`tomographic1`	3.16×10^7	2.70	3.08	4.206×10^1	4.194×10^1	5

and 4.3, we note that we obtain much sparser factors than previously and, as the number of iterations of GMRES is generally modest (indeed, in many cases we did not need to use GMRES to obtain the requested accuracy), we have significantly faster total solution times (note, in particular, problems `Maragal_8` and `mir2`). We observe that for problem `BAXTER`, we can reduce the number of GMRES iterations if we use a smaller β: with $\beta = 10^{-11}$, only 11 iterations are needed.

The number of entries in the factors and the total solution times using `HSL_MA97` to solve the regularized augmented system (4.2) and `HSL_MA87` applied to the shifted normal equations are compared in Figure 4.2. A point above the line $y = 1$ indicates using the normal equations is the better choice; the converse is true for a point below the line. We see that in many cases there is little to choose between the approaches in terms of the size of the factors but that for a small number of examples (including the `Maragal` problems) the augmented system factors are significantly sparser. However, the normal equation approach with the positive-definite solver is faster for most of the remaining problems.

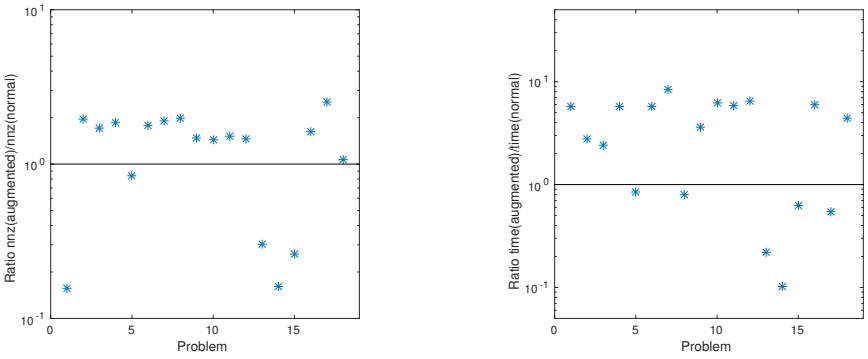

Figure 4.2: Ratios of the number of entries in the factor (left) and the CPU time (right) for the regularized augmented system solved using `HSL_MA97` in indefinite mode and the shifted normal equations solved using the positive-definite solver `HSL_MA87`.

5 Incomplete factorization of the normal matrix C

5.1 Limited memory approach

An incomplete Cholesky (IC) factorization takes the form LL^T in which some of the fill entries that would occur in a complete factorization are ignored. The preconditioned normal equations become

$$(AL^{-T})^T(AL^{-T})z = L^{-1}CL^{-T}z = L^{-1}A^Tb, \qquad z = L^Tx.$$

Performing preconditioning operations involves solving triangular systems with L and L^T. Over the years, a wealth of different IC variants have been proposed, including structure-based methods, those based on dropping entries below a prescribed threshold and those based on prescribing the maximum number of entries allowed in L (see, for instance, [2, 46, 51] and the references therein). Here we use the limited memory approach of Scott and Tůma [50, 51], which generalises the ICFS algorithm of Lin and Moré [35]. The basic scheme is based on a matrix factorization of the form

$$C = (L + R)(L + R)^T - E, \tag{5.1}$$

where L is the lower triangular matrix with positive diagonal entries that is used for preconditioning, R is a strictly lower triangular matrix with small entries that is used to stabilize the factorization process but is subsequently discarded, and $E = RR^T$. On the j-th step of the factorization, the first column of the

Schur complement is decomposed into a sum of two vectors

$$l_j + r_j,$$

such that $|l_j|^T|r_j| = 0$ (with the first entry in l_j nonzero), where l_j (respectively, r_j) contains the entries that are retained in (respectively, discarded from) the incomplete factorization. On the next step of a complete decomposition, the Schur complement of order $n-j$ would be updated by subtracting the outer product of the pivot row and column. That is, by subtracting

$$(l_j + r_j)(l_j + r_j)^T.$$

In the incomplete case, the positive semi definite term

$$E_j = r_j r_j^T$$

is not subtracted. Moreover, to further limit the memory required, drop tolerances are (optionally) used. If at some stage a zero or negative pivot is encountered, the factorization suffers breakdown and, as in Section 3, a shift is applied and the incomplete factorization of the shifted matrix (3.1) is computed.

