Science & Technology Facilities Council

Preprint

RAL-P-2015-010

The state-of-the-art of preconditioners for sparse linear least-squares problems

NIM Gould, J Scott,

November 2015

Submitted for publication in ACM Transactions on Mathematical Software

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

The state-of-the-art of preconditioners for sparse linear least-squares problems

Nicholas Gould[1] and Jennifer Scott[1]

ABSTRACT

In recent years a variety of preconditioners have been proposed for use in solving large sparse linear least-squares problems. These include simple diagonal preconditioning, preconditioners based on a number of different approaches to incomplete factorization and stationary inner iterations used with Krylov subspace methods. In this study, we briefly review available preconditioners for which software has been made available and then present a numerical evaluation of them using performance profiles and a large set of problems arising from practical applications. Comparisons are made with state-of-the-art sparse direct methods.

Keywords: least-squares problems, normal equations, augmented system, sparse matrices, iterative solvers, preconditioning.

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

---

[1] Scientific Computing Department, STFC Rutherford Appleton Laboratory, Harwell Oxford, Oxfordshire, OX11 0QX, UK.
nick.gould@stfc.ac.uk and jennifer.scott@stfc.ac.uk
Project supported by EPSRC grants EP/I013067/1 and EP/M025179/1.

November 23, 2015

1 Introduction

The method of least-squares is a commonly used approach to find an approximate solution of overdetermined or inexactly specified systems of equations. Since its development in the 18^{th} century [21], the solution of least-squares problems has been, and continues to be, a fundamental method in scientific data fitting. least-squares solvers are used across a wide range of disciplines, for everything from simple curve fitting, through the estimation of satellite image sensor characteristics, data assimilation for weather forecasting and for climate modelling, to powering internet mapping services, exploration seismology, NMR spectroscopy, piezoelectric crystal identification (used in ultrasound for medical imaging), aerospace systems, and neural networks.

In this study, we are interested in the important special case of the linear least-squares problem,

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

where $A \in \mathbb{R}^{m \times n}$ with $m \geq n$ is large and sparse and $b \in \mathbb{R}^m$. Solving (1.1) is mathematically equivalent to solving the $n \times n$ *normal equations*

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

and this, in turn, is equivalent to solving the $(m+n) \times (m+n)$ *augmented system*

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

where $r(x) = b - Ax$ is the residual vector and I_m is the $m \times m$ identity matrix. Increasingly, the sizes of the problems that scientists and engineers wish to solve are getting larger (problems in many millions of variables are becoming typical); they are also often ill-conditioned. In other applications, it is necessary to solve many thousands of problems of modest size and so efficiency in this case is essential. The normal equations are attractive in that, if A is of full column rank, they involve a symmetric positive definite linear system. However, a well-known drawback is that the condition number of C is the square of the condition number of A so that the normal equations are often highly ill-conditioned [8]. Furthermore, the density of C can be much greater than that of A (if A has a single dense row, C will be dense). The main disadvantages of working with the augmented system are that K is symmetric indefinite and is much larger than C (particularly if $m \gg n$).

Two main classes of method may be used to try and solve these linear systems: direct methods and iterative methods. A direct method proceeds by computing an explicit factorization, either a sparse LL^T Cholesky factorization of the normal equations (1.2) (assuming A is of full column rank so that C is positive definite) or a sparse LDL^T factorization of the augmented system (1.3). Alternatively, a QR factorization of A may be used, that is, a "thin" QR factorization of the form

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix},$$

where Q is an $m \times m$ orthogonal matrix and R is an $n \times n$ upper triangular matrix. Whilst direct solvers are generally highly reliable, iterative methods may be preferred because they often require significantly less storage and in some applications it may not be necessary to solve the system with the high accuracy offered by a direct solver. However, the successful application of an iterative method often requires a suitable preconditioner to achieve acceptable (and ideally, fast) convergence rates. Currently, there is much less knowledge of preconditioners for least-squares problems than there is for sparse symmetric linear systems and, as remarked in [9], "the problem of robust and efficient iterative solution of least-squares problems is much harder than the iterative solution of systems of linear equations". This is, at least in part, because A does not have the properties of differential problems that can make standard preconditioners effective.

In the past decade or so, a number of different techniques for preconditioning Krylov subspace methods for least-squares problems have been developed. A brief overview with a comprehensive list of references

is included in the introduction to the recent paper by Bru et al [9]. However, in the literature the reported experiments on the performance of different preconditioners are often limited to a small set of problems, generally arising from a specific application. Moreover, they may use prototype codes that are not available for others to test and they may only be run using MATLAB. Our aim is to perform a wider study in which we use a large set of test problems to evaluate the performance of a range of preconditioners for which software has been made available. The intention is to gain a clearer understanding of when particular preconditioners perform well (or, indeed, perform poorly) and we will use this to influence our future work on linear least-squares. Our attention is limited to preconditioners for which software in Fortran or C is available; it is beyond the scope of this work to provide efficient and robust implementations for all the approaches that have appeared in the literature (although even then, as we discuss in Section 8, we have found it necessary in some cases to modify and possibly re-engineer some of the existing software to make it suitable for use in this study).

The rest of the paper is organised as follows. In Section 2, we describe our test environment, including the set of problems used in this study. Direct solvers for solving the normal equations and/or the augmented system are briefly recalled in Section 3. One of these (`HSL_MA97`) is used for comparison with the performance of the preconditioned iterative methods. In Section 4, we report on experiments with two methods, LSQR and LSMR, that are mathematically equivalent to applying conjugate gradients and MINRES, respectively, to the normal equations but have favourable numerical properties. On the basis of our findings, LSMR is used in the rest of our experiments. Preconditioning strategies are briefly described in Sections 5 to 7. The software used in our experiments is discussed in Section 8. We present numerical results in Section 9 and finally, in Section 10, concluding remarks are made.

2 Test environment

The characteristics of the machine used to perform our tests are given in Table 2.1. In our experiments,

Table 2.1: Test machine characteristics

Processor	8× Intel i7-4790 (3.6 GHz)
Memory	15.6 Gbytes
Compiler	gfortran version 4.7 with option -O
BLAS	open BLAS

the direct solvers (see Section 3) are run in parallel, using 4 processors. We do not attempt to parallelize the sparse matrix-vector products used by the iterative solvers and all tests with these solvers are run in serial (although where BLAS are used, these take advantage of parallelism).

For each solver and each problem, an elapsed time limit of 600 seconds is imposed; if this limit is exceeded, the solver is flagged as having failed on that problem. Failures resulting from insufficient memory are also flagged and, in the case of the iterative solvers, the number of iterations per problem is limited to 100,000. We observe that, although the tests were performed on a lightly loaded machine, the timings can vary if the experiments are repeated. In our experience, this variation is small (typically less than 5%), although for large problems for which memory becomes an issue, the variation can be more significant. Unfortunately, given the large scale nature of this study and time taken to perform the experiments, it was not possible to produce average timings. However, variations in time that may arise from reruns will have little effect on the conclusions we can draw from the performance profiles that we use as our main tool for assessing performance (see Section 2.3).

2.1 Test problems

The problems used in our study are all taken from the CUTEst linear programme set [24] and the UFL Collection [16]. To determine the test set that we shall use for the majority of our experiments, we selected all the rectangular matrices A and removed "duplicates" (that is, similar problems belonging to same group), leaving a single representative. This gave us a set of 921 problems. In all our tests, we check A for duplicate entries (they are summed), out-of-range entries (they are removed) and explicit zeros (they are removed). In addition, A is checked for null rows and columns. Any such rows or columns are removed and if, after removal $n < m$, the matrix is transposed. The computation then continues with the resulting cleaned matrix.

To ensure we only include non-trivial problems, for each cleaned matrix we solved the normal equations (1.2) using LSMR (see Section 4, with the local reorthogonalization parameter set to 10) without preconditioning and retained those problems for which convergence (using the stopping criteria discussed in Section 2.2) was not achieved within 100,000 iterations or required more than 10 seconds. Using the provided right-hand side vector b if available or taking b to be the vector of 1's if not (so that the problems are not necessarily consistent but at the same time this choice makes it straightforward to regenerate the same b for running tests with a range of solvers) resulted in a test set $\mathcal{T}$ of 83 problems. This set is listed along with some of the characteristics of each problem (including the number of entries, the density of the row with the most entries, an estimate of the deficiency in the rank) in Table A.1 in the Appendix (see [28] for details of the full set).

Having chosen our test set, for each solver, 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.

2.2 Stopping criteria

Recall that the linear LS problem we seek to solve is

$$\min \phi(x), \quad \phi(x) = \|r(x)\|_2,$$

where $r(x) = Ax - b$ is the residual. If the minimum residual is zero, $\phi(x)$ is non differentiable at the solution and so the first check we make at iteration k is on the kth residual $\|r_k\|_2$, where $r_k = b - Ax_k$ with x_k the computed solution on the kth iteration. If the minimum residual is non zero then

$$\nabla\phi(x) = \frac{A^T r(x)}{\|r(x)\|_2},$$

and we want to terminate once $\nabla\phi(x)$ is small. Thus, in our tests with iterative solvers we use the following stopping rules:

C1: Stop if $\|r_k\|_2 < \delta_1$

C2: Stop if

$$\frac{\|A^T r_k\|_2}{\|r_k\|_2} < \frac{\|A^T r_0\|_2}{\|r_0\|_2} * \delta_2,$$

where A is the "cleaned" matrix and δ_1 and δ_2 are convergence tolerances that we set to 10^{-8} and 10^{-6}, respectively. In all our experiments, we take the initial solution guess to be $x_0 = 0$ and in this case C2 reduces to

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

Note that these stopping criteria are **independent of the preconditioner** and thus they enable us to compare the performances of different preconditioners. In the case of no preconditioning, these stopping criteria are closely related to those used by Fong and Saunders [22] in their implementation of LSMR (see `http://web.stanford.edu/group/SOL/download.html`). However, if a preconditioner is

used, the Fong and Saunders implementation bases the stopping criteria on $\|(AM^{-1})^T r\|_2$, where M is the (right) preconditioner. This means that a different test is applied for different preconditioners and thus is not appropriate for comparing the performances of different preconditioners. Using C1 and C2 involves additional work; in our tests, we have chosen to exclude the cost of computing the residuals for testing C1 and C2 from the reported runtimes (and from the 600s time limit per problem) and we use a modified reverse communication version of LSMR that enables us to use C1 and C2 in place of the Fong and Saunders stopping criteria.

2.3 Performance profiles

To assess the performance of different solvers on our test set $\mathcal{T}$, we report the raw data but we also employ performance profiles [17], which in recent years have become a popular and widely used tool for providing objective information when benchmarking software. The performance ratio for an algorithm on a particular problem is the performance measure for that algorithm divided by the smallest performance measure for the same problem over all the algorithms being tested (here we are assuming that the performance measure is one for which smaller is better, for example, the iteration count or time taken). The performance profile is the set of functions $\{p_i(f) : f \in [1, \infty)\}$, where $p_i(f)$ is the proportion of problems where the performance ratio of the ith algorithm is at most f. Thus $p_i(f)$ is a monotonically increasing function taking values in the interval $[0, 1]$. In particular, $p_i(1)$ gives the fraction of the examples for which algorithm i is the winner (that is, the best according to the performance measure), while if we assume failure to solve a problem (for example, through the maximum iteration count or time limit being exceeded) is signaled by a performance measure of infinity, $p_i^* := \lim_{f\to\infty} p_i(f)$ gives the fraction for which algorithm i is successful. If we are just interested in the number of wins, we need only compare the values of $p_i(1)$ for all the algorithms but, if we are interested in algorithms with a high probability of success, we should choose the ones for which p_i^* has the largest values. In our performance profile plots, we use a logarithmic scale in order to observe the performance of the algorithms over a large range of f while still being able to discern in some detail what happens for small f.

Whilst performance profiles are a very helpful tool when working with a large test set and several algorithms, as Dolan and Moré point out, they do need to be used and interpreted with care. This is especially true if we want to try and rank the algorithms in order. Our preliminary experiments for this study led us to re-examine performance profiles [27]. We found that, while they give a clear measure of which is the better algorithm for a chosen f and given set $\mathcal{T}$, if performance profiles are used to compare more than two algorithms, they determine which algorithm has the best probability $p_i(f)$ for f in a chosen interval, but we cannot necessarily assess the performance of one algorithm relative to another that is not the best using a single performance profile plot. Thus in Section 9, we limit some of our performance profiles to two solvers at a time.

2.4 Parameter setting

Where codes offer a number of options (such as orderings and scalings), we normally use the default or otherwise recommended settings; no attempt is made to tune the parameters for a particular problem (this would not be realistic given the size of the test set and number of solvers). However, it is recognised that, for some examples, choosing settings other than the defaults may significantly enhance performance (or adversely effect it) and, in practice, a user may find it advisable to invest time in experimenting with difference choices to try and optimize performance for his/her application. Details of the software we use are given in Section 8, together with the parameter settings.

3 Direct solvers

While the focus of our study is on preconditioning iterative methods for least-squares problems, it is of interest to look at how these methods perform in comparison to sparse direct methods. For the

normal equations, a positive definite solver that computes a sparse Cholesky factorization can be used, such as CHOLMOD [11] or HSL_MA87 [34]. Alternatively, there are sparse packages that can factorize both positive definite and indefinite systems. These include a number of HSL codes (notably, MA57 [18], HSL_MA86 [36], and HSL_MA97 [35]) as well as MUMPS [46], WSMP [30], PARDISO [52] and SPRAL_SSIDS [33]. Comparisons of some of these packages for solving sparse linear systems may be found in [23, 26]. When used to solve the augmented system, the solvers employ some kind of pivoting to try and ensure numerical stability, and this can impose a non trivial runtime overhead (as well as adding significantly to the complexity of the software and the memory requirements).

Most modern sparse direct solvers are designed to run in parallel, either through the use of MPI, OpenMP or GPUs. It is beyond the scope of the current study to compare the performance of direct solvers on least-squares problems; instead we opt to use HSL_MA97 (Version 2.3.0) for our comparisons with iterative methods. This choice was made since HSL_MA97 is a recent state-of-the-art package that is designed for multicore machines, it can solve both the normal equations and augmented system and, because we are responsible for its development, we find it convenient to use and to incorporate into our test environment. We note that CHOLMOD has an attractive feature that is not currently offered by any of the HSL codes which is that it can factor the normal equations without being given C explicitly; just providing A^T suffices and this saves memory.

For sparse QR, far fewer software packages have been developed. Those that are available include MA49 [2] from the mid 1990s and, much more recently, SuiteSparseQR [15] and qr_mumps [10]. Here we use SuiteSparseQR version 4.4.4 (for which we have written a Fortran interface).

A time performance profile comparing SuiteSparseQR (denoted by SPQR) and HSL_MA97 applied to the normal equations and the augmented system (denoted by MA97-normal and MA97-augmented, respectively) is given in Figure 3.1. In our experiments, one step of iterative refinement was used. We see that using HSL_MA97 for the augmented system leads to the smallest number of failures while using it to solve the normal equations is the fastest approach for almost half of the problems. The failures are for some of the largest problems and are because of insufficient memory (see Tables 3.25–3.27 in [28] for details). In addition, for SPQR there are two problems (mri1 and mri2) for which no error is flagged but the returned residuals are clearly far too large (10^{13} and 10^{24}, respectively). Although a direct solver such as HSL_MA77 [53] that allows the main work arrays and the matrix factors to be held out of core can extend the size of problem that can be solved, such solvers can be significantly slower. Thus there is a clear need for iterative solvers that require less memory.

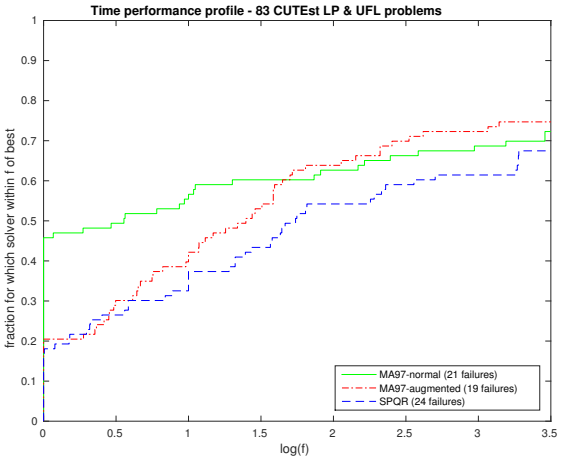

Figure 3.1: Time performance profile for the direct solvers HSL_MA97 (applied to the normal equations and the augmented system) and SuiteSparseQR (SPQR) for test set $\mathcal{T}$.