A software package `HSL_MI35` that implements this limited memory IC algorithm for least squares problems has been developed. This code is a modification of `HSL_MI28` [50], which is designed for symmetric positive-definite systems. Modifications were needed to allow the user to specify the maximum number of entries allowed in each column of the incomplete factor L (in `HSL_MI28` the user specified the amount of fill allowed but as columns of the normal matrix C may be dense, or close to dense, this change was needed to keep L sparse). Furthermore, there is no need to form and store all of C explicitly; rather, the lower triangular part of its columns can be computed one at a time and then used to perform the corresponding step of the incomplete Cholesky algorithm before being discarded. `HSL_MI35` includes a number of scaling and ordering options so that an incomplete factorization of

$$\overline{C}_\alpha = P^T S C S P + \alpha I$$

is computed, where P is a permutation matrix chosen on the basis of sparsity, S is a diagonal scaling matrix and $\alpha \geq 0.0$. Based on extensive experimentation in [50], the defaults are the profile reduction ordering of Sloan [54] and the l_2-norm scaling, which needs only one column of C at a time. In the following, *lsize* and *rsize* denote the parameters that control the maximum number of entries in each column of L and R, respectively. In each test, the input value of the shift α is 0.001. In the event of breakdown, it is increased until the incomplete factorization is successful (see [50] for details). The recorded times include the time to restart the factorization following any breakdowns.

Figure 5.1 illustrates the potential benefits of employing intermediate memory R in the construction of L. Here we set *lsize* = 200 and vary *rsize* from 0 to 1000; the drop tolerances are set to 0.0. For each value of *rsize*, the number of entries in L is $nnz(L) = 6.63 \times 10^6$. We see that increasing the intermediate memory stabilizes the factorization, reducing the shift and giving a higher quality preconditioner that requires fewer LSMR iterations and less time. Note that because the number of restarts following a breakdown decreases as *rsize* increases, the time for computing the IC factorization does not necessarily increase with *rsize*.

In Table 5.1, we report results for `HSL_MI35` applied to our test set; default settings are used for the ordering, scaling and dropping parameters. We experimented with a range of values for the parameters *lsize* and *rsize* that control the memory and for each example chose values that perform well. As the memory increases, the preconditioner quality improves and the number of LSMR iterations decreases. However, the factorization times generally increase and, as $nnz(L)$ increases, each application of the preconditioner becomes more expensive. Thus the best values of *lsize* and *rsize* in terms of the total time are highly problem dependent; the results given in Table 5.1 illustrate this. We struggle to solve problem `BAXTER`: a larger number of iterations are required and the computed $\|r\|_2$ is significantly larger than reported elsewhere. If we compare the results for the remaining problems with those in Table 3.1 for the direct solver applied to the normal equations, we see that, with the exception of the `bit` problems, the serial IC times

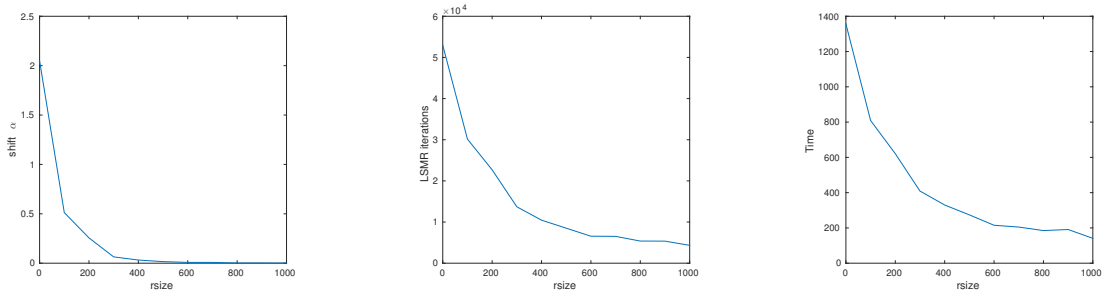

Figure 5.1: The effect of increasing the amount of intermediate memory used in the construction of the IC preconditioner on the size of the shift (left), the number of LSMR iterations (centre) and the total CPU time (right) in seconds for problem `Maragal_8`.