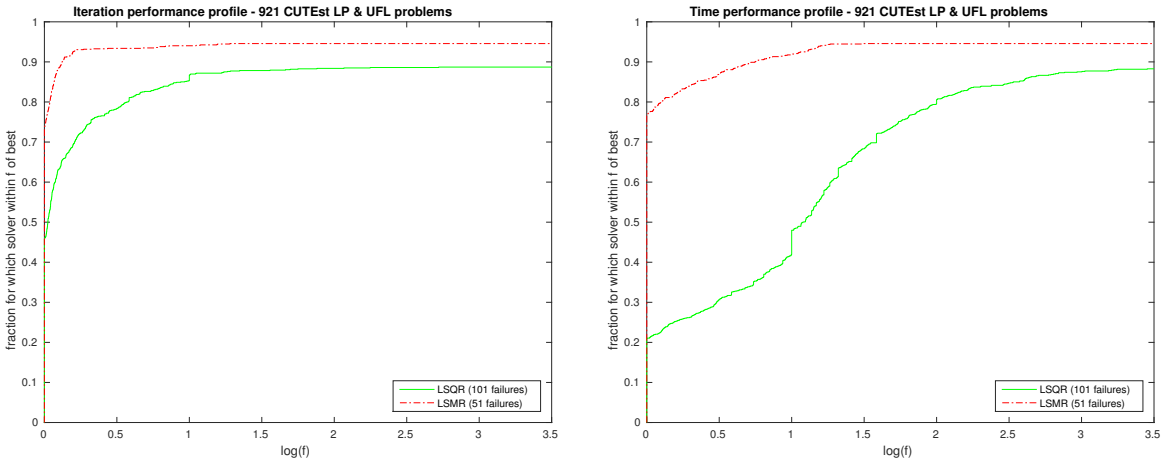

Figure 4.1: Iteration performance profile (left) and time performance profile for LSMR and LSQR (right) for the complete CUTEst and UFL test set of 921 eligible problems with no preconditioning.

4 LSQR vs LSMR

CGLS (or CGNR) [32] is a long-established extension of the conjugate gradient method (CG) to least-squares problems. It is mathematically equivalent to applying CG to the normal equations, without actually forming them. The well-known and widely used LSQR algorithm of Paige and Saunders [49, 50] is algebraically identical to CGLS but has more favourable numerical properties and in finite precision LSQR is capable of achieving more accurate results than CGLS and so in some applications LSQR is preferred. LSQR is based on the Golub-Kahan bidiagonalization of A and has the property of reducing $\|r_k\|_2$ monotonically, where $r_k = b - Ax_k$ is the residual for the approximate solution x_k.

The more recent LSMR algorithm of Fong and Saunders [22] is similar to LSQR in that it too is based on Golub-Kahan bidiagonalization of A. However, in exact arithmetic it is equivalent to MINRES [48] applied to the normal equations, so that the quantities $\|A^T r_k\|_2$ (as well as, perhaps more surprisingly, $\|r_k\|_2$) 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. Observe that if right-preconditioning with preconditioner M is employed, then $\|(AM^{-1})^T r\|_2$ is monotonic decreasing.

As already noted, the implementation of LSMR used in this paper is a slightly modified version of the one available from `http://web.stanford.edu/group/SOL/download.html`. The modifications include using allocatable arrays rather than automatic arrays (the latter can cause the code to crash with a segmentation fault error if the problem is large whereas allocated arrays allow memory failures to be captured and the code to be terminated with a suitable error flag set). More importantly, we incorporate a reverse communication interface that allows greater flexibility in how the user performs matrix vector products Ax and $A^T x$ and applies the (optional) preconditioner as well as enabling us to use our stopping criteria C1 and C2 (independently of the preconditioner used). The same modifications are made to LSQR for our tests.

In Figure 4.1, we present an iteration performance profile and a time performance for LSQR and LSMR with no preconditioning on the entire CUTEst and UFL set of 921 eligible examples. We see that LSMR has fewer failures compared to LSQR and requires a smaller number of iterations, which results in faster execution time. This confirms the findings of Fong and Saunders; in the remainder of this study we will limit our attention to LSMR.

Fong and Saunders propose incorporating local reorthogonalization in which each new basis vector is reorthogonalized with respect to the previous `localSize` vectors, where `localSize` is a user specified parameter. Setting `localSize` to 0 corresponds to no reorthogonalization while setting `localSize` to n

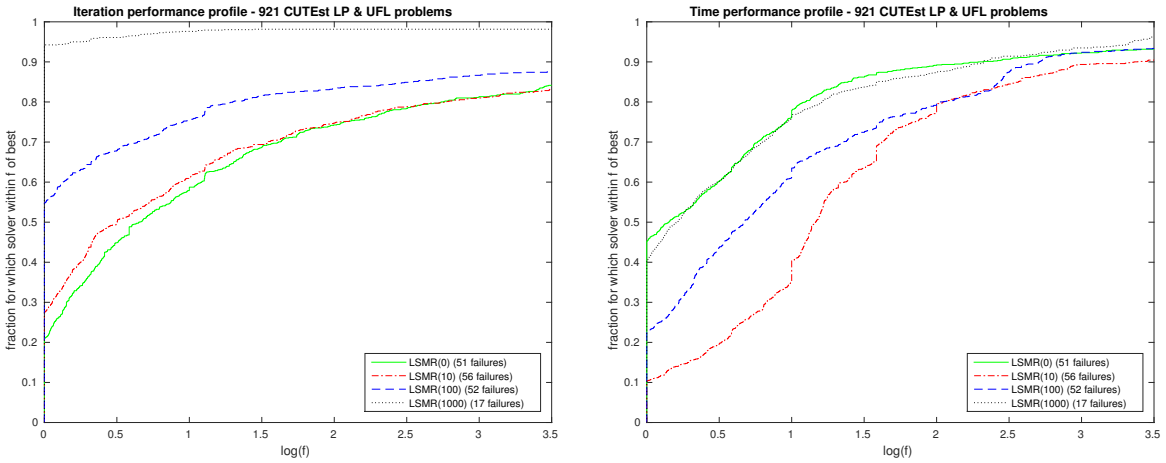

Figure 4.2: Iteration performance profile (left) and time performance profile for LSMR with a range of values of `localSize` for the complete CUTEst and UFL test set of 921 eligible problems with no preconditioning.

gives full reorthogonalization. Fong and Saunders report iteration counts for two linear programming problems with `localSize` set to 0, 5, 10, 50 and n. These illustrate that, compared with no reorthogonalization, setting `localSize` $= 10$ or 50 can lead to a worthwhile reduction in the number of iterations for convergence compared to no reorthogonalization but, as expected, more iterations are needed than for full reorthogonalization. Note that as n vectors of size `localSize` are needed, for large problems full reorthogonalization is impractical both in terms of the computational time and memory requirements.

To examine the effect of the reothogonalization parameter `localSize` on our much larger test set, an iteration performance profile and a time performance profile for `localSize` set to 0, 10, 100 and 1000 are given in Figure 4.2 (no preconditioning). We see that using a large value for `localSize` can significantly reduce the number of iterations and improve the success rate; the disadvantage is that the cost of each iteration (in terms of time and memory) increases with `localSize`.

5 Preconditioning strategies for normal equations

In this section, we first consider a number of ways to choose the preconditioner M for use with LSMR.

5.1 Diagonal preconditioning

The simplest form of preconditioning is diagonal preconditioning in which we we solve

$$\min_y \|b - ASy\|_2, \qquad x = Sy,$$

where S is a diagonal matrix that scales the columns of A to give them unit 2-norm. This requires only the diagonal entries of the normal matrix C to be computed or, equivalently, the squares of the 2-norms of the columns of A. This can be done in parallel, making the computation of the preconditioner and its application both straightforward and efficient (in terms of time and memory).

5.2 Incomplete Cholesky factorizations

Incomplete Cholesky (IC) factorizations have long been 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, [5, 56, 61] and the long lists of references therein. Since the normal equations (1.2) are positive definite, an obvious choice for the preconditioner is an IC factorization of the matrix C.

An IC factorization takes the form LL^T in which some of the fill entries (entries in L that were zero in the system matrix C) that would occur in a complete factorization are ignored. The preoconditioned normal equations become

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

Performing preconditioning operations involves solving triangular systems with L and L^T. The simplest example of an incomplete factorization is $IC(0)$ in which the lower triangular incomplete factor L only has entries in the same positions as in the lower triangular part of the system matrix. This is memory efficient but the resulting preconditioner is often not powerful enough to solve problems from practical applications (and where it does give convergence, the number of iterations required can be prohibitive). Thus, 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. Different variants are suitable for different problems but picking which to use can often come down to experience. Most recently, Scott and Tůma [60, 61] have proposed a limited memory approach that exploits ideas from the positive semidefinite Tismenetsky-Kaporin modification scheme [40, 65]. This involves the use of intermediate memory that is employed during the construction of L but is then discarded; the amount of memory used is controlled by the user. The basic scheme is based on a matrix factorization of the form

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

where L is the lower triangular matrix with positive diagonal entries that is used for preconditioning, $\hat{L}$ is a strictly lower triangular matrix with small entries that is used to stabilize the factorization process but is subsequently discarded, and E has the structure

$$E = \hat{L}\hat{L}^T.$$

On the j-th step of the factorization, the first column of the Schur complement is decomposed into a sum of two vectors

$$l_j + \hat{l}_j,$$

such that $|l_j|^T|\hat{l}_j| = 0$ (with the first entry in l_j nonzero), where l_j (respectively, $\hat{l}_j$) 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

$$(l_j + \hat{l}_j)(l_j + \hat{l}_j)^T.$$

However, the Tismenetsky incomplete factorization does not compute the full update as it does not subtract

$$E_j = \hat{l}_j\hat{l}_j^T.$$

Thus, the positive semidefinite modification E_j is implicitly added to C. To limit the memory required, drop tolerances are optionally used. In practice, the matrix C is optionally preordered and scaled and, if necessary, shifted to avoid breakdown of the factorization (which occurs if a zero or negative pivot is encountered). Thus the LL^T incomplete factorization of the matrix

$$\overline{C} = SQ^TCQS + \alpha I$$

is computed, where Q is a permutation matrix chosen on the basis of sparsity, S is a diagonal scaling matrix and α is a non-negative shift. It follows that $M = \overline{L}\overline{L}^T$ with $\overline{L} = QS^{-1}L$ is the incomplete factorization preconditioner.

Extensive numerical experiments reported in [60, 61] involving a large set of linear systems that come from a wide range of real-world applications demonstrate that the approach is efficient (in terms of time and memory) and, through the use of diagonal shifts [43], is robust, and thus it is this variant of IC that we employ in our study.

When used to compute an incomplete factorization of the normal equations, 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, used to perform the corresponding step of the incomplete Cholesky algorithm and then discarded. Of course, forming the normal matrix, even piecemeal, can entail a significant overhead (particularly if m and n are large and if A has one or more dense rows) and potentially may lead to a severe loss of information in very ill-conditioned cases.

5.3 MIQR

An alternative to an incomplete Cholesky factorization of C is an approximate orthogonal factorization of A. If

$$A \approx Q \begin{bmatrix} R \\ 0 \end{bmatrix},$$

where Q is orthogonal and R is upper triangular, then $C = A^T A \approx R^T R$ and, $M = R^T R$ can be used as a preconditioner. Again, applying the preconditioner involves triangular solves. Observe that the factor Q is not used. There have been a number of approaches based on incomplete orthogonal factorizations of A [38, 51, 55, 66]. Most recently, there is the Multilevel Incomplete QR (MIQR) factorization of Li and Saad [42].

When A is sparse, many of its columns are likely to be orthogonal because of their structure. These structurally orthogonal columns form an independent set S. Once S is obtained, the remaining columns of A are orthogonalized against the columns in S. Since the matrix of remaining columns will in general still be sparse, it is natural to recursively repeat the process until the number of columns is small enough to orthogonalize with standard methods, or a prescribed number of reductions (levels) has been reached, or the matrix cannot be reduced further. This process results in a QR factorization of a column-permuted A and forms the basis of the MIQR factorization. In practice, the QR factorization causes significant fill-in and so MIQR improves sparsity by relaxing the orthogonality and applying dropping strategies.

The MIQR algorithm does not require the normal matrix C to be computed explicitly; only one row of C is needed at any given time. Moreover, since C is symmetric, only its upper triangular part (i.e., the inner products between the i-th column of A and columns $i+1$ to n) needs to be calculated.

5.4 RIF

The RIF (Robust Incomplete Factorization) algorithm of Benzi and Tůma [6, 7] computes an LDLT factorization of the normal matrix C without forming any entries of C, working only with A. The method is based on C-orthogonalization, i.e., orthogonalization with respect to the C-inner product defined by

$$\langle x, y \rangle_C := x^T C y = (Ax)^T (Ay) \quad \text{for all} \quad x, y \in \mathbb{R}^n. \tag{5.2}$$

Assuming A is of full column rank, C is symmetric positive definite and this then provides an inner product on $\mathbb{R}^n$. Given the n linear independent vectors $e_1, e_2, ..., e_n$ (e_i is the i-th unit basis vector), a C-orthogonal set of vectors $z_1, z_2, ..., z_n$ is built using a Gram-Schmidt process with respect to (5.2). This can be written in the form

$$Z^T C Z = D = diag(d_1, d_2, ..., d_n), \tag{5.3}$$

where $Z = [z_1, z_2, ..., z_n]$ is unit upper triangular and the d_i are positive. It follows that $Z^T = L^{-1}$, where L is the unit lower triangular factor of the root-free Cholesky factorization $C = LDL^T$. It can be shown [6] that the L factor can be obtained as a by-product of the C-orthogonalization process at no extra cost.

Two different types of preconditioner can be obtained by carrying out the C-orthogonalization process incompletely. The first drops small entries from the computed vectors as the C-orthogonalization proceeds, resulting in a sparse matrix $\tilde{Z} \approx L^{-T}$; that is, an incomplete inverse factorization of C of the form

$$C^{-1} \approx \tilde{Z}\tilde{D}^{-1}\tilde{Z}^T,$$

where $\tilde{D}$ is diagonal with positive entries, is computed. This is a factored sparse approximate inverse that can be used as a preconditioner for the CG algorithm applied to the normal equations. The preconditioner is guaranteed to be positive definite and can be applied in parallel since its application requires only matrix-vector products. It is generally known as the stabilized approximate inverse (SAINV) preconditioner.

The second approach (the RIF preconditioner) is obtained by discarding the computed sparsified vector $\tilde{z}_i$ as soon as it has been used to form the corresponding parts of the incomplete factor $\tilde{L}$ of C. This gives an algorithm for computing an incomplete Cholesky factorization for the normal equations

$$C \approx \tilde{L}\tilde{D}\tilde{L}^T.$$

Again, the preconditioner $M = \tilde{L}\tilde{D}\tilde{L}^T$ is positive definite and (in exact arithmetic) breakdown during its computation is not possible. An important feature of the RIF preconditioner is that it incurs only modest intermediate storage costs, although implementing the algorithm for its construction so as to exploit the sparsity of A is far from straightforward (see [7] for a brief discussion). Benzi and Tůma report that the RIF preconditioner is generally more effective at reducing the number of CG iterations than the SAINV one and is thus the one included in this study. Over the past few years, a number of papers on preconditioners for least-squares problems have used RIF as a benchmark, but these papers limit their reported tests to a small number of examples [3, 9, 42, 44].

6 BA-GMRES

The BA-GMRES method for solving least-squares problems combines using a stationary inner iteration method with the Krylov subspace method GMRES [57] applied to the normal equations. In practice, restarted GMRES is used. In contrast to the other methods discussed so far, rather than forming an explicit preconditioner, a number of steps of a stationary iterative method are applied within the GMRES algorithm whenever an application of the preconditioner is needed. Such techniques are often called inner-outer iteration methods. While the basic idea is not new, it has recently been explored in the context of least-squares problems by Hayami et al. [31, 44, 45]. In particular, for overdetermined least-squares problems, they propose using Jacobi- (Cimmino [14]) and SOR-type (Kaczmarz [39]) iterative methods as inner-iteration preconditioners for GMRES and advocate their so-called BA-GMRES approach for the efficient solution of rank-deficient problems. Jacobi iterations can be advantageous for parallel implementations but in this study, we limit our attention to serial implementations and use SOR iterations with automatic selection of the relaxation parameter ω as described in [45].

BA-GMRES corresponds to GMRES applied to

$$\min_x \|Bb - BAx\|_2, \tag{6.1}$$

where the rectangular matrix $B \in \mathbb{R}^{n\times m}$ is the (left) preconditioner. Morikuni and Hayami [44, 45] show that if B satisfies $\mathcal{R}(A) = \mathcal{R}(B^T)$ and $\mathcal{R}(A^T) = \mathcal{R}(B)$, the solution of (6.1) is also a solution of the least-squares problem (1.1). B is not formed or stored explicitly. Instead, at each GMRES iteration k, when preconditioning is needed a fixed number of steps of a stationary iterative method are applied to a system of the form

$$A^T Az = A^T Av_k$$

to obtain $\tilde{z}$ for a given v_k, which is used to compute the next GMRES basis vector v_{k+1}. Thus at each GMRES iteration, another system of normal equations is solved approximately using a stationary

iterative method and this can be done without forming any entries of A^TA explicitly (see [56], Section 8.2 for details); all that is required are repeated matrix-vector products with A and A^T. This allows nonsymmetric preconditioning for least-squares problems. Another potential advantage of BA-GMRES is that it avoids forming and storing an incomplete factorization; the memory used is determined solely by the number of steps of GMRES that are applied before restarting.