Table 5.1: Results for LSMR with the IC factorization preconditioner from `HSL_MI35` applied to $C = A^T A$. *lsize* and *rsize* control the memory used by `HSL_MI35`, $nnz(L)$ denotes the number of entries in the factor L, α is the shift, $time_f$ and $time_t$ are the factorization and total solution CPU times (in seconds), the value of the least squares objective is $\|r\|_2$ and the number of LSMR iterations is *itn*.

Problem	*lsize*	*rsize*	$nnz(L)$	α	$time_f$	$time_t$	$\|r\|_2$	*itn*
`BAXTER`	100	100	2.96×10^5	0.001	0.13	55.9	8.444×10^1	26930
`DBIR1`	100	0	4.17×10^5	0.002	0.41	1.50	1.668×10^2	270
`DBIR2`	100	0	4.27×10^5	0.004	0.56	2.54	1.666×10^2	470
`LPL1`	100	0	9.03×10^5	0.001	0.16	1.05	7.088×10^1	170
`NSCT2`	100	0	6.84×10^5	0.008	1.23	2.37	1.838×10^2	300
`PDS-100`	20	20	2.89×10^6	0.001	1.09	3.17	2.849×10^2	90
`PDS-90`	20	20	2.63×10^6	0.001	1.00	2.99	2.685×10^2	90
`beaflw`	100	200	2.55×10^4	0.001	0.04	1.61	4.531	9380
`162bit`	20	20	7.07×10^4	0.001	0.05	0.16	1.177×10^1	300
`176bit`	20	20	1.46×10^5	0.001	0.12	0.61	1.842×10^1	590
`192bit`	20	20	2.67×10^5	0.002	0.35	2.40	2.485×10^1	1290
`208bit`	20	20	4.71×10^5	0.002	0.69	7.37	3.850×10^1	2200
`Maragal_6`	20	20	2.12×10^5	0.256	6.67	8.01	1.069×10^1	680
`Maragal_7`	100	0	2.57×10^6	0.128	17.7	19.6	1.369×10^1	170
`Maragal_8`	200	1000	1.41×10^6	0.001	14.2	64.2	2.387×10^2	5310
`mri1`	100	100	1.10×10^6	0.001	0.55	1.43	2.674×10^1	110
`mri2`	20	20	7.83×10^5	2.048	3.18	18.7	1.413×10^2	2500
`tomographic1`	100	0	3.19×10^6	0.001	0.76	7.94	4.194×10^1	540

are not competitive with the times for the parallel direct solver. However, the IC factorization produces significantly sparser factors, giving it the potential to be used successfully for much larger problems than can be tackled by a direct solver. Observe that the shift α required by the IC factorization is significantly larger than is used by the direct solver. Consequently, the number of iterations needed for convergence can be large, significantly increasing the total solution time.

6 Preconditioning strategies for the augmented system

To compute an incomplete factorization preconditioner for the augmented system (1.3), one possibility is to extend the positive-definite limited memory approach that was outlined in Section 5. This was proposed by Scott and Tůma, who presented a limited memory incomplete signed Cholesky factorization [52]. They compute an incomplete factorization of the form LDL^T, where L is a lower triangular matrix with positive diagonal entries and D is a diagonal matrix with entries ± 1. In practice, an LDL^T factorization of

$$\overline{K} = P^T S K S P + \begin{bmatrix} \alpha_1 I & \\ & -\alpha_2 I \end{bmatrix}$$

is computed, where α_1 and α_2 are non-negative shifts chosen to prevent breakdown of the factorization. The preconditioner is taken to be $\overline{L}D\overline{L}^T$, with $\overline{L} = S^{-1}PL$. Scott and Tůma choose the permutation P not only on the basis of sparsity but also so that a variable in the $(2,2)$ block of K is not ordered ahead of any of its neighbours in the $(1,1)$ block. The idea here is to try and prevent a small (or zero) pivot candidate from being chosen; see [52] for details of this so-called constrained ordering.