Morikuni and Hayami observe that inner iteration preconditioners can also be applied to CGLS and LSMR. This has the merit of using only short-term recurrences and so the memory requirements are less than for BA-GMRES. The results reported in [44, 45] for a small set of test problems (including rank-deficient examples) indicate faster times, fewer iterations and greater robustness using BA-GMRES; thus BA-GMRES (for which software is available, see Section 8.5) is used in this study.

7 Preconditioning strategies for the augmented system

An alternative to preconditioning the normal equations is to precondition the augmented system (1.3). In some applications, preconditioning the augmented system is advocated when the normal equations are highly ill-conditioned (see, for instance, [47]). A number of possible approaches exist, including employing an incomplete factorization designed for general indefinite symmetric systems or a signed incomplete Cholesky factorization [62] designed specifically for systems of the form (1.3). Chow and Saad [13] considered the class of incomplete LU preconditioners for solving indefinite problems and later Li and Saad [41] integrated pivoting procedures with scaling and reordering. Building on this, Greif, He, and Liu [29] recently developed a new incomplete factorization package SYM-ILDL for general sparse symmetric indefinite matrices. The factorization is of the form

$$K \approx LDL^T, \tag{7.1}$$

where 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). For SYM-ILDL, K may be any sparse indefinite matrix; no advantage is made of the specific block structure of (1.3). Independently, Scott and Tůma [63] 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. The SYM-ILDL software is available [29]. It is written in C++ and is designed either to be called from within MATLAB or from the command line. The user may save the computed factor data to a file but (when used from the command line) the package offers no procedure to take that data and use it as a preconditioner. Without substantial further work to set up a more flexible and convenient user interface, we were restricted to running individual problems one at a time. We performed limited experiments using SYM-ILDL (see also [62, 63]). These demonstrated that there are least-squares problems for which SYM-ILDL is able to provide an effective preconditioner but for other problems we were unsuccessful in obtaining the required least-squares solution. The prototype code of Scott and Tůma likewise gave very mixed results. We conclude that further work is needed for these codes to be useful for least-squares problems; they are not explored further in this study.

For matrices K of the augmented form (1.3), Scott and Tůma [62] propose extending their limited memory IC approach to a limited memory signed incomplete Cholesky factorization of the form (7.1) 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

$$\overline{K} = SQ^TKQS + \begin{bmatrix} \alpha_1 I & \\ & -\alpha_2 I \end{bmatrix}$$

is computed, where Q is a permutation matrix, S is a diagonal scaling matrix, and α_1 and α_2 are non-negative shifts chosen to prevent breakdown. The preconditioner is $M = \overline{L}D\overline{L}^T$, with $\overline{L} = QS^{-1}L$. In this case, the permutation Q is chosen 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; see [62] for details of this so-called constrained ordering.

An important advantage of a signed IC factorization over a general indefinite incomplete factorization is its simplicity in that it avoids the use of numerical pivoting. If we use the natural order ($Q = I$), the factorization becomes

$$K \approx \begin{bmatrix} I & \\ L_1 & L_2 \end{bmatrix} \begin{bmatrix} I & \\ & -I \end{bmatrix} \begin{bmatrix} I & L_1^T \\ & L_2^T \end{bmatrix}$$

and so

$$L_1 \approx A^T \text{ and } L_1 L_1^T \approx L_2 L_2^T.$$

If we choose $L_1 = A^T$ then this reduces to an IC factorization of the normal equations. However, by choosing $L_1 \neq A^T$ or $Q \neq I$, this approach can exploit the structure of the augmented system while avoiding the normal equations.

As the signed IC preconditioner is indefinite, a general non symmetric iterative method such as GMRES [57] is needed; we use right preconditioned restarted GMRES. 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 either of the stopping conditions C1 or C2 (which are based on A) is satisfied; they can, of course, be checked once GMRES has terminated. Instead, we use the scaled backward error

$$\frac{\|Ky_k - c\|_2}{\|c\|_2} < \delta_3, \tag{7.2}$$

where y_k is the computed solution on the kth step. In our experiments we set $\delta_3 = 10^{-8}$.

If we want to use a solver that is designed for symmetric indefinite systems, in place of GMRES we can use MINRES [48]. However, MINRES requires a positive definite preconditioner and so we use $M = \overline{LL}^T$, that is, we replace the entries -1 in D by $+1$ so that D becomes the identity. Again, the stopping conditions C1 or C2 cannot be checked inside MINRES and we use instead (7.2).

8 Preconditioning software and parameter settings

8.1 Diagonal preconditioning

In Figure 8.1 we present iteration and time performance profiles for LSMR with diagonal preconditioning using a range of values for the LSMR reorthogonalization parameter `localSize`. A large value reduces the iteration count but increases the time (and memory) required (so that a number of problems exceed the time limit if `localSize` is set to 1000, which accounts for the increase in the number of failures).

8.2 IC preconditioner for normal equations

A software package `HSL_MI35` that implements the limited memory IC algorithm discussed in Section 5.2 for the normal equations has been developed for the HSL mathematical software library [37]. This code is a modified version of `HSL_MI28` [60]. 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 C may be dense, or close to dense, this change was needed to keep L sparse). In addition, the user may either supply the matrix A and call a subroutine within the package to form C explicitly or, to save memory, A may be passed directly to the factorization routine. In this case, the lower triangular part of each column of the (permuted) normal matrix is computed as needed during the factorization (although the sparsity pattern of C is computed if reordering is selected). Note that, if A and not C is supplied, the range of scaling options is restricted since the equilibration and maximum matching-based scalings that are offered through the use of the packages `MC77` [54] and `MC64` [19, 20], respectively, require C explicitly. The default scaling is l_2 scaling, in which column j of

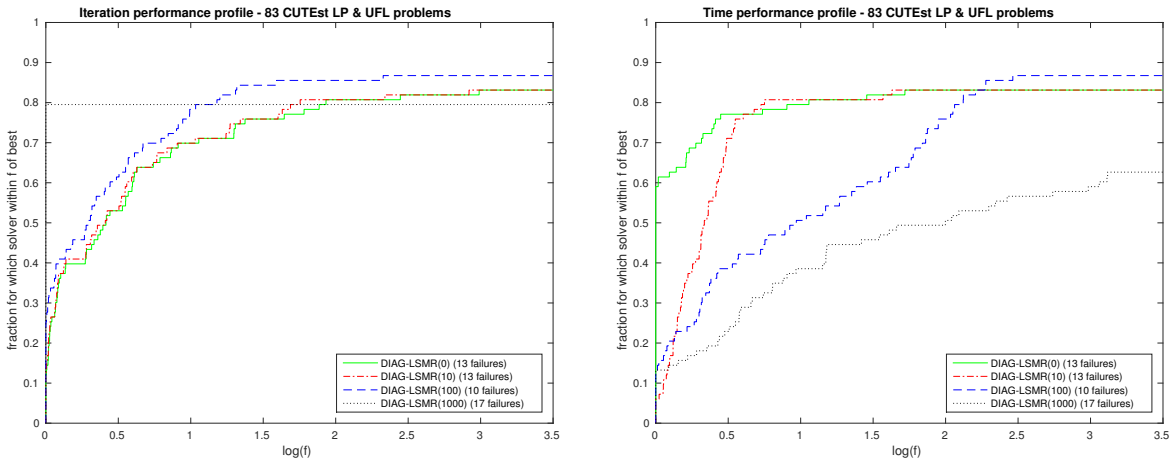

Figure 8.1: Iteration performance profile (left) and time performance profile (right) for LSMR with diagonal preconditioning using a range of values of `localSize` for test set $\mathcal{T}$.

C is normalised by its 2-norm; this needs only one column of C at a time. We observe that `HSL_MI35` is designed to solve the weighted least-squares problem but in our tests the weights are set to one.

The parameters `lsize` and `rsize` respectively control the maximum number of entries in each column of L and each column of the matrix $\hat{L}$ that is used in the computation of L (recall (5.1)). Iteration and time performance profiles for LSMR preconditioned by `HSL_MI35` using `lsize` = `rsize` = 10 and `lsize` = `rsize` = 20 are given in Figure 8.2. We see that the iteration count is reduced by increasing the number of entries allowed and as the time is not significantly adversely effected, `lsize` = `rsize` = 20 is used in all other experiments with `HSL_MI35`.

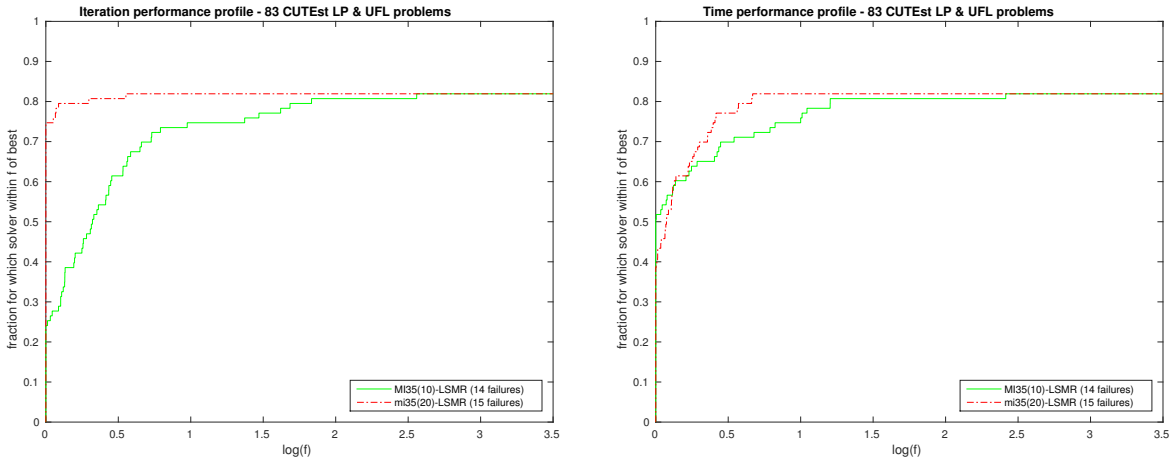

Figure 8.2: Iteration performance profile (left) and time performance profile (right) for LSMR preconditioned by `HSL_MI35` with `lsize` = `rsize` = 10 and `lsize` = `rsize` = 20 for test set $\mathcal{T}$.

In Figure 8.3 we present iteration and time performance profiles for LSMR preconditioned by `HSL_MI35` using a range of values for the LSMR reorthogonalization parameter `localSize`. As expected, using a large value reduces the iteration count but increases the time (and memory) required; `localSize` = 10 is used in all other experiments with `HSL_MI35`.

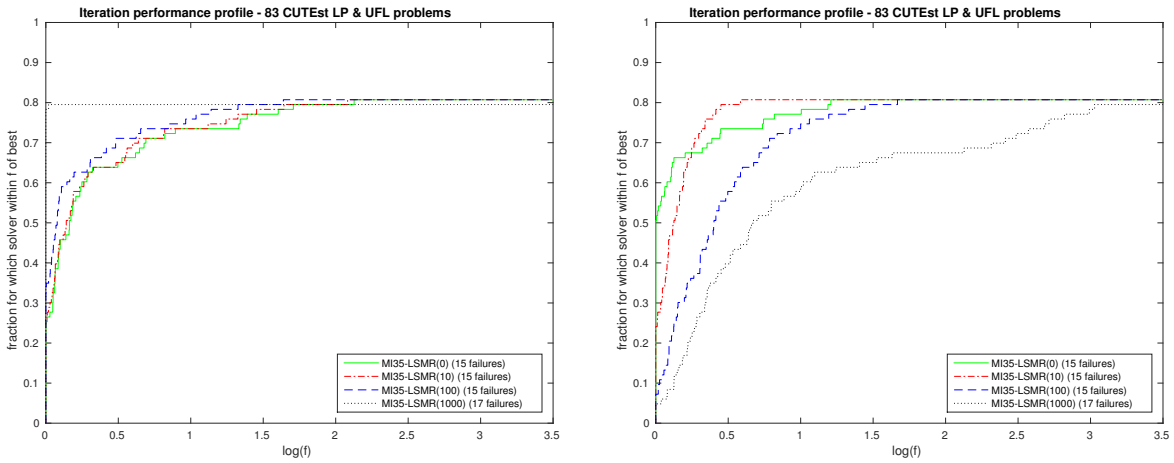

Figure 8.3: Iteration performance profile (left) and time performance profile (right) for LSMR preconditioned by `HSL_MI35` using a range of values of `localSize` for test set $\mathcal{T}$.

8.3 MIQR

The MIQR package available at `http://www-users.cs.umn.edu/~saad/software/` is for solving least-squares systems by a preconditioned CGNR algorithm and is written in C. As all our experiments are performed in Fortran, we have chosen to use a Fortran version of MIQR that is available from the GALAHAD optimization software library [25]. This is essentially a translation of Li and Saad [42]'s code, but with extra checks and features to make the problem data input easier. Key parameters, such as the maximum number of recursive levels of orthogonalization, the required angles between approximately orthogonal columns, the drop tolerance, and the maximum number of fills permitted per column, are precisely as given by Li and Saad.

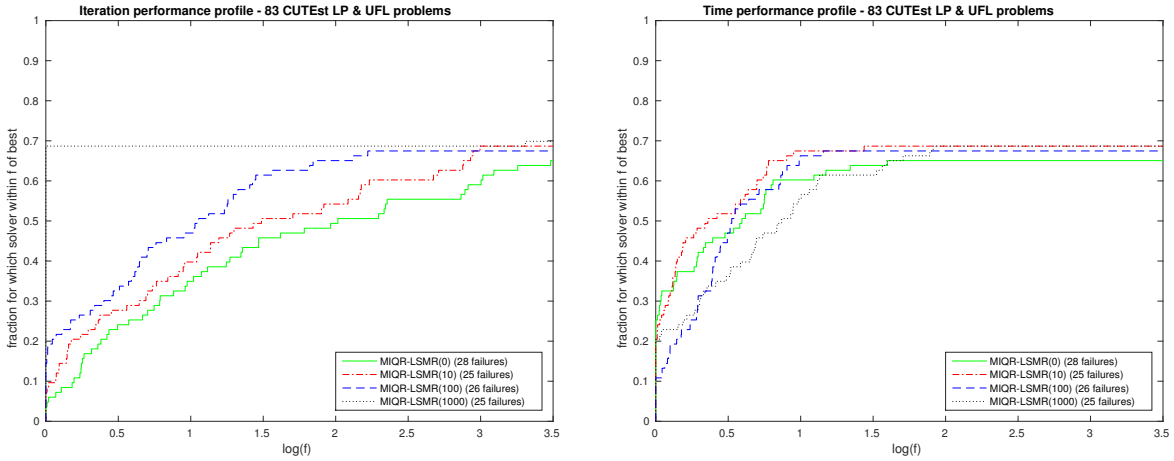

Figure 8.4: Iteration performance profile (left) and time performance profile (right) for LSMR with MIQR preconditioning using a range of values of `localSize` for test set $\mathcal{T}$.

Figure 8.5 presents iteration and time performance profiles for MIQR-preconditioned LSMR using a range of values of the reorthogonalization parameter `localSize`. The number of failures appears relatively insensitive to the choice of `localSize` but the iteration count decreases as `localSize` increases while using a value of 10 is the best in terms of time.

8.4 RIF

A right-looking implementation of RIF is available at `http://www2.cs.cas.cz/~tuma/sparslab.html`. However, for our tests, Tůma provided a more recent left-looking version. This works only with A and A^T. As full documentation for the software is lacking, we relied on Tůma for advice on the parameter settings; in particular, we used absolute dropping with a drop tolerance of 0.1. In Figure 8.5 we give iteration and time performance profiles for RIF-preconditioned LSMR using a range of values of the reorthogonalization parameter `localSize`.

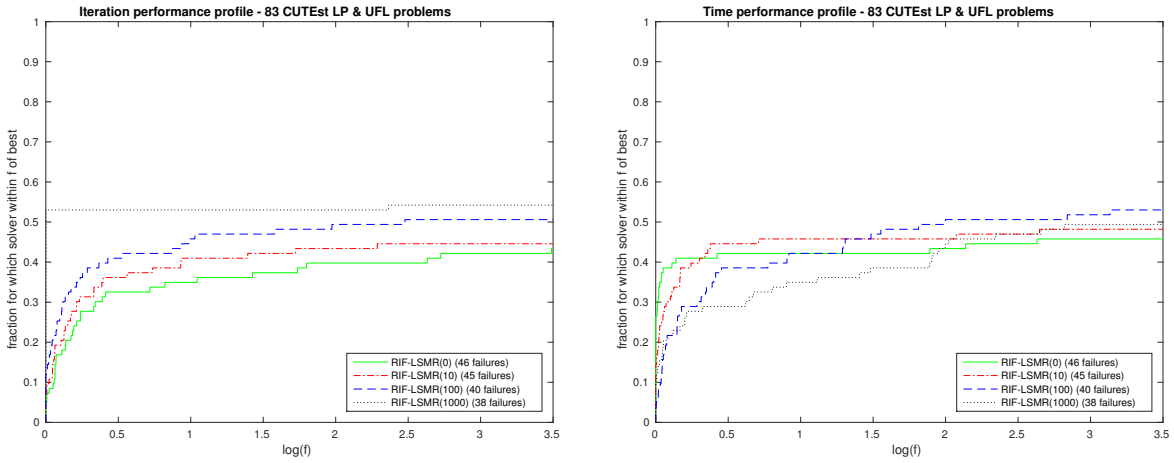

Figure 8.5: Iteration performance profile (left) and time performance profile (right) for LSMR with RIF preconditioning using a range of values of `localSize` for test set $\mathcal{T}$.

8.5 BA-GMRES

There are codes for the BA-GMRES method preconditioned by NR-SOR inner iterations developed by Morikuni available at `http://researchmap.jp/KeiichiMorikuni/Implementations` (March 2015). However, these are not in the form that we can readily use for large-scale testing purposes. In particular, they employ automatic arrays (and will thus fail for a very large problem for which there is insufficient memory) and they contain "stop" statements (so again, they can fail without prior warning). As a result, we implemented a modified version of BA-GMRES. This also allowed us to use the stopping criteria C1 and C2 for consistency with the preconditioned LSMR tests (as in our tests with other methods, the time for computing the residuals needed for checking C1 and C2 at each iteration are excluded from the reported times).

As restarted GMRES is employed, the user must choose the number `gmres_its` of iterations between restarts. A compromise between a large value that reduces the overall number of iterations and a small value that limits the storage should be used. We performed some preliminary experiments to try and choose a suitable value to use for all our tests; our findings are in Figure 8.6. On the basis of these, we set `gmres_its` = 1000. Note that if the number (iter) of iterations required for convergence is less than `gmres_its`, so that we do not unfairly overestimate the memory required, the reported memory for BA-GMRES is for `gmres_its` = iter. Following Morikuni, our implementation of BA-GMRES allows the user to choose between using NR-SOR and Cimmino inner iterations. For the former, the user may supply the number of inner iterations and the relaxation parameter; otherwise, these are computed automatically using the procedure proposed in [45]. We use NR-SOR inner iteration with automatic parameter selection in our tests.

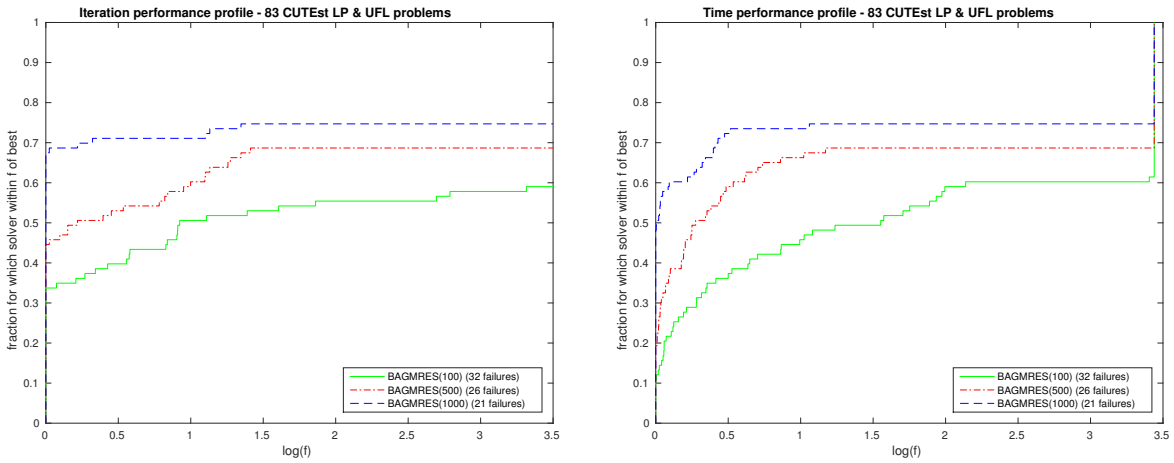

Figure 8.6: Iteration performance profile (left) and time performance profile for BA-GMRES with different restart parameters for test set $\mathcal{T}$.

8.6 Signed IC preconditioner: augmented system

A software package `HSL_MI30` that implements the limited memory signed IC algorithm discussed in Section 7 for the augmented system is available within HSL; details are given in [62]. In our tests, we use the default settings for `HSL_MI30` and the parameters `lsize` and `size` that control the number of entries in L and the intermediate memory used to compute the factorization are both set to 20. For GMRES and MINRES we use the HSL implementations `MI24` (with the restart parameter set to 1000) and `HSL_MI32`, respectively.

9 Solver comparison results

9.1 Performance comparison for preconditioning LSMR

Figure 9.1 presents iteration and time performance profiles for LSMR run both without preconditioning and with diagonal, MIQR, RIF and IC (`HSL_MI35`) preconditioning. Here we chose `localSize` $= 0$ for no

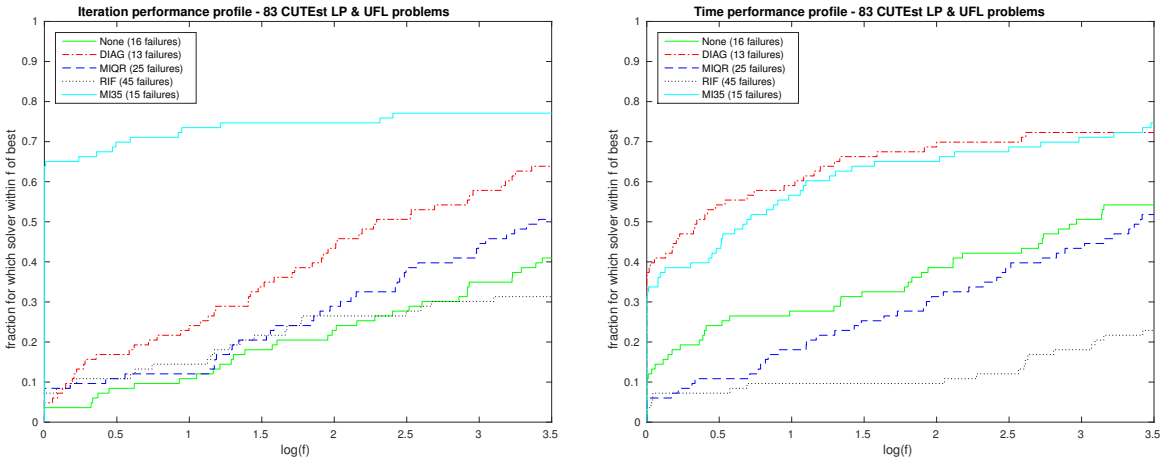

Figure 9.1: Iteration performance profile (left) and time performance profile for different preconditioners used with LSMR for test set $\mathcal{T}$.

preconditioning and diagonal preconditioning and `localSize` = 10 for MIQR, RIF and IC preconditioning since these appeared to give the best (time) performances in the individual preconditioner comparisons reported in Sections 4 and 8. We see that, in terms of iteration counts, the incomplete factorization is the best preconditioner but, in terms of time, the simplest option of diagonal preconditioning is slightly better than IC preconditioning (and has the advantages of needing minimal memory and being trivially parallelizable). The close time-ranking of the diagonal and IC preconditioners is confirmed in Figure 9.2. We observe that Morikuni and Hayami [44] also found diagonal preconditioning to give the fastest solution times in some of their tests.

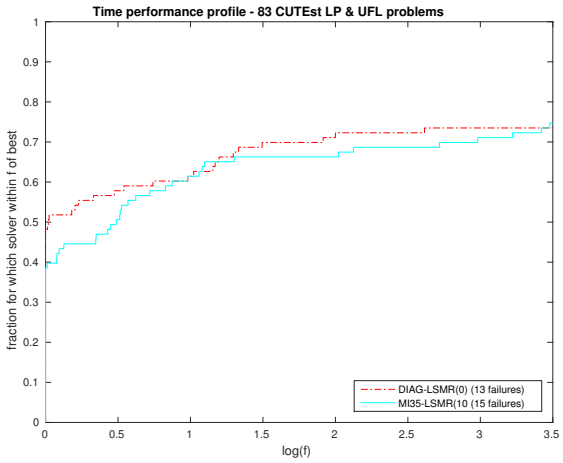

Figure 9.2: Time performance profile for diagonal and IC preconditioners used with LSMR for test set $\mathcal{T}$.

In Figure 9.3 we compare the remaining three preconditioners. We see that in terms of time MIQR preconditioning is broadly similar to running without a preconditioner, and that the effects of a reduction in iteration counts for the former is balanced by the cost of computing and applying the preconditioner. This is reinforced in Figure 9.4 when RIF is removed from the picture.

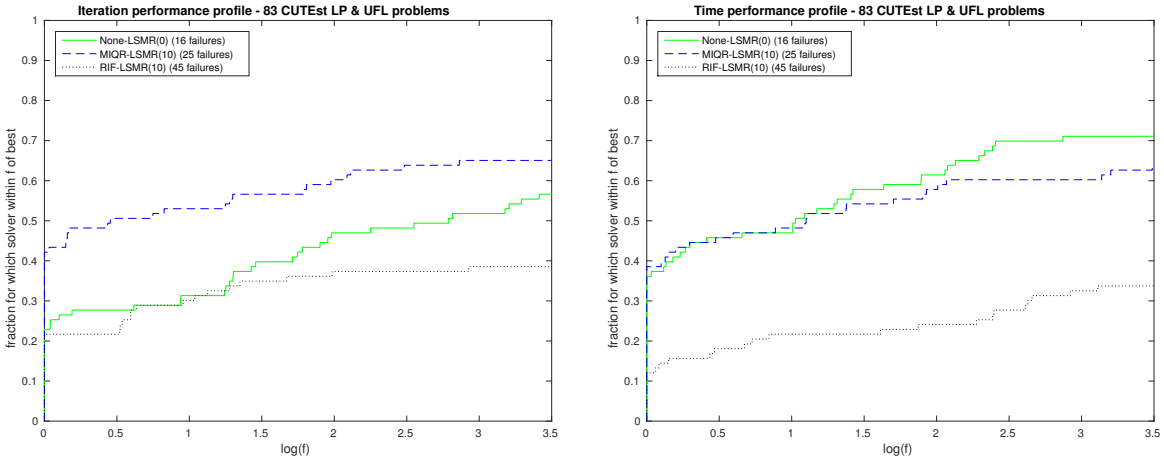

Figure 9.3: Iteration and time performance profiles for LSMR with no preconditioning and MIQR and RIF preconditioning for test set $\mathcal{T}$.

The current implementation of RIF is somewhat slow. For problems for which the RIF preconditioner performs reasonably well (including the IG5-1x problems), more than 95% of the total solution time can be spent on computing the preconditioner, even though it can be significantly sparser than that computed

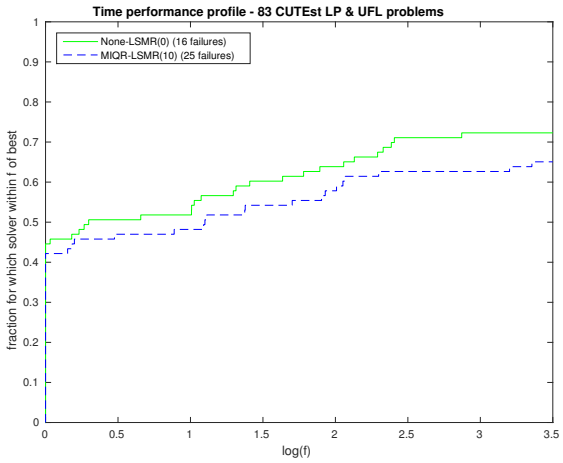

Figure 9.4: Time performance profile for LSMR with no preconditioning and MIQR preconditioning for test set $\mathcal{T}$.

using `HSL_MI35` or MIQR. The uncompetitive construction time appears to be largely attributable to the searches performed to determine which C-inner products need to be computed; this is currently a subject of separate investigation [64]. For 21 of the 83 test problems, computing the RIF preconditioner exceeded our time limit of 600 seconds. Furthermore, for our test set $\mathcal{T}$ as a whole and the current settings, RIF is not especially effective. For the 62 problems for which the RIF preconditioner was successfully computed, 22 went on to exceed the LSMR iteration limit and a further 2 exceeded the total time limit. Again, this is consistent with [44]. We observe, however, that in many cases the RIF preconditioner is sparser than, for example, the IC preconditioner. Using a smaller drop tolerance may improve the quality at the cost of more fill but the time to compute the preconditioner can also increase significantly.

9.2 Performance comparison with BA-GMRES

Time performance profiles for BA-GMRES are given in Figure 9.5. We see that, on our test set, BA-GMRES is slower than using LSMR with diagonal or IC preconditioning but is faster than LSMR with no preconditioning and MIQR preconditioning. However, a closer look at the results (see the summary tables given in the Appendix and [28]) shows that BA-GMRES is able to efficiently solve some examples that preconditioned LSMR and the direct solvers struggle with. In particular, BA-GMRES performs strongly on the `GL7dxx` problems and solves problem SPAL_004 in only one iteration. However, it is poor for the `pseex` problems.

9.3 Performance comparison with signed incomplete factorization

In Figure 9.6, time performance profiles are given for solving the augmented system using the signed incomplete Cholesky factorization preconditioner (`HSL_MI30`) run with GMRES(1000) and MINRES; the IC preconditioner (`HSL_MI35`) for the normal equations run with LSMR is also included. We see that `HSL_MI35` preconditioned LSMR is faster than solving the augmented system and has the least number of failures. Note that the number of entries in the factors for the normal equations is approximately $n \times$ `lsize` whereas for the augmented system the number is bounded above by $m + nz(A) + (m+n) \times$ `lsize` (where $nz(A)$ is the number of entries in A). Thus when working with the augmented system each application of the preconditioner is considerably more expensive.

As observed in Section 7, for the signed incomplete factorization run with MINRES or GMRES, the stopping criteria is the scaled backward error for the augmented system (1.3) and thus conditions C1 and/or C2 may not be satisfied. For a significant portion of our test set, if δ_3 in (7.2) is set to be 10^{-8}

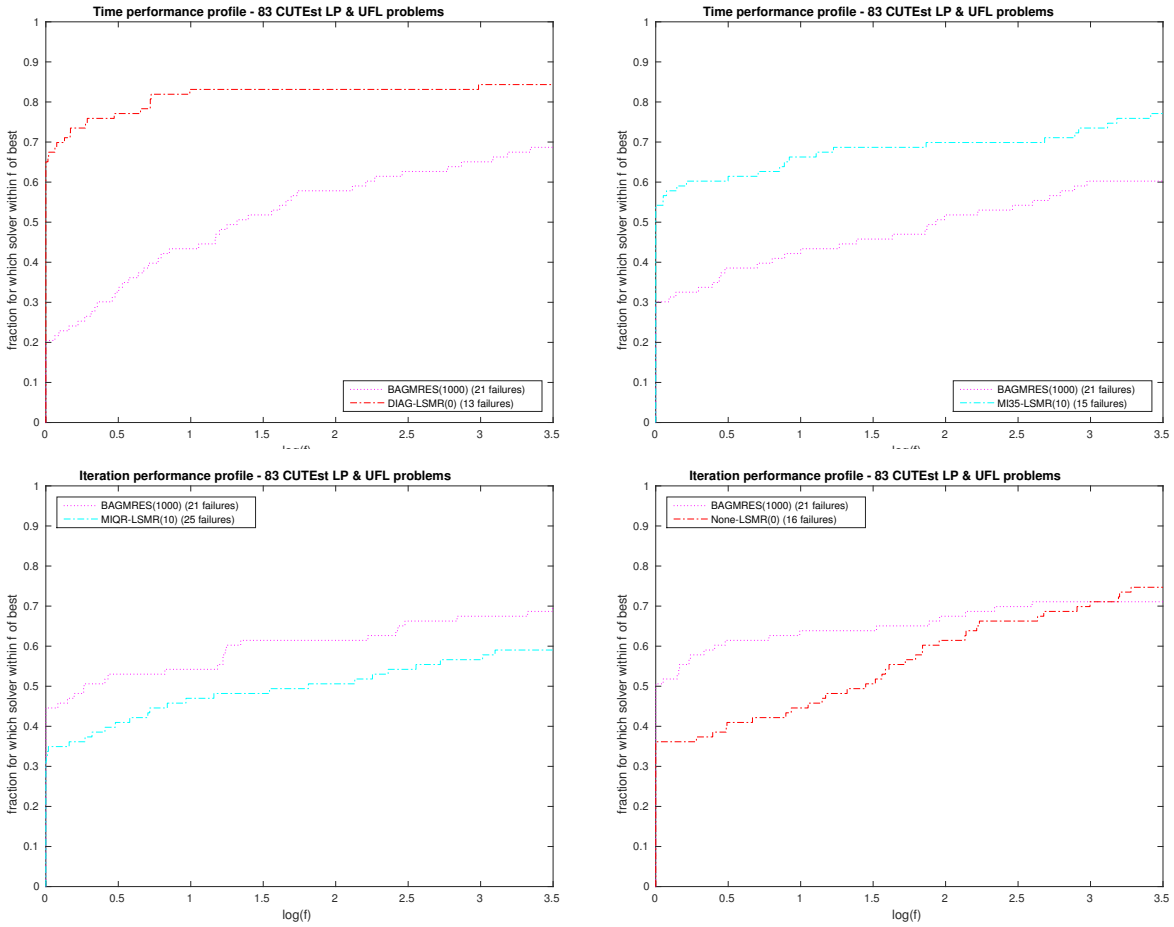

Figure 9.5: Time performance profile for BA-GMRES(1000) and LSMR with diagonal, IC (`HSL_MI35`), MIQR and no preconditioning for test set $\mathcal{T}$.