An implementation is available as the HSL package `HSL_MI30`. As with the IC code `HSL_MI35`, intermediate memory (R) is optionally used in the construction of the factor L and is then discarded. The user controls the amount of fill allowed in each column of L (*lsize*) and the number of entries in each column of R (*rsize*). The code also includes a range of ordering and scaling options as well as optional dropping parameters (to control the discarding of small entries from L and R). Moreover, the user can supply initial values for the shifts.

`HSL_MI30` is related to the recent `LLDL` software that was developed independently by Orban [39]. The latter extends the ICFS code of Lin and Moré [35] to SQD matrices and thus can be applied to the regularized augmented system (4.2). The main differences between `HSL_MI30` and `LLDL` are:

1. `LLDL` uses a single shift (that is, $\alpha_1 = \alpha_2$).
2. `LLDL` does not employ intermediate memory (that is, $rsize = 0$).
3. The `HSL_MI30` factorization suffers breakdown and is restarted with an increased shift whenever a candidate pivot is not of the expected sign (or is zero). Thus if a pivot corresponding to an entry in the $(1,1)$ block is negative, the factorization is stopped and restarted with an increased value of α_1. Likewise, if a pivot corresponding to an entry in the $(2,2)$ block is positive, the factorization is stopped and restarted with an increased value of α_2. In `LLDL` there is breakdown only if a pivot is zero; in this case, α is increased and the factorization restarted.
4. `LLDL` does not use a constrained ordering but advises the user to preorder K using a sparsity-preserving ordering, such as AMD (there is no in-built option for ordering K).
5. `LLDL` does not include an option to drop small entries during the factorization.
6. `HSL_MI35` allows "spare" space from one column of L to be used for the next column (so that if, after dropping, column j of L has $p < lsize$ fill entries then column $j+1$ may have up to $2 * lsize - p$ fill entries).

For our experiments, we edited `HSL_MI30` so that it can optionally be used to implement the same algorithm as `LLDL`; this facilitates testing using the same ordering and scaling. We refer to this as the `LLDL` option

Table 6.1: Results for the signed IC factorization preconditioner `HSL_MI30` run with MINRES to solve the augmented system. *lsize* and *rsize* control the memory used by `HSL_MI30`, $nnz(L)$ denotes the number of entries in the factor L, α_2 is the shift for the (2,2) block (in all cases, $\alpha_1 = 0.0$), $time_f$ and $time_t$ are the factorization and total solution CPU times (in seconds), the value of the least squares objective is $\|r\|_2$, $ratio(r)$ is given by (1.4) and the number of MINRES iterations is itn.

Problem	*lsize*	*rsize*	$nnz(L)$	α_2	$time_f$	$time_t$	$\|r\|_2$	$ratio(r)$	itn
BAXTER	50	0	6.49×10^5	0.016	0.13	105	5.932×10^1	5.00×10^{-4}	39660
DBIR1	20	0	2.08×10^6	16.4	1.53	5.90	1.667×10^2	1.89×10^{-10}	515
DBIR2	50	0	2.41×10^6	0.512	1.35	77.5	1.665×10^2	2.46×10^{-12}	1729
LPL1	20	20	1.19×10^6	0.001	0.60	0.77	7.088×10^1	1.82×10^{-7}	21
PDS-100	20	0	6.08×10^6	0.000	1.74	2.51	2.849×10^2	3.16×10^{-7}	18
PDS-90	20	0	5.72×10^6	0.000	1.67	2.38	2.685×10^2	2.89×10^{-7}	18
beaflw	100	0	1.48×10^5	4.10	0.19	7.80	6.234×10^1	2.8	20730
162bit	20	20	1.45×10^5	0.000	0.10	0.22	1.179×10^1	2.25×10^{-6}	224
176bit	20	20	3.03×10^5	0.000	0.22	0.46	1.842×10^1	2.39×10^{-6}	205
192bit	50	50	8.86×10^5	0.001	1.54	3.48	2.487×10^1	2.49×10^{-6}	679
208bit	50	50	1.60×10^6	0.001	2.93	10.0	3.852×10^1	2.05×10^{-6}	1159
Maragal_6	20	20	6.75×10^5	1.020	2.61	5.71	1.069×10^1	7.73×10^{-7}	1043
Maragal_7	20	20	1.61×10^6	0.512	4.75	7.81	1.369×10^1	8.21×10^{-7}	353
Maragal_8	100	0	3.91×10^6	0.016	2.02	33.8	2.388×10^2	2.67×10^{-6}	1970
mri1	20	20	2.30×10^6	0.001	0.95	1.78	2.674×10^1	4.43×10^{-6}	45
mri2	100	0	1.43×10^7	0.512	61.92	92.4	1.413×10^2	9.16×10^{-7}	628
tomographic1	50	0	5.28×10^6	0.001	1.68	12.3	4.195×10^1	1.17×10^{-6}	507

Table 6.2: Results for the `LLDL` option run with MINRES to solve the augmented system. *lsize* controls the fill in each column of the factor L, $nnz(L)$ denotes the number of entries in the factor L, $time_f$ and $time_t$ are the factorization and total solution CPU times (in seconds), the value of the least squares objective is $\|r\|_2$, $ratio(r)$ is given by (1.4) and the number of MINRES iterations is itn. In all cases, $\alpha_1 = \alpha_2 = 1.0$.

Problem	*lsize*	$nnz(L)$	$time_f$	$time_t$	$\|r\|_2$	$ratio(r)$	itn
BAXTER	50	2.79×10^5	0.05	> 21.9	8.230×10^1	7.2608×10^{-4}	> 100000
DBIR1	20	1.27×10^6	0.23	> 600	1.686×10^2	3.1478×10^{-3}	> 96668
DBIR2	50	1.51×10^6	0.28	> 600	1.665×10^2	7.6498×10^{-4}	> 86415
LPL1	20	1.04×10^6	0.22	17.5	7.088×10^1	1.9460×10^{-5}	2472
PDS-100	20	3.65×10^6	1.21	23.6	2.849×10^2	6.4336×10^{-7}	695
PDS-90	20	3.37×10^6	1.16	20.6	2.685×10^2	6.5083×10^{-7}	667
beaflw	100	1.05×10^5	0.03	0.53	4.162	4.3794×10^{-8}	960
162bit	20	1.15×10^5	0.04	3.34	1.179×10^1	3.3382×10^{-6}	7089
176bit	20	2.38×10^5	0.09	13.2	1.845×10^1	2.3641×10^{-6}	13771
192bit	50	7.05×10^5	0.26	84.7	2.488×10^1	2.4705×10^{-6}	35921
208bit	50	1.28×10^6	0.52	187	3.863×10^1	2.1209×10^{-6}	37512
Maragal_6	20	7.44×10^5	0.28	13.9	1.069×10^1	8.3166×10^{-7}	4264
Maragal_7	20	1.68×10^6	0.62	13.8	1.369×10^1	1.1766×10^{-6}	1457
Maragal_8	100	2.06×10^6	0.65	586	2.388×10^2	6.5873×10^{-6}	56625
mri1	20	1.94×10^6	0.30	32.7	2.674×10^1	5.3777×10^{-6}	2660
mri2	100	2.40×10^6	0.35	76.1	1.413×10^2	1.3771×10^{-6}	6569
tomographic1	50	2.65×10^6	0.58	434	4.201×10^1	1.2579×10^{-6}	33330

(although note that if Orban's LLDL package is used, it will return similar but not identical results). Numerical results are given in Tables 6.1 and 6.2. Again, values of the memory parameters *lsize* and *rsize* are chosen after testing a range of values (we use the same *lsize* for HSL_MI30 and for the LLDL option). Recall $ratio(r)$ is given by (1.4); it enables us to see whether the LSMR stopping criteria is satisfied. We employ l_2-norm scaling and AMD ordering; based on the recommendations from [20, 21], the iterative solver is MINRES. For tests using the LLDL option, we set $\alpha_1 = \alpha_2 = 1.0$; this choice was made on the basis of experimentation. We set the regularization parameter β to 10^{-6} (recall (4.2)) but our experience is that, for our test set and chosen settings, using a nonzero value has little effect on the quality of the results. We observe that, while HSL_MI30 appears robust (it successfully solved all the problems in our test set except NSCT2, which is omitted since we were unable to choose parameters that led to successful convergence), with the LLDL option, not all the problems are solved within our limit of 600 seconds and 100,000 iterations. Moreover, for some examples, although MINRES reports convergence, the value of the least squares objective is too large and the LSMR stopping criteria is far from being satisfied.