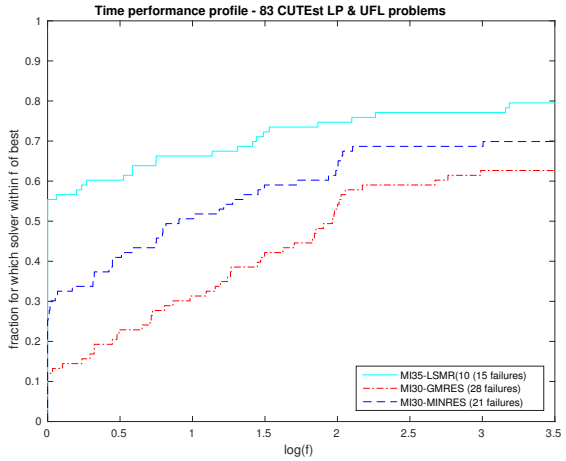

Figure 9.6: Time performance profile for LSMR with IC (`HSL_MI35`) preconditioning and GMRES(1000) and MINRES with signed IC (`HSL_MI30`) preconditioning for test set $\mathcal{T}$.

then either C1 or C2 is satisfied (see Tables 3.25 and 3.26 in [28]). Indeed, in some cases where we report a failure because the time limit or iteration count limit has been reached without satisfying (7.2), C1 or C2 is actually satisfied and for other examples, a larger value of δ_3 would still have resulted in C1 or C2 holding (and thus our reported iteration counts and total times can sometimes be larger than necessary). However, for some problems, including the TFxx examples, a smaller δ_3 is needed to satisfy C1 or C2. For example, for MINRES with $\delta_3 = 10^{-11}$, C1 is satisfied for problems TF14 and TF15 (the iteration counts increase from 1987 and 1107 to 10,700 and 46,341, respectively, which are similar to those needed by LSMR with `HSL_MI35`). But for the other TFxx problems, the number of iterations needed to satisfy C1 exceeds our limit of 100,000. Note that we were unable solve TF17, TF18 and TF19 to the required accuracy (with our time and iteration count limits) using any of the direct solvers or preconditioners in this study.

9.4 Performance comparison with a direct solver

In Figure 9.7, we present time performance profiles for the direct solver `HSL_MA97` applied to the normal equations and the diagonal and IC (`HSL_MI35`) preconditioned LSMR. We see that for the set $\mathcal{T}$ the parallel direct solver is the fastest for more than 50% of the problems, but it is unable to solve 25% of the problems for which there was insufficient memory. Moreover, the difference in the run times could potentially be significantly reduced (especially for large problems) if we parallelize the matrix vector products required by LSMR. Efficient parallel implementations depend both on the matrix structure and on the machine architecture. This is a separate subject of research and is outside the scope of this study. The performance of the iterative solvers with incomplete factorization preconditioners may also be enhanced by employing parallel triangular solves, which has recently be considered with some potentially encouraging results by Chow and Patel [12].

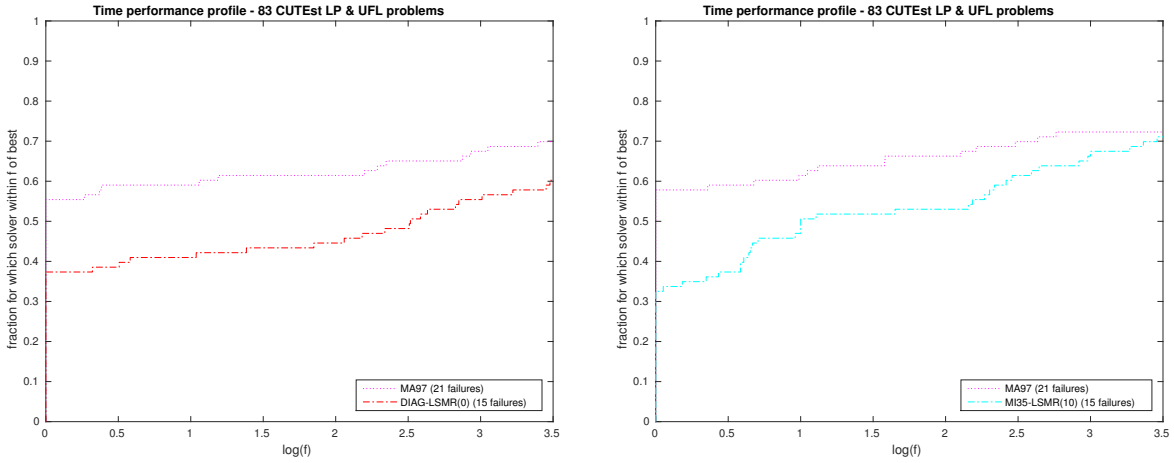

Figure 9.7: Time performance profile for direct solver `HSL_MA97` and diagonal and IC (`HSL_MI35`) preconditioned LSMR for test set $\mathcal{T}$.

9.5 Summary tables

In Tables A.2–A.5, we present summary data that allows a direct comparison of a particular statistic across the range of methods considered. We remove SPQR and `HSL_MA97` (for the augmented system) as these perform less well than `HSL_MA97` (for the normal equations). Similarly, MINRES preconditioned by `HSL_MI30` is included (denoted by `MI30-MIN`) while GMRES preconditioned by `HSL_MI30` is omitted. Full results for all methods (including those omitted here) may be found in [28]. For the iterative methods, we have selected what appears to be the "best" global choice of `localsize` or `gmres_its` as appropriate; these

are `localsize`=0 for the un-preconditioned and diagonal LSMR, `localsize`=10 for the MIQR, RIF, and `HSL_MI35` versions, and `gmres_its`=1000 for BA-GMRES (denoted in the tables by BA-G). We summarise the storage required for the factors (and for GMRES), the number of iterations performed, the elapsed time required to build the preconditioner and the total elapsed time to solve the problem. Note that, in Table A.3, the iteration count for BA-GMRES is the number of GMRES iterations whereas for the other methods it is the LSMR iteration count; the direct solvers are not included in this table since the iteration count is always 1. A − indicates that the run was unsuccessful; again, for full details the reader is referred to [28].

10 Concluding remarks

In this study, we have compared the performances of a number of preconditioning techniques for sparse linear least-squares problems. Our main tool has been performance profiles, but the complete numerical results are also available [28]. The findings of our study confirm that preconditioning least-squares problems is hard and that at present there is no single approach that works well for all problems; we thus conclude that there is scope for considerable further developments in this area. We have found that, in many cases, diagonal preconditioning performs as well as or better than more sophisticated approaches and, as it is very simple to implement and to apply (and can be used in parallel), we would suggest trying diagonal preconditioning first. Investigating extending simple diagonal preconditioning to a block diagonal approach (combined with a preordering step) would be interesting (note that block diagonal preconditioning is currently offered by the Ceres non-linear least-squares solver [1]). In terms of iteration counts, using an incomplete factorization of the normal equations performs well and, as we would expect since diagonal preconditioning can be regarded as a special case in which only one entry per row/column is retained, it generally requires far fewer iterations than diagonal preconditioning.

We observe that the direct solvers and the incomplete factorization codes `HSL_MI30` and `HSL_MI35` include options for scaling (and use scaling by default) whereas the software for MIQR, RIF and BA-GMRES that is currently available does not offer scaling. It would be of interest in the future to examine how much the performance of these approaches can be improved by the incorporation of scaling.

A further contribution of this study has been a detailed comparison of the LSQR and LSMR methods and of the effect of local reorthogonalization within LSMR. Our findings have confirmed those of Fong and Saunders [22] and have shown that the choice of the best local reorthogonalization parameter is problem and preconditioner dependent and also depends on whether reducing the iteration count or the total time is the primary objective.

Finally, we observe that a number of other approaches have been proposed in recent years, including the limited memory preconditioner (LMP) of Bellavia, Gonzio and Morini [4] and the balanced incomplete factorization (BIF) preconditioner of Bru, Marín Mas and Tůma [9]. LU preconditioning, which was discussed by Saunders [58] in 1979 (see also Section 7.5.3 of the book by Björck [8]), has also received renewed attention (see the 2015 paper by Arioli and Duff [3] and presentation by Saunders [59]). These are not included in this study since implementations that allow timings that are suitable for making fair comparisons with our software are not currently available and the algorithms are sufficiently complicated for it to be infeasible for us to develop efficient implementations for use here. Note that in [3] and [4], experimental results are reported using MATLAB codes. Unfortunately, the recent Fortran results reported by Saunders [59] do not encourage us to expect that the LU approach will be efficient in terms of time. But it would be interesting to see if it can be used to solve some of the examples that are currently intractable.

Acknowledgements

We are grateful to Michael Saunders for a number of discussions related to his LSQR and LSMR software packages and to Miroslav Tůma for help with employing his RIF code.

References

[1] S. Agarwal, K. Mierle, and Others, *Ceres solver*. `http://ceres-solver.org`.

[2] P. Amestoy, I. S. Duff, and C. Puglisi, *Multifrontal QR factorization in a multiprocessor environment*, Numerical Linear Algebra with Applications, 3 (1996), pp. 275–300.

[3] M. Arioli and I. S. Duff, *Preconditioning linear least-squares problems by identifying a basis matrix*, SIAM J. on Scientific Computing, 37 (2015), pp. S544–S561.

[4] S. Bellavia, J. Gondzio, and B. Morini, *A matrix-free preconditioner for sparse symmetric positive definite systems and least-squares problems*, SIAM J. on Scientific Computing, 35 (2013), pp. A192–A211.

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

[6] M. Benzi and M. Tůma, *A robust incomplete factorization preconditioner for positive definite matrices*, Numerical Linear Algebra with Applications, 10 (2003), pp. 385–400.

[7] ———, *A robust preconditioner with low memory requirements for large sparse least squares problems*, SIAM J. on Scientific Computing, 25 (2003), pp. 499–512.

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

[9] 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.

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

[11] 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.

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

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

[14] G. Cimmino, *Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari*, Ric. Sci. Progr. tecn. econom. naz., 9 (1939), pp. 326–333.

[15] 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.

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

[17] E. D. Dolan and J. J. Moré, *Benchmarking optimization software with performance profiles*, Mathematical Programming, 91 (2002), pp. 201–213.

[18] 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.

[19] 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.

[20] ———, *On algorithms for permuting large entries to the diagonal of a sparse matrix*, SIAM J. on Matrix Analysis and Applications, 22 (2001), pp. 973–996.

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

[22] 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.

[23] N. I. M. Gould, Y. Hu, and J. A. Scott, *A numerical evaluation of sparse direct symmetric solvers for the solution of large sparse, symmetric linear systems of equations.*, ACM Transactions on Mathematical Software, 33 (2007). Article 10, 32 pages.

[24] 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.

[25] N. I. M. Gould, D. Orban, and Ph. L. Toint, GALAHAD—*a library of thread-safe fortran 90 packages for large-scale nonlinear optimization*, ACM Transactions on Mathematical Software, 29 (2003), pp. 353–372.

[26] N. I. M. Gould and J. A. Scott, *A numerical evaluation of HSL packages for the direct solution of large sparse, symmetric linear systems of equations*, ACM Transactions on Mathematical Software, 30 (2004), pp. 300–325.

[27] 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.

[28] ———, *The state-of-the-art of preconditioners for sparse linear least squares problems: the complete results*, Technical Report RAL-TR-2015-009, Rutherford Appleton Laboratory, 2015.

[29] 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/`.

[30] 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.

[31] K. Hayami, J.-F. Yin, and T. Ito, *GMRES methods for least squares problems*, SIAM J. on Matrix Analysis and Applications, 31 (2010), pp. 2400–2430.

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

[33] 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.

[34] 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.

[35] 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.

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

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

[38] A. Jennings and M. A. Ajiz, *Incomplete methods for solving $A^T Ax = b$*, SIAM J. on Scientific and Statistical Computing, 5 (1984), pp. 978–987.

[39] S. Kaczmarz, *Ängenäherte Auflösung von Systemen linearer Gleichungen*, Bull. Internat. Acad. Polon. Sci. Cl. A., (1937), pp. 355–356.

[40] I. E. Kaporin, *High quality preconditioning of a general symmetric positive definite matrix based on its $U^TU + U^TR + R^TU$ decomposition*, Numerical Linear Algebra with Applications, 5 (1998), pp. 483–509.

[41] 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.

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

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

[44] 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.

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

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

[47] A. R. L. Oliveira and D. C. Sorensen, *A new class of preconditioners for large-scale linear systems from interior point methods for linear programming*, Linear Algebra and its Applications, 394 (2005), pp. 1–24.

[48] 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.

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

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

[51] A. T. Papadopoulus, I. S. Duff, and A. J. Wathen, *A class of incomplete orthogonal factorization methods. II: Implementation and results*, BIT Numerical Mathematics, 45 (2005), pp. 159–179.

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

[53] 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.

[54] 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.

[55] Y. Saad, *Preconditioning techniques for nonsymmetric and indefinite linear systems*, J. of Computational and Applied Mathematics, 24 (1988), pp. 89–105.

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

[57] 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.

[58] M. A. Saunders, *Sparse least squares problems by conjugate gradients: a comparison of preconditioning methods*, in Proceedings of Computer Science and Statistics: Twelfth Annual Conference on the Interface, Waterloo, Canada, 1979.

[59] ——, *LU preconditioning for full rank and singular sparse least squares*, 2015. Presentation at SIAM Conference on Applied Linear Algebra (LA15) available from `https://www.pathlms.com/siam/courses/1697/sections/2326`.

[60] 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.

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

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

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

[64] ——, *Preconditioning of linear least squares by RIF for implicitly held normal equations*, Technical Report RAL-TR-2016-P-00x, Rutherford Appleton Laboratory, 2016. In preparation.

[65] M. Tismenetsky, *A new preconditioning technique for solving large sparse linear systems*, Linear Algebra and its Applications, 154–156 (1991), pp. 331–353.

[66] X. Wang, K. A. Gallivan, and R. Bramley, *CIMGS: an incomplete orthogonal factorization preconditioner*, SIAM J. on Scientific Computing, 18 (1997), pp. 516–536.

Appendix: statistics for our test set

For each problem in the test subset $\mathcal{T}$ described in Section 2.1, m, n and $nz(A)$ are the row and column counts and the number of nonzeros in A. In addition, "nullity" is the estimated deficiency in the rank as computed by `HSL_MA97`, "density(A)" is the largest ratio of number of nonzeros in a row to n over all rows, and "density(C)" is the ratio of the number of entries in C to n^2. A $-$ denotes insufficient memory to compute the statistic.