Comparing the factor sizes for HSL_MI30 and the LLDL option, the latter generally has sparser factors. This is because of points (4) and (6) above. However, the LLDL preconditioner is generally of poorer quality and requires significantly more iterations, leading to a greater total time. If we compare the HSL_MI30 results with those for HSL_MI35 applied to the normal equations (Table 5.1), we see that HSL_MI30 is more expensive in terms of both the size of the incomplete factor and the total time (although the Maragal problems are notable exceptions with HSL_MI30 significantly faster). Moreover, for HSL_MI30 we found it was not always advantageous to use $rsize > 0$.

An alternative to using an incomplete Cholesky-based factorization is to employ a general incomplete indefinite factorization code. Chow and Saad [8] considered the class of incomplete LU preconditioners for solving indefinite problems and later Li and Saad [33] integrated pivoting procedures with scaling and reordering. Building on this, Greif, He, and Liu [22] recently developed an incomplete factorization package called SYM-ILDL for general sparse symmetric indefinite matrices. Here the system matrix may be any sparse indefinite matrix; no advantage is made of the specific block structure of (1.3). Independently, Scott and Tůma [53] report on the development of incomplete factorization algorithms for symmetric indefinite systems and propose a number of new ideas with the goal of improving the stability, robustness and efficiency of the resulting preconditioner. Experiments on our rank-deficient least squares test problems have found that the indefinite factorization is much less robust that the signed IC approach and so are not included here.

7 Concluding remarks

In this paper, we have used numerical experiments to study solving rank-deficient sparse linear least squares problems. These are hard problems to solve. Our approach is to use existing software to compute the factors of a regularized system and then to employ these factors as a preconditioner with an iterative method to recover the solution of the original problem. We have explored using state-of-the-art parallel sparse direct solvers to compute a complete factorization as well as recent approaches to compute limited-memory incomplete factorizations. Regularization allows a Cholesky-based direct solver to be used to factorize the scaled and shifted normal matrix C, avoiding the need for numerical pivoting that can adversely affect the performance of a sparse indefinite direct solver. However, this requires C to be available. If C cannot be formed or if it is unacceptably dense, the regularized augmented system with the threshold parameter $u = 0.0$ offers a feasible alternative approach (see also [49]).

As well as direct solvers, we have considered limited memory IC factorizations. Our results show that IC factorizations of the normal equations computed using the package HSL_MI35 provide robust preconditioners with significantly sparser factors than those from a complete factorization but they require a much larger number of LSMR iterations to achieve the requested accuracy. The use of intermediate memory in constructing these IC factors can significantly enhance the quality of the preconditioner without adding extra fill. For the signed IC factorizations of the augmented system, it is less clear that intermediate

memory is advantageous for solving rank-deficient least squares problems. In many of our test cases, the total solution time for the signed IC factorizations of the augmented system is greater than for the IC factorizations of the normal equations but, again, the former has the advantage of avoiding the construction of the normal matrix.

Currently, the codes that compute the IC factorizations and then perform the subsequent forward and back substitutions that are needed when using the factors as preconditioners are serial. As much of the total time is taken by the iterative solver, in the future parallel implementations of the application of the preconditioner needed. This is currently an area of active research [7].

Finally, although this paper has focused on tackling rank-deficient least squares problems using sparse direct LL^T and LDL^T solvers and their incomplete factorization counterparts, a number of other approaches are available. In particular, a QR factorization of A may be used, either a complete sparse QR factorization as offered by SuiteSparseQR [9] and qr_mumps [5], or an incomplete QR factorization such as the Multilevel Incomplete QR (MIQR) factorization of Li and Saad [34]. Moreover, the results reported in [20, 21] suggest that the BA-GMRES approach of Hayami et al. [36, 37] may offer a feasible alternative.

Acknowledgements

We are very grateful to our colleague Nick Gould for many useful discussions and to Michael Saunders for his carefully reading of and constructive comments on a draft of this paper. We would also like to thank Dominique Orban for discussions related to using his `LLDL` software for problems with A rank deficient.

References

[1] M. Arioli, I. S. Duff, and P. P. M. Rijk, *On the augmented system approach to sparse least-squares problem*, Numerische Mathematik, 55 (1989), pp. 667–684.

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

[3] Å. Björck, *Numerical methods for Least Squares Problems*, SIAM, Philadelphia, 1996.

[4] R. Bru, J. Marín, J. Mas, and M. Tůma, *Preconditioned iterative methods for solving linear least squares problems*, SIAM J. Sci. Comput., 36 (2014), pp. A2002–A2022.

[5] A. Buttari, *Fine-grained multithreading for the multifrontal qr factorization of sparse matrices*, SIAM J. on Scientific Computing, 35 (2013), pp. C323–C345.

[6] Y. Chen, T. A. Davis, W. H. Hager, and S. Rajamanickam, *Algorithm 887: CHOLMOD, supernodal sparse cholesky factorization and update/downdate*, ACM Transactions on Mathematical Software, 35 (2008), pp. 22:1–22:14.

[7] E. Chow and A. Patel, *Fine-grained parallel incomplete LU factorization*, SIAM J. on Scientific Computing, 37 (2015), pp. C169–C193.

[8] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, J. of Computational and Applied Mathematics, 86 (1997), pp. 387–414.

[9] T. A. Davis, *Algorithm 915: SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse QR factorization*, ACM Transactions on Mathematical Software, 38 (2011), pp. 8:1–8:22.

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

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

[12] I. S. Duff, N. I. M. Gould, J. K. Reid, J. A. Scott, and K. Turner, *Factorization of sparse symmetric indefinite matrices*, IMA Journal of Numerical Analysis, 11 (1991), pp. 181–204.

[13] I. S. Duff and J. Koster, *The design and use of algorithms for permuting large entries to the diagonal of sparse matrices*, SIAM J. on Matrix Analysis and Applications, 20 (1999), pp. 889–901.

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

[15] A. George and M. A. Saunders, *Solution of sparse linear equations using Cholesky factors of augmented systems*, Research Report SOL 99-1, Dept of EESOR, Stanford University, 1999.

[16] P. E. Gill, W. Murray, D. B. Ponceleón, and M. A. Saunders, *Preconditioners for indefinite systems arising in optimization*, SIAM J. on Matrix Analysis and Applications, 13 (1992), pp. 292–311.

[17] P. E. Gill, M. A. Saunders, and J. R. Shinnerl, *On the stability of Cholesky factorization for symmetric quasidefinite systems*, SIAM J. on Matrix Analysis and Applications, 17 (1996), pp. 35–46.

[18] G. H. Golub and C. F. Van Loan, *Unsymmetric positive definite linear systems*, Linear Algebra and its Applications, 28 (1979), pp. 85–97.

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

[20] N. I. M. Gould and J. A. Scott, *A note on performance profiles for benchmarking software*, Technical Report RAL-P-2015-004, Rutherford Appleton Laboratory, 2015.

[21] ———, *The state-of-the-art of preconditioners for sparse linear least squares problems*, Technical Report RAL-P-2015-010, Rutherford Appleton Laboratory, 2015.

[22] C. Greif, S. He, and P. Liu, *SYM-ILDL: incomplete LDLT factorization of symmetric indefinite and skew symmetric matrices*, technical report, Department of Computer Science, The University of British Columbia, 2015. Software available from `http://www.cs.ubc.ca/~inutard/html/`.

[23] A. Gupta, *WSMP Watson sparse matrix package (Part II: direct solution of general sparse systems*, Technical Report RC 21888 (98472), IBM T.J. Watson Research Center, 2000.

[24] J. D. Hogg, E. Ovtchinnikov, and J. A. Scott, *A sparse symmetric indefinite direct solver for GPU architectures*, Preprint RAL-P-2014-006, Rutherford Appleton Laboratory, 2014.

[25] 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 (2010), pp. 3627–3649.

[26] J. D. Hogg and J. A. Scott, *The effects of scalings on the performance of a sparse symmetric indefinite solver*, Technical Report RAL-TR-2008-007, Rutherford Appleton Laboratory, 2008.

[27] 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.

[28] ———, *A study of pivoting strategies for tough sparse indefinite systems*, Technical Report RAL-TR-2012-009, Rutherford Appleton Laboratory, 2012.