Table A.1: Statistics for the test set $\mathcal{T}$.

name	m	n	$nz(A)$	nullity	density(A)	density(C)
CUTEst examples						
BAS1LP	9825	5411	587775	0	0.0675	0.0887
BAXTER	30733	27441	111576	2993	0.0017	0.0016
BCDOUT	7078	5412	67344	2	0.1554	0.0686
CO9	22924	10789	109651	0	0.0026	0.0021
CONT11_L	1961394	1468599	5382999	0	0.0000	0.0000
DBIR1	45775	18804	1077025	103	0.0119	0.0119
DBIR2	45877	18906	1158159	101	0.0123	0.0069
D2Q06C	5831	2171	33081	0	0.0157	0.0074
DELF000	5543	3128	13741	0	0.0029	0.0027
GE	16369	10099	44825	0	0.0036	0.0011
LARGE001	7176	4162	18887	0	0.0026	0.0025
LPL1	129959	39951	386218	44	0.0004	0.0003
MOD2	66409	34774	199810	0	0.0005	0.0005
MODEL10	16819	4400	150372	0	0.0039	0.0151
MPSBCD03	7078	5412	66210	2	0.1554	0.0682
NSCT2	37563	23003	697738	287	0.0273	0.0157
NSIR2	10057	4453	154939	0	0.0528	0.0239
PDE1	271792	270595	990587	-	0.6696	-
PDS-100	514577	156016	1096002	227	0.0000	0.0001
PDS-90	475448	142596	1014136	227	0.0000	0.0001
PILOT-JA	2267	940	14977	0	0.0585	0.0336
PILOTNOV	2446	975	13331	0	0.0410	0.0265
RAIL2586	923269	2586	8011362	0	0.0046	0.0705
RAIL4284	1096894	4284	11284032	0	0.0028	0.1187
SPAL_004	321696	10203	46168124	0	0.0165	0.4985
STAT96V2	957432	29089	2852184	0	0.0004	0.0004
STAT96V3	1113780	33841	3317736	0	0.0004	0.0004
STAT96V4	63076	3173	491336	0	0.0028	0.0054
STORMG21K	1377306	526185	3459881	0	0.0019	0.0003
WATSON_1	386992	201155	1055093	0	0.0000	0.0000
WATSON_2	677224	352013	1846391	0	0.0000	0.0000
WORLD	67147	34506	198883	0	0.0005	0.0005
UF Sparse Matrix Collection examples						
12month1	872622	12471	22624727	-	0.2742	0.6869
162bit	3606	3476	37118	16	0.0040	0.0195
176bit	7441	7150	82270	40	0.0022	0.0103
192bit	13691	13093	154303	82	0.0012	0.0057
208bit	24430	23191	299756	199	0.0008	0.0036
beaflw	500	492	53403	4	0.8130	0.8945

Table A.1: Statistics for the test set $\mathcal{T}$ (continued).

name	m	n	$nz(A)$	nullity	density(A)	density(C)
c8_mat11	5761	4562	2462970	0	0.5298	0.8120
connectus	394707	458	1127525	0	0.1594	0.1579
ESOC	327062	37349	6019939	0	0.0005	0.0052
EternityII_Etilde	204304	10054	1170516	0	0.0007	0.0170
f855_mat9	2511	2456	171214	0	0.3375	0.7436
GL7d16	955127	460260	14488881	-	0.0001	0.0009
GL7d17	1548649	955127	25978098	-	0.0001	0.0004
GL7d18	1955309	1548645	35590540	-	0.0000	0.0003
GL7d19	1955296	1911130	37322725	-	0.0000	0.0002
GL7d20	1911124	1437546	29893084	-	0.0000	0.0002
GL7d21	1437546	822922	18174775	-	0.0000	0.0003
GL7d22	822906	349443	8251000	-	0.0001	0.0006
GL7d23	349443	105054	2695430	-	0.0002	0.0017
graphics	29493	11822	117954	0	0.0003	0.0006
HFE18_96_in	2372	2371	933343	0	0.5065	0.9912
IG5-15	11369	6146	323509	0	0.0195	0.1521
IG5-16	18846	9519	588326	0	0.0126	0.1280
IG5-17	30162	14060	1035008	0	0.0085	0.1140
IG5-18	47894	20818	1790490	0	0.0058	0.0991
IMDB	896302	303617	3782463	-	0.0052	0.0015
kneser_10_4_1	349651	330751	992252	-	0.0000	0.0001
landmark	71952	2673	1146848	2	0.0060	0.0168
LargeRegFile	2111154	801374	4944201	0	0.0000	0.0000
Maragal_6	21251	10144	537694	516	0.5857	0.7491
Maragal_7	46845	26525	1200537	2046	0.3604	0.3099
Maragal_8	60845	33093	1308415	7107	0.0503	0.0356
mri1	114637	65536	589824	603	0.0037	0.0003
mri2	104597	63240	569160	-	0.0660	0.0078
NotreDame_actors	383640	127823	1470404	-	0.0051	0.0025
psse0	26722	11028	102432	0	0.0004	0.0006
psse1	14318	11028	57376	0	0.0016	0.0007
psse2	28634	11028	115262	0	0.0025	0.0008
rel9	5921786	274667	23667183	-	0.0000	0.0005
relat9	9746232	274667	38955420	-	0.0000	0.0005
Rucci1	1977885	109900	7791168	0	0.0000	0.0008
sls	1748122	62729	6804304	0	0.0001	0.0012
TF14	3159	2644	29862	0	0.0049	0.0312
TF15	7741	6334	80057	0	0.0022	0.0163
TF16	19320	15437	216173	0	0.0010	0.0082
TF17	48629	38132	586218	-	0.0004	0.0040
TF18	123867	95368	1597545	-	0.0002	0.0019
TF19	317955	241029	4370721	-	0.0001	0.0009
tomographic1	59360	45908	647495	3436	0.0003	0.0009
Trec14	15904	3159	2872265	0	0.7914	0.9317
wheel_601	902103	723605	2170814	-	0.0008	0.0004

Table A.2: Storage required for factors (or for GMRES) for subset CUTEst problems by each method

name	m	n	$nz(A)$	no	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN	MA97
BAS1LP	9825	5411	587775	0	5411	154335	58422	6419411	113414	858406	3078236
BAXTER	30733	27441	111576	0	27441	166672	254733	27441	218513	709648	25904371
BCDOUT	7078	5412	67344	0	5412	227118	38953	5412	97350	214193	2167514
CO9	22924	10789	109651	0	10789	139168	52621	11802789	122520	346055	1188478
CONT11_L	1961394	1468599	5382999	0	1468599	6823917	4883216	149807602	10208898	22208667	147967730
DBIR1	45775	18804	1077025	0	2171	22404	13660	700243	30258	100726	260137
DBIR2	45877	18906	1158159	0	18804	306833	92947	7055662	264644	1977702	117251868
D2Q06C	5831	2171	33081	0	18906	311225	94898	17102778	172316	1876024	141400121
DELF000	5543	3128	13741	0	3128	9959	11667	4134128	11314	33260	136015
GE	16369	10099	44825	0	10099	98350	49164	8719998	134483	248210	730178
LARGE001	7176	4162	18887	0	4162	14243	16930	5169162	16000	46375	195019
LPL1	129959	39951	386218	0	39951	184639	143744	28745911	541699	1605246	12926410
MOD2	66409	34774	199810	0	34774	559518	229986	20505898	527770	1300573	4209242
MODEL10	16819	4400	150372	0	4400	34670	24083	1863308	78015	378047	733403
MPSBCD03	7078	5412	66210	0	5412	228815	38965	5412	98584	220678	2163800
NSCT2	37563	23003	697738	0	23003	561767	109287	4781258	165302	1407946	16602700
NSIR2	10057	4453	154939	0	4453	98551	27196	984313	54142	305406	704032
PDE1	271792	270595	990587	0	270595	-	>993046	270595	-	-	-
PDS-100	514577	156016	1096002	0	156016	1027112	618440	12019236	2891511	7696832	189690066
PDS-90	475448	142596	1014136	0	142596	950792	570713	10557652	2626289	7233396	180340804
PILOT-JA	2267	940	14977	0	940	9650	5780	427458	10870	47073	105443
PILOTNOV	2446	975	13331	0	975	9320	5958	449095	11317	44818	103809
RAIL2586	923269	2586	8011362	0	2586	6235	24447	495112	51833	9613272	1531996
RAIL4284	1096894	4284	11284032	0	4284	6345	45055	958072	89219	15880038	6895656
SPAL_004	321696	10203	46168124	0	10203	18683	>7018	20410	213972	40754070	47074959
STAT96V2	957432	29089	2852184	0	29089	38807	81055	12573814	276290	4210897	1741109
STAT96V3	1113780	33841	3317736	0	33841	45145	94273	14216653	319109	4901099	2033605
STAT96V4	63076	3173	491336	0	3173	11493	10513	415798	47295	512268	155889
STORMG21K	1377306	526185	3459881	0	526185	6933837	>1957476	526185	7505682	34433755	>406183656
WATSON_1	386992	201155	1055093	0	201155	1005650	519977	11469139	2634818	6106788	8455310
WATSON_2	677224	352013	1846391	0	352013	2808070	1171936	17251085	4648375	10588235	14968100
WORLD	67147	34506	198883	0	34506	535778	224464	20065186	511951	1288095	4015154

Table A.2: Storage required for factors (or for GMRES) for subset UF problems by each method

name	m	n	$nz(A)$	no	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN	MA97
12month1	872622	12471	22624727	0	12471	73375	>65361	764511	260889	34096963	>13133836
162bit	3606	3476	37118	0	3476	193542	28550	1215038	70512	166083	2378948
176bit	7441	7150	82270	0	7150	388870	56269	5121390	145649	349258	8383997
192bit	13691	13093	154303	0	13093	685467	96275	14109093	267084	654954	46204979
208bit	24430	23191	299756	0	23191	1202798	168273	24217191	471420	1177025	132705125
beaflw	500	492	53403	0	492	28073	5307	475792	9814	71656	367705
c8_mat11	5761	4562	2462970	0	4562	78255	49640	4562	95477	2621770	9692833
connectus	394707	458	1127525	0	458	478	1744	1850	5185	1527818	32052
ESOC	327062	37349	6019939	0	37349	698366	>271701	37349	776570	12993333	47830600
EternityII_Etilde	204304	10054	1170516	0	10054	24368	103380	4019398	205375	4202504	5383429
f855_mat9	2511	2456	171214	0	2456	136743	26961	2456	51366	264054	7250113
GL7d16	955127	460260	14488881	0	460260	9346036	>342783	4602708	9665201	36508148	>1327570176
GL7d17	1548649	955127	25978098	0	955127	31102238	>342999	8596231	20057165	-	>211108386
GL7d18	1955309	1548645	35590540	0	1548645	>82048471	>375925	17035225	>32520973	-	-
GL7d19	1955296	1911130	37322725	0	1911130	>178396100	>572915	19111408	40133312	-	-
GL7d20	1911124	1437546	29893084	0	1437546	>118385247	>544782	8625316	30188087	-	>2214333761
GL7d21	1437546	822922	18174775	0	822922	49569906	>537760	5760508	17281075	-	>291709185
GL7d22	822906	349443	8251000	0	349443	12616485	>552082	2446155	7337962	30115416	318893119
GL7d23	349443	105054	2695430	0	105054	1721600	>569276	735432	2205827	10424076	>557267849
graphics	29493	11822	117954	0	11822	31960	30470	11822	24901	194127	450522
HFE18_96_in	2372	2371	933343	0	2371	10978	26016	3376371	49576	1028976	2810872
IG5-15	11369	6146	323509	0	6146	189538	62626	580318	128833	643103	13602718
IG5-16	18846	9519	588326	0	9519	279931	97466	1470142	199678	1115361	31140231
IG5-17	30162	14060	1035008	0	14060	409603	145262	1758938	295028	1843157	69825388
IG5-18	47894	20818	1790490	0	20818	587680	216979	2260114	436954	2992054	153672995
IMDB	896302	303617	3782463	0	303617	15909318	>521169	303617	5858164	13438747	>7625707183
kneser_10_4_1	349651	330751	992252	0	330751	6394519	-	330751	6759341	11405272	>176051300
landmark	71952	2673	1146848	0	2673	11307	17778	75654	26909	873043	368699
LargeRegFile	2111154	801374	4944201	0	801374	3615761	>462274	6411062	4106048	24923112	14314381
Maragal_6	21251	10144	537694	0	10144	246737	71907	3727498	212144	679769	22561128
Maragal_7	46845	26525	1200537	0	26525	662167	166881	1835053	553856	1641051	68305889
Maragal_8	60845	33093	1308415	0	33093	1702715	229119	33093	597971	1392093	163207703
mri1	114637	65536	589824	0	65536	519051	325965	62016508	636290	1776829	17833859
mri2	104597	63240	569160	0	63240	1486341	392461	47669568	781491	2575072	>8316379
NotreDame_actors	383640	127823	1470404	0	127823	6768455	>708013	127823	2506265	6151175	>428559270
psse0	26722	11028	102432	0	11028	31603	23833	12042028	35197	180414	338043
psse1	14318	11028	57376	0	11028	40288	28925	12042028	35771	133438	345004
psse2	28634	11028	115262	0	11028	38437	34657	12042028	40814	216020	359698
rel9	5921786	274667	23667183	0	274667	498966	>320081	3570851	5764774	-	>40800758
relat9	9746232	274667	38955420	0	274667	337475	>272798	3845546	5763434	-	>53209865
Rucci1	1977885	109900	7791168	0	109900	638743	932200	109900	2306811	19056222	197328655
sls	1748122	62729	6804304	0	62729	71400	>108831	3013342	1226997	13636543	52048692
TF14	3159	2644	29862	0	2644	150764	28727	3649644	55249	148306	2427532
TF15	7741	6334	80057	0	6334	379663	69250	6334	132747	368065	14097226
TF16	19320	15437	216173	0	15437	933646	169320	15437	323886	928743	83788762
TF17	48629	38132	586218	0	38132	2304372	418887	38132	800458	2359904	>537526176
TF18	123867	95368	1597545	0	95368	5699962	>829968	95368	2002386	6080905	>1273959539
TF19	317955	241029	4370721	0	241029	14260905	>817427	241029	5061262	15718547	>2785976433
tomographic1	59360	45908	647495	0	45908	988212	280726	45908	906437	2247216	127261944
Trec14	15904	3159	2872265	0	3159	11942	34692	2661039	66099	3251107	14324608
wheel_601	902103	723605	2170814	0	723605	7796970	4253762	723605	14201674	25831230	>21906082

Table A.3: Iterations required for subset CUTEst problems by each method

name	m	n	$nz(A)$	no	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN
BAS1LP	9825	5411	587775	14870	7610	48420	28	4406	7983	11477
BAXTER	30733	27441	111576	>100000	>100000	103	>100000	-	>100000	>100000
BCDOUT	7078	5412	67344	>100000	>100000	>100000	>100000	-	>100000	>100000
CO9	22924	10789	109651	24925	5063	3611	7006	2619	381	194
CONT11_L	1961394	1468599	5382999	206	206	>2	60	101	22	19
DBIR1	45775	18804	1077025	58936	1597	471	12283	284	209	18
DBIR2	45877	18906	1158159	1451	2229	30673	>100000	367	1863	62307
D2Q06C	5831	2171	33081	19090	2208	33838	>100000	864	833	>84581
DELF000	5543	3128	13741	>100000	26469	693	30637	3616	60	58
GE	16369	10099	44825	69445	6249	572	830	799	28	81
LARGE001	7176	4162	18887	52505	26782	53686	>100000	6589	75	90
LPL1	129959	39951	386218	30201	3175	563	>100000	706	420	66
MOD2	66409	34774	199810	10664	1370	1515	46350	579	151	88
MODEL10	16819	4400	150372	34369	2229	5870	>100000	388	743	202
MPSBCD03	7078	5412	66210	>100000	>100000	>100000	>100000	-	>100000	>100000
NSCT2	37563	23003	697738	9991	1395	13525	>100000	205	615	>100000
NSIR2	10057	4453	154939	9611	1037	20205	9	210	386	72961
PDE1	271792	270595	990587	906	965	-	-	-	-	-
PDS-100	514577	156016	1096002	681	342	228	203	76	90	64
PDS-90	475448	142596	1014136	639	331	216	195	73	88	76
PILOT-JA	2267	940	14977	>100000	2344	61	28197	334	323	54
PILOTNOV	2446	975	13331	83448	1927	43	17920	340	214	20
RAIL2586	923269	2586	8011362	919	401	816	233	178	151	9
RAIL4284	1096894	4284	11284032	887	733	923	375	212	224	19
SPAL_004	321696	10203	46168124	>2572	>2507	>2740	-	1	>1047	535
STAT96V2	957432	29089	2852184	986	726	462	414	425	19	22
STAT96V3	1113780	33841	3317736	1055	765	485	435	414	20	27
STAT96V4	63076	3173	491336	4144	810	1767	449	125	17	24
STORMG21K	1377306	526185	3459881	1383	183	>6446	-	-	2285	>4234
WATSON_1	386992	201155	1055093	2160	422	165	249	56	73	8
WATSON_2	677224	352013	1846391	1812	349	119	185	48	54	7
WORLD	67147	34506	198883	9811	1369	1084	24796	571	154	70

Table A.3: Iterations required for subset UF problems by each method

name	m	n	$nz(A)$	no	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN
12month1	872622	12471	22624727	>6310	268	972	-	60	371	294
162bit	3606	3476	37118	29396	2540	728	1495	319	247	252
176bit	7441	7150	82270	>100000	6537	2710	3894	655	454	981
192bit	13691	13093	154303	>100000	12203	4790	7236	6005	1282	1521
208bit	24430	23191	299756	>100000	17073	9764	13993	6219	2198	3131
beaflw	500	492	53403	43875	40985	38436	>5	485	33967	>100000
c8_mat11	5761	4562	2462970	40787	38344	>100000	>100000	-	29944	>76276
connectus	394707	458	1127525	1748	7	114	7	3	6	3
ESOC	327062	37349	6019939	5604	15004	>20935	-	-	>21629	>14678
EternityII_Etilde	204304	10054	1170516	1354	1122	2098	883	384	585	81
f855_mat9	2511	2456	171214	19017	20354	>100000	>100000	-	12372	>100000
GL7d16	955127	460260	14488881	61	48	264	-	9	32	35
GL7d17	1548649	955127	25978098	58	48	422	-	8	28	-
GL7d18	1955309	1548645	35590540	80	64	-	-	10	-	-
GL7d19	1955296	1911130	37322725	205	53	-	-	9	46	-
GL7d20	1911124	1437546	29893084	136	31	-	-	5	28	-
GL7d21	1437546	822922	18174775	143	26	196	-	6	25	-
GL7d22	822906	349443	8251000	238	24	124	-	6	22	30
GL7d23	349443	105054	2695430	340	24	91	-	6	21	22
graphics	29493	11822	117954	>100000	>100000	>100000	>100000	-	1891	224
HFE18_96_in	2372	2371	933343	30697	15102	30130	12433	1632	14659	16095
IG5-15	11369	6146	323509	4577	608	126	321	92	239	535
IG5-16	18846	9519	588326	7406	863	150	478	151	348	772
IG5-17	30162	14060	1035008	7264	828	169	410	123	326	706
IG5-18	47894	20818	1790490	7282	738	205	447	107	309	886
IMDB	896302	303617	3782463	>12955	>10670	>3303	-	-	>5974	>4263
kneser_10_4_1	349651	330751	992252	17209	10781	>8403	>1	-	3257	1867
landmark	71952	2673	1146848	19937	894	36	274	27	12	25
LargeRegFile	2111154	801374	4944201	795	54	167	-	7	12	21
Maragal_6	21251	10144	537694	5178	1942	8699	>100000	354	679	1291
Maragal_7	46845	26525	1200537	2769	1071	1444	5718	68	264	477
Maragal_8	60845	33093	1308415	>100000	>100000	>43720	>100000	-	>71973	65802
mri1	114637	65536	589824	6108	6116	8810	1575	932	2217	65
mri2	104597	63240	569160	11852	11822	4315	>1	744	2935	10271
NotreDame_actors	383640	127823	1470404	>52691	>49880	>9971	-	-	>18396	>12190
psse0	26722	11028	102432	82324	43122	1210	22445	28003	71	30
psse1	14318	11028	57376	64151	50610	4588	>100000	37887	575	136
psse2	28634	11028	115262	81831	58572	7677	>100000	43868	722	174
rel9	5921786	274667	23667183	110	81	107	-	12	37	-
relat9	9746232	274667	38955420	88	76	82	-	13	36	-
Rucci1	1977885	109900	7791168	17837	8330	>15623	1823	-	>12455	553
sls	1748122	62729	6804304	638	189	620	-	47	68	13
TF14	3159	2644	29862	34727	25709	44760	12914	1758	11418	1087
TF15	7741	6334	80057	>100000	81922	>100000	43460	-	41034	1744
TF16	19320	15437	216173	>100000	>100000	>100000	>100000	-	>100000	1107
TF17	48629	38132	586218	>100000	>100000	>43947	>100000	-	>84020	1199
TF18	123867	95368	1597545	>100000	>93527	>16944	-	-	>28500	1463
TF19	317955	241029	4370721	>32164	>30368	>5798	-	-	>8670	1087
tomographic1	59360	45908	647495	65455	18905	>72373	12	-	1867	1840
Trec14	15904	3159	2872265	2007	1593	8533	>1	690	1603	16005
wheel_601	902103	723605	2170814	>19475	>18285	>5102	>7367	-	>4024	>4585

Table A.4: Time required for factors for subset CUTEst problems by each method

name	m	n	$nz(A)$	no	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN	MA97
BAS1LP	9825	5411	587775	0.00	0.00	0.57	7.25	0.00	1.33	1.03	0.19
BAXTER	30733	27441	111576	0.00	0.00	0.30	1.32	0.00	0.29	0.80	0.10
BCDOUT	7078	5412	67344	0.00	0.00	0.40	1.51	0.00	0.46	0.27	0.04
C09	22924	10789	109651	0.00	0.00	0.16	1.04	0.00	0.22	0.77	0.10
CONT11_L	1961394	1468599	5382999	0.00	0.01	16.14	45.56	0.00	9.38	18.36	5.26
DBIR1	45775	18804	1077025	0.00	0.00	0.04	0.07	0.00	0.04	0.14	0.02
DBIR2	45877	18906	1158159	0.00	0.00	1.32	40.98	0.00	2.24	4.57	0.39
D2Q06C	5831	2171	33081	0.00	0.00	0.83	47.59	0.00	2.14	4.18	0.41
DELF000	5543	3128	13741	0.00	0.00	0.01	0.01	0.00	0.01	0.01	0.02
GE	16369	10099	44825	0.00	0.00	0.07	0.17	0.00	0.08	0.37	0.05
LARGE001	7176	4162	18887	0.00	0.00	0.02	0.01	0.00	0.01	0.02	0.02
LPL1	129959	39951	386218	0.00	0.00	0.28	0.44	0.00	0.94	3.25	0.31
MOD2	66409	34774	199810	0.00	0.00	0.44	2.15	0.00	1.07	1.21	0.17
MODEL10	16819	4400	150372	0.00	0.00	0.20	0.39	0.00	0.09	0.48	0.08
MPSBCD03	7078	5412	66210	0.00	0.00	0.39	1.57	0.00	0.42	0.33	0.05
NSCT2	37563	23003	697738	0.00	0.00	1.46	37.53	0.00	2.51	3.88	0.27
NSIR2	10057	4453	154939	0.00	0.00	0.21	2.41	0.00	0.21	0.17	0.06
PDE1	271792	270595	990587	0.00	0.00	>600.00	>600.00	0.00	0.01	>600.00	>1.00
PDS-100	514577	156016	1096002	0.00	0.00	1.21	22.34	0.00	1.74	14.03	1.12
PDS-90	475448	142596	1014136	0.00	0.00	1.09	21.45	0.00	1.44	12.86	1.04
PILOT-JA	2267	940	14977	0.00	0.00	0.02	0.04	0.00	0.03	0.02	0.01
PILOTNOV	2446	975	13331	0.00	0.00	0.02	0.03	0.00	0.02	0.02	0.01
RAIL2586	923269	2586	8011362	0.00	0.01	2.37	110.43	0.00	2.43	4.31	3.32
RAIL4284	1096894	4284	11284032	0.00	0.01	4.82	338.76	0.00	4.65	13.08	4.52
SPAL_004	321696	10203	46168124	0.00	0.04	9.67	>600.00	0.00	74.72	42.00	17.98
STAT96V2	957432	29089	2852184	0.00	0.00	0.15	0.60	0.00	0.27	1.29	2.50
STAT96V3	1113780	33841	3317736	0.00	0.00	0.17	0.77	0.00	0.31	1.28	2.95
STAT96V4	63076	3173	491336	0.00	0.00	0.04	2.16	0.00	0.08	0.21	0.24
STORMG21K	1377306	526185	3459881	0.00	0.01	66.80	>600.00	0.00	30.83	56.90	>48.23
WATSON_1	386992	201155	1055093	0.00	0.00	1.69	0.49	0.00	1.50	0.99	1.01
WATSON_2	677224	352013	1846391	0.00	0.00	5.04	4.11	0.00	2.85	2.15	1.77
WORLD	67147	34506	198883	0.00	0.00	0.41	2.19	0.00	0.76	3.42	0.17

Table A.4: Time required for factors for subset UF problems by each method

name	m	n	$nz(A)$	no	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN	MA97
12month1	872622	12471	22624727	0.00	0.06	23.18	>600.00	0.00	171.69	373.85	-165.18
162bit	3606	3476	37118	0.00	0.00	0.47	1.76	0.00	0.15	0.15	0.18
176bit	7441	7150	82270	0.00	0.00	1.23	8.31	0.00	0.28	0.45	0.80
192bit	13691	13093	154303	0.00	0.00	2.55	30.08	0.00	0.47	0.69	9.90
208bit	24430	23191	299756	0.00	0.00	5.52	109.21	0.00	0.97	1.79	47.80
beaflw	500	492	53403	0.00	0.00	0.08	0.19	0.00	0.11	0.12	0.12
c8_mat11	5761	4562	2462970	0.00	0.00	7.82	63.80	0.00	31.86	7.41	24.13
connectus	394707	458	1127525	0.00	0.00	0.12	2.23	0.00	0.24	0.58	0.17
ESOC	327062	37349	6019939	0.00	0.01	6.75	>600.00	0.00	3.32	316.35	9.03
EternityII_Etilde	204304	10054	1170516	0.00	0.00	0.67	37.16	0.00	1.14	2.17	0.44
f855_mat9	2511	2456	171214	0.00	0.00	0.62	2.89	0.00	0.89	0.51	1.87
GL7d16	955127	460260	14488881	0.00	0.01	397.42	>600.00	0.00	50.70	220.47	>60.27
GL7d17	1548649	955127	25978098	0.00	0.03	1148.65	>600.00	0.00	301.96	>600.00	>185.83
GL7d18	1955309	1548645	35590540	0.00	0.04	>600.00	>600.00	0.00	>600.00	>600.00	>417.03
GL7d19	1955296	1911130	37322725	0.00	0.04	>600.00	>600.00	0.00	438.21	>600.00	>71.78
GL7d20	1911124	1437546	29893084	0.00	0.03	>600.00	>600.00	0.00	219.99	>600.00	>317.00
GL7d21	1437546	822922	18174775	0.00	0.02	657.94	>600.00	0.00	73.73	>600.00	>110.01
GL7d22	822906	349443	8251000	0.00	0.01	165.69	>600.00	0.00	21.64	192.74	>38.17
GL7d23	349443	105054	2695430	0.00	0.00	24.26	>600.00	0.00	5.03	94.06	>11.29
graphics	29493	11822	117954	0.00	0.00	0.08	0.16	0.00	0.04	0.08	0.05
HFE18_96_in	2372	2371	933343	0.00	0.00	1.08	14.28	0.00	6.11	0.70	3.38
IG5-15	11369	6146	323509	0.00	0.00	0.96	9.73	0.00	0.71	1.81	2.15
IG5-16	18846	9519	588326	0.00	0.00	1.85	26.86	0.00	1.68	1.48	5.53
IG5-17	30162	14060	1035008	0.00	0.00	3.28	70.05	0.00	2.76	7.00	18.09
IG5-18	47894	20818	1790490	0.00	0.00	5.87	179.05	0.00	6.76	10.04	58.55
IMDB	896302	303617	3782463	0.00	0.01	51.71	>600.00	0.00	40.04	124.49	>25.71
kneser_10_4_1	349651	330751	992252	0.00	0.00	9.50	0.00	0.00	5.69	14.40	>97.12
landmark	71952	2673	1146848	0.00	0.00	0.27	1.67	0.00	0.29	0.48	0.18
LargeRegFile	2111154	801374	4944201	0.00	0.01	11.14	>600.00	0.00	2.21	78.63	1.05
Maragal_6	21251	10144	537694	0.00	0.00	2.28	40.18	0.00	13.56	2.10	24.36
Maragal_7	46845	26525	1200537	0.00	0.00	8.42	144.30	0.00	47.14	8.13	146.14
Maragal_8	60845	33093	1308415	0.00	0.00	16.12	210.23	0.00	5.27	2.96	88.43
mri1	114637	65536	589824	0.00	0.00	1.26	93.78	0.00	2.24	1.95	1.54
mri2	104597	63240	569160	0.00	0.00	4.09	67.78	0.00	4.95	6.26	>59.24
NotreDame_actors	383640	127823	1470404	0.00	0.00	11.41	>600.00	0.00	9.20	28.88	>20.41
psse0	26722	11028	102432	0.00	0.00	0.07	0.04	0.00	0.02	0.07	0.03
psse1	14318	11028	57376	0.00	0.00	0.10	0.05	0.00	0.03	0.04	0.03
psse2	28634	11028	115262	0.00	0.00	0.08	0.13	0.00	0.04	0.06	0.03
rel9	5921786	274667	23667183	0.00	0.02	16.10	>600.00	0.00	44.03	>600.00	>29.86
relat9	9746232	274667	38955420	0.00	0.03	14.26	>600.00	0.00	53.35	>600.00	>34.84
Rucci1	1977885	109900	7791168	0.00	0.01	2.37	172.24	0.00	2.49	40.80	21.88
sls	1748122	62729	6804304	0.00	0.01	1.56	>600.00	0.00	6.44	15.05	12.23
TF14	3159	2644	29862	0.00	0.00	0.24	0.43	0.00	0.06	0.21	0.18
TF15	7741	6334	80057	0.00	0.00	0.75	2.61	0.00	0.21	0.43	1.43
TF16	19320	15437	216173	0.00	0.00	2.38	17.44	0.00	0.45	1.05	16.18
TF17	48629	38132	586218	0.00	0.00	7.59	119.28	0.00	1.22	3.77	>6.82
TF18	123867	95368	1597545	0.00	0.00	23.83	>600.00	0.00	3.58	12.20	>7.62
TF19	317955	241029	4370721	0.00	0.00	82.22	>600.00	0.00	12.75	37.71	>31.67
tomographic1	59360	45908	647495	0.00	0.00	2.86	3.00	0.00	1.48	1.49	33.93
Trec14	15904	3159	2872265	0.00	0.00	2.35	57.38	0.00	27.41	10.68	20.31
wheel_601	902103	723605	2170814	0.00	0.01	20.39	36.75	0.00	51.45	46.90	>42.90

Table A.5: Total time required for subset CUTEst problems by each method

name	m	n	$nz(A)$	no	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN	MA97
BAS1LP	9825	5411	587775	27.18	14.10	133.70	7.31	119.59	21.71	32.51	0.93
BAXTER	30733	27441	111576	>67.29	>86.64	0.52	>79.54	>600.00	>293.46	>305.83	3.61
BCDOUT	7078	5412	67344	>28.51	>32.47	>137.51	>50.06	>600.00	>92.58	>66.19	0.36
CO9	22924	10789	109651	12.93	2.98	5.20	7.03	37.17	0.79	1.06	0.10
CONT11_L	1961394	1468599	5382999	8.47	10.76	>16.44	52.40	32.92	13.68	22.88	8.61
DBIR1	45775	18804	1077025	8.87	0.27	0.18	2.97	0.42	0.11	0.15	0.02
DBIR2	45877	18906	1158159	5.44	8.79	185.29	>518.59	7.21	13.04	455.44	72.80
D2Q06C	5831	2171	33081	82.11	9.45	213.36	>288.71	24.89	6.80	>600.00	46.14
DELF000	5543	3128	13741	>7.77	2.65	0.15	4.99	12.78	0.03	0.02	0.02
GE	16369	10099	44825	20.64	2.29	0.64	0.71	7.22	0.12	0.46	0.06
LARGE001	7176	4162	18887	5.55	3.69	14.39	>22.47	33.58	0.04	0.05	0.02
LPL1	129959	39951	386218	65.54	7.88	2.62	>343.06	26.24	3.99	3.83	1.27
MOD2	66409	34774	199810	11.92	1.83	7.56	117.56	13.36	1.93	1.72	0.32
MODEL10	16819	4400	150372	18.64	1.28	5.02	>70.89	1.43	0.88	0.73	0.08
MPSBCD03	7078	5412	66210	>28.30	>32.57	>137.96	>49.69	>600.00	>92.08	>66.27	0.37
NSCT2	37563	23003	697738	25.19	3.80	81.95	>200.45	2.74	5.19	>526.77	4.57
NSIR2	10057	4453	154939	5.12	0.59	21.23	2.41	0.56	0.58	65.32	0.13
PDE1	271792	270595	990587	6.27	8.38	-	-	>600.00	-	-	-
PDS-100	514577	156016	1096002	6.08	5.24	5.18	25.07	3.18	5.19	17.16	60.36
PDS-90	475448	142596	1014136	5.18	4.60	4.57	23.91	2.78	3.93	16.12	56.21
PILOT-JA	2267	940	14977	>5.76	0.18	0.03	2.57	0.16	0.08	0.03	0.01
PILOTNOV	2446	975	13331	4.47	0.12	0.03	1.60	0.14	0.03	0.02	0.02
RAIL2586	923269	2586	8011362	40.50	17.64	39.62	121.17	41.97	10.55	4.98	2.19
RAIL4284	1096894	4284	11284032	70.04	55.71	77.84	368.38	77.62	25.03	14.91	4.00
SPAL_004	321696	10203	46168124	>600.00	>600.00	>600.00	-	3.87	>600.00	125.69	124.87
STAT96V2	957432	29089	2852184	11.18	8.21	5.56	5.79	13.11	0.54	2.19	0.40
STAT96V3	1113780	33841	3317736	13.93	10.46	6.84	7.25	15.55	0.64	2.54	0.41
STAT96V4	63076	3173	491336	6.63	1.35	3.15	2.94	1.41	0.11	0.28	0.07
STORMG21K	1377306	526185	3459881	30.85	5.14	>600.00	-	>600.00	258.65	>600.00	-
WATSON_1	386992	201155	1055093	14.67	3.39	4.27	3.48	4.89	3.58	1.25	0.67
WATSON_2	677224	352013	1846391	22.73	5.25	9.26	8.48	5.18	5.73	2.60	1.26
WORLD	67147	34506	198883	10.53	1.82	5.24	64.30	12.43	1.58	3.83	0.32