[29] ———, *New parallel sparse direct solvers for multicore architectures*, Algorithms, 6 (2013), pp. 702–725. Special issue: Algorithms for Multi Core Parallel Computation.

[30] ———, *Pivoting strategies for tough sparse indefinite systems*, ACM Transactions on Mathematical Software, 40 (2013). Article 4, 19 pages.

[31] *HSL. A collection of Fortran codes for large-scale scientific computation*, 2013. `http://www.hsl.rl.ac.uk`.

[32] 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)*, tech. rep., University of Minnesota, Department of Computer Science and Army HPC Research Center, October 1997.

[33] N. Li and Y. Saad, *Crout versions of ILU factorization with pivoting for sparse symmetric matrices*, Electronic Transactions on Numerical Analysis, 20 (2005), pp. 75–85.

[34] ———, *MIQR: A multilevel incomplete QR preconditioner for large sparse least-squares problems*, SIAM J. on Matrix Analysis and Applications, 28 (2006).

[35] C.-J. Lin and J. J. Moré, *Incomplete Cholesky factorizations with limited memory*, SIAM J. on Scientific Computing, 21 (1999), pp. 24–45.

[36] K. Morikuni and K. Hayami, *Inner-iteration Krylov subspace methods for least squares problems*, SIAM J. on Matrix Analysis and Applications, 34 (2013), pp. 1–22.

[37] ———, *Convergence of inner-iteration GMRES methods for rank deficient least squares problems*, SIAM J. on Matrix Analysis and Applications, 36 (2015), pp. 225–250.

[38] *MUMPS 5.0.0: a multifrontal massively parallel sparse direct solver*, 2015. `http://mumps-solver.org`.

[39] D. Orban, *Limited-memory LDLT factorization of symmetric quasi-definite matrices*, Numerical Algorithms, 70 (2015), pp. 9–41.

[40] C. C. Paige and M. A. Saunders, *Solution of sparse indefinite systems of linear equations*, SIAM J. on Numerical Analysis, 12 (1975), pp. 617–629.

[41] ———, *Algorithm 583; LSQR: Sparse linear equations and least-squares problems*, ACM Transactions on Mathematical Software, 8 (1982), pp. 195–209.

[42] ———, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Transactions on Mathematical Software, 8 (1982), pp. 43–71.

[43] *PARDISO 5.0.0 solver project*, 2014. `http://www.pardiso-project.org`.

[44] J. K. Reid and J. A. Scott, *An out-of-core sparse Cholesky solver*, ACM Transactions on Mathematical Software, 36 (2009). Article 9, 33 pages.

[45] D. Ruiz, *A scaling algorithm to equilibrate both rows and columns norms in matrices*, Technical Report RAL-TR-2001-034, Rutherford Appleton Laboratory, Chilton, Oxfordshire, England, 2001.

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

[47] 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 (1986), pp. 856–869.

[48] M. A. Saunders, *Solution of sparse rectangular systems using LSQR and CRAIG*, BIT Numerical Mathematics, 35 (1995), pp. 588–604.

[49] ———, *Cholesky-based methods for sparse least sqaures; the benefits of regularization*, in Linear and Nonlinear Conjugate Gardaient-Related Methods, L. Adams and J. L. Nazareth, ed., SIAM, Philadelphia, PA, 1996, pp. 92–100.

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

[51] ———, *On positive semidefinite modification schemes for incomplete Cholesky factorization*, SIAM J. on Scientific Computing, 36 (2014), pp. A609–A633.

[52] ———, *On signed incomplete Cholesky factorization preconditioners for saddle-point systems*, SIAM J. on Scientific Computing, 36 (2014), pp. A2984–A3010.

[53] ———, *Solving symmetric indefinite systems using memory efficient incomplete factorization preconditioners*, Technical Report RAL-P-2015-002, Rutherford Appleton Laboratory, 2015.

[54] S. W. Sloan, *An algorithm for profile and wavefront reduction of sparse matrices*, International J. of Numerical Methods in Engineering, 23 (1986), pp. 239–251.

[55] R. J. Vanderbei, *Symmetric quasidefinite matrices*, SIAM J. on Optimization, 5 (1995), pp. 100–113.