Table A.5: Total time required for subset UF problems by each method

name	m	n	$nz(A)$	no	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN	MA97
12month1	872622	12471	22624727	>600.00	26.26	122.52	-	27.96	207.50	417.74	-
162bit	3606	3476	37118	3.76	0.39	1.21	2.17	0.81	0.31	0.27	0.19
176bit	7441	7150	82270	40.30	2.14	7.60	10.50	6.30	0.85	1.43	0.82
192bit	13691	13093	154303	173.37	7.66	23.69	37.80	132.01	3.34	4.15	10.00
208bit	24430	23191	299756	>600.00	21.08	84.25	139.45	213.97	10.38	18.28	48.07
beaflw	500	492	53403	4.07	4.04	8.67	>0.19	0.51	5.44	>17.42	0.12
c8_mat11	5761	4562	2462970	175.75	171.65	>527.77	>535.49	>600.00	183.17	>600.00	24.20
connectus	394707	458	1127525	7.56	0.03	0.62	2.27	0.13	0.28	0.65	0.22
ESOC	327062	37349	6019939	157.20	332.98	>600.00	-	>600.00	>600.00	>600.00	9.38
EternityII_Etilde	204304	10054	1170516	6.67	5.69	12.34	42.12	9.30	5.02	3.49	0.52
f855_mat9	2511	2456	171214	6.03	6.81	>95.00	>45.52	>600.00	9.55	>66.90	1.89
GL7d16	955127	460260	14488881	13.99	13.44	478.30	-	6.75	58.59	235.09	-
GL7d17	1548649	955127	25978098	23.38	24.26	>600.00	-	14.72	318.66	-	-
GL7d18	1955309	1548645	35590540	57.09	39.84	-	-	25.33	-	-	-
GL7d19	1955296	1911130	37322725	100.66	28.33	-	-	34.12	486.49	-	-
GL7d20	1911124	1437546	29893084	56.31	16.13	-	-	20.63	242.04	-	-
GL7d21	1437546	822922	18174775	33.19	7.33	>600.00	-	9.27	84.13	-	-
GL7d22	822906	349443	8251000	20.68	2.35	190.26	-	3.33	25.17	202.06	-
GL7d23	349443	105054	2695430	6.90	0.53	27.75	-	0.92	5.91	95.99	-
graphics	29493	11822	117954	>600.00	>47.86	>82.32	>68.95	>600.00	1.68	0.35	0.06
HFE18_96_in	2372	2371	933343	48.69	24.58	55.29	36.58	61.33	35.13	47.90	3.41
IG5-15	11369	6146	323509	3.08	0.43	1.16	10.03	1.39	1.06	2.90	2.19
IG5-16	18846	9519	588326	9.53	1.22	2.29	27.69	2.75	2.58	4.48	5.60
IG5-17	30162	14060	1035008	18.04	2.39	4.12	71.35	4.09	4.24	11.96	18.24
IG5-18	47894	20818	1790490	40.50	4.55	7.75	182.09	10.44	9.48	21.55	58.91
IMDB	896302	303617	3782463	>600.00	>600.00	>600.00	-	>600.00	>600.00	>600.00	-
kneser_10_4_1	349651	330751	992252	186.42	142.54	>600.00	>0.10	>600.00	203.24	120.88	-
landmark	71952	2673	1146848	43.35	1.96	0.36	2.73	1.94	0.32	0.60	0.21
LargeRegFile	2111154	801374	4944201	29.43	2.44	24.47	-	3.03	3.51	86.25	1.63
Maragal_6	21251	10144	537694	5.87	2.32	24.61	>199.43	4.19	15.27	5.44	24.41
Maragal_7	46845	26525	1200537	8.76	3.47	18.94	169.55	2.87	48.97	11.43	146.29
Maragal_8	60845	33093	1308415	>600.00	>422.80	>600.00	>600.00	>600.00	>600.00	504.85	88.78
mri1	114637	65536	589824	16.60	18.17	63.25	101.51	83.92	23.14	2.62	1.63
mri2	104597	63240	569160	24.04	30.06	51.52	>67.78	41.31	29.66	118.90	-
NotreDame_actors	383640	127823	1470404	>600.00	>600.00	>600.00	-	>600.00	>600.00	>600.00	-
psse0	26722	11028	102432	31.74	20.05	1.09	16.34	428.84	0.08	0.10	0.04
psse1	14318	11028	57376	18.95	18.91	3.77	>65.68	479.57	0.51	0.14	0.04
psse2	28634	11028	115262	34.75	29.48	7.10	>81.34	554.11	0.76	0.28	0.04
rel9	5921786	274667	23667183	34.16	26.57	47.50	-	24.30	56.86	-	-
relat9	9746232	274667	38955420	48.46	44.81	55.63	-	39.89	73.79	-	-
Rucci1	1977885	109900	7791168	596.42	321.85	>600.00	243.72	>600.00	>600.00	109.48	22.45
sls	1748122	62729	6804304	33.75	9.84	27.46	-	9.42	9.92	17.18	12.72
TF14	3159	2644	29862	3.42	3.03	35.01	3.12	6.81	5.55	0.61	0.19
TF15	7741	6334	80057	26.85	24.25	>198.99	24.76	>600.00	47.82	2.06	1.46
TF16	19320	15437	216173	209.74	>75.68	>545.35	>149.74	>600.00	392.60	4.33	16.34
TF17	48629	38132	586218	>600.00	>212.92	>600.00	>493.09	>600.00	>600.00	14.09	-
TF18	123867	95368	1597545	>600.00	>600.00	>600.00	-	>600.00	>600.00	57.35	-
TF19	317955	241029	4370721	>600.00	>600.00	>600.00	-	>600.00	>600.00	144.15	-
tomographic1	59360	45908	647495	145.39	48.55	>600.00	3.05	>600.00	17.23	17.52	34.19
Trec14	15904	3159	2872265	14.54	9.78	54.43	>57.39	53.76	39.68	170.66	20.37
wheel_601	902103	723605	2170814	>600.00	>600.00	>600.00	>600.00	>600.00	>600.00	>600.00	-

Table A.6: Residuals obtained for subset CUTEst problems by each method

name	no	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN	MA97	MA97-aug	SPQR
BAS1LP	5.54E+1	5.54E+1	5.54E+1	8.02E+1	5.54E+1	5.54E+1	5.54E+1	5.54E+1	5.54E+1	5.54E+1
BAXTER	-	-	8.61E+1	-	-	-	6.61E+1	5.92E+1	5.92E+1	7.39E+1
BCDOUT	-	-	-	-	-	-	3.54E+1	3.53E+1	3.53E+1	3.53E+1
CO9	8.91E+1	8.91E+1	8.91E+1	8.91E+1	8.91E+1	8.91E+1	8.92E+1	8.91E+1	8.91E+1	8.91E+1
CONT11_L	8.09E+2	8.09E+2	-	8.09E+2	8.09E+2	8.09E+2	8.09E+2	8.09E+2	8.09E+2	8.09E+2
DBIR1	3.49E+1	3.49E+1	3.49E+1	3.49E+1	3.49E+1	3.49E+1	3.49E+1	3.49E+1	3.49E+1	3.49E+1
DBIR2	1.67E+2	1.66E+2	1.66E+2	-	1.66E+2	1.67E+2	1.67E+2	1.66E+2	1.66E+2	1.66E+2
D2Q06C	1.67E+2	1.66E+2	1.66E+2	-	1.66E+2	1.67E+2	1.67E+2	1.66E+2	1.66E+2	1.66E+2
DELF000	-	5.38E+1	5.38E+1	5.38E+1	5.38E+1	5.38E+1	5.38E+1	5.38E+1	5.38E+1	5.38E+1
GE	7.25E+1	7.24E+1	7.24E+1	7.24E+1	7.24E+1	7.24E+1	7.24E+1	7.24E+1	7.24E+1	7.24E+1
LARGE001	6.06E+1	6.06E+1	6.06E+1	-	6.06E+1	6.06E+1	6.07E+1	6.06E+1	6.06E+1	6.06E+1
LPL1	7.08E+1	7.08E+1	7.08E+1	-	7.08E+1	7.08E+1	7.09E+1	7.08E+1	7.08E+1	7.08E+1
MOD2	1.38E+2	1.38E+2	1.38E+2	1.38E+2	1.38E+2	1.38E+2	1.39E+2	1.38E+2	1.38E+2	1.38E+2
MODEL10	5.35E+1	5.35E+1	5.35E+1	-	5.35E+1	5.35E+1	5.36E+1	5.35E+1	5.35E+1	5.35E+1
MPSBCD03	-	-	-	-	-	-	3.54E+1	3.52E+1	3.52E+1	3.52E+1
NSCT2	1.83E+2	1.83E+2	1.83E+2	-	1.83E+2	1.83E+2	1.84E+2	1.83E+2	1.83E+2	1.83E+2
NSIR2	8.05E+1	8.04E+1	8.04E+1	9.93E+1	8.04E+1	8.05E+1	8.05E+1	8.04E+1	8.04E+1	8.04E+1
PDE1	3.03E+2	3.03E+2	-	-	-	-	-	-	3.03E+2	-
PDS-100	2.84E+2	2.84E+2	2.84E+2	2.84E+2	2.84E+2	2.84E+2	2.85E+2	2.84E+2	2.84E+2	2.84E+2
PDS-90	2.68E+2	2.68E+2	2.68E+2	2.68E+2	2.68E+2	2.68E+2	2.68E+2	2.68E+2	2.68E+2	2.68E+2
PILOT-JA	-	3.19E+1	3.19E+1	3.19E+1	3.19E+1	3.19E+1	3.20E+1	3.19E+1	3.19E+1	3.19E+1
PILOTNOV	3.50E+1	3.28E+1	3.28E+1	3.28E+1	3.28E+1	3.28E+1	3.29E+1	3.28E+1	3.28E+1	3.28E+1
RAIL2586	1.41E+2	1.41E+2	1.41E+2	1.41E+2	1.41E+2	1.41E+2	1.41E+2	1.41E+2	1.41E+2	1.41E+2
RAIL4284	1.69E+2	1.69E+2	1.69E+2	1.69E+2	1.69E+2	1.69E+2	1.69E+2	1.69E+2	1.69E+2	-
SPAL_004	-	-	-	-	3.27E-11	-	1.47E+0	3.51E-14	-	-
STAT96V2	9.72E+2	9.72E+2	9.72E+2	9.72E+2	9.72E+2	9.72E+2	9.73E+2	9.72E+2	9.72E+2	9.72E+2
STAT96V3	1.04E+3	1.04E+3	1.04E+3	1.04E+3	1.04E+3	1.04E+3	1.05E+3	1.04E+3	1.04E+3	1.04E+3
STAT96V4	1.20E+2	1.20E+2	1.20E+2	1.20E+2	1.20E+2	1.20E+2	1.21E+2	1.20E+2	1.20E+2	1.20E+2
STORMG21K	8.96E+2	8.96E+2	-	-	-	8.96E+2	8.96E+2	-	8.96E+2	-
WATSON_1	2.54E+2	2.54E+2	2.54E+2	2.54E+2	2.54E+2	2.54E+2	2.55E+2	2.54E+2	2.54E+2	2.54E+2
WATSON_2	3.35E+2	3.35E+2	3.35E+2	3.35E+2	3.35E+2	3.35E+2	3.36E+2	3.35E+2	3.35E+2	3.35E+2
WORLD	1.40E+2	1.40E+2	1.40E+2	1.40E+2	1.40E+2	1.40E+2	1.41E+2	1.40E+2	1.40E+2	1.40E+2

Table A.6: Residuals obtained for subset UF problems by each method

name	no	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN	MA97	MA97-aug	SPQR
12month1	-	9.27E+2	9.27E+2	-	9.27E+2	9.27E+2	9.27E+2	-	-	-
162bit	1.17E+1	1.17E+1	1.17E+1	1.17E+1	1.17E+1	1.17E+1	1.18E+1	1.17E+1	1.17E+1	1.17E+1
176bit	1.84E+1	1.84E+1	1.84E+1	1.84E+1	1.84E+1	1.84E+1	1.84E+1	1.84E+1	1.84E+1	1.84E+1
192bit	2.48E+1	2.48E+1	2.48E+1	2.48E+1	2.48E+1	2.48E+1	2.49E+1	2.48E+1	2.48E+1	2.48E+1
208bit	-	3.85E+1	3.84E+1	3.85E+1	3.84E+1	3.84E+1	3.85E+1	3.84E+1	3.84E+1	3.84E+1
beaflw	5.05E+0	4.57E+0	4.60E+0	-	4.35E+0	4.69E+0	4.56E+0	4.16E+0	4.16E+0	4.16E+0
c8_mat11	2.21E+1	2.19E+1	-	-	-	2.19E+1	2.18E+1	2.12E+1	2.12E+1	2.12E+1
connectus	6.27E+2	6.27E+2	6.27E+2	6.27E+2	6.27E+2	6.27E+2	6.28E+2	6.27E+2	6.27E+2	6.27E+2
ESOC	4.04E+2	2.23E+1	-	-	-	-	4.36E-3	5.23E-10	-	5.97E-9
EternityII_Etilde	4.45E-6	4.51E-6	4.47E-6	4.43E-6	5.33E-6	4.42E-6	1.21E-4	3.56E-14	6.41E-14	6.31E-13
f855_mat9	1.79E+1	1.79E+1	-	-	-	1.79E+1	1.65E+1	2.14E+2	1.69E+1	4.37E+3
GL7d16	7.48E+2	7.48E+2	7.48E+2	-	7.48E+2	7.48E+2	7.49E+2	-	-	-
GL7d17	8.98E+2	8.98E+2	8.98E+2	-	8.98E+2	8.98E+2	-	-	-	-
GL7d18	9.31E+2	9.31E+2	-	-	9.31E+2	-	-	-	-	-
GL7d19	1.04E+3	1.04E+3	-	-	1.04E+3	1.04E+3	-	-	-	-
GL7d20	1.09E+3	1.09E+3	-	-	1.09E+3	1.09E+3	-	-	-	-
GL7d21	1.00E+3	1.00E+3	1.00E+3	-	1.00E+3	1.00E+3	-	-	-	-
GL7d22	7.89E+2	7.89E+2	7.89E+2	-	7.89E+2	7.89E+2	7.89E+2	-	-	-
GL7d23	5.35E+2	5.35E+2	5.35E+2	-	5.35E+2	5.35E+2	5.36E+2	-	-	-
graphics	-	-	-	-	-	3.03E-4	3.03E-4	3.03E-4	3.03E-4	3.03E-4
HFE18_96_in	4.91E-1	4.90E-1	4.90E-1	4.90E-1	4.90E-1	4.90E-1	4.91E-1	4.90E-1	4.90E-1	4.90E-1
IG5-15	5.48E+1	5.48E+1	5.48E+1	5.48E+1	5.48E+1	5.48E+1	5.49E+1	5.48E+1	5.48E+1	5.48E+1
IG5-16	7.15E+1	7.15E+1	7.15E+1	7.15E+1	7.15E+1	7.15E+1	7.15E+1	7.15E+1	7.15E+1	7.15E+1
IG5-17	9.10E+1	9.10E+1	9.10E+1	9.10E+1	9.10E+1	9.10E+1	9.10E+1	9.10E+1	9.10E+1	9.10E+1
IG5-18	1.15E+2	1.15E+2	1.15E+2	1.15E+2	1.15E+2	1.15E+2	1.15E+2	1.15E+2	1.15E+2	1.15E+2
IMDB	-	-	-	-	-	-	8.04E+2	-	-	-
kneser_10_4_1	1.62E+2	1.62E+2	-	-	-	1.62E+2	1.62E+2	-	-	1.63E+2
landmark	1.31E-5	1.12E-5	1.12E-5	1.12E-5	1.15E-5	1.12E-5	8.36E-5	1.12E-5	1.12E-5	1.12E-5
LargeRegFile	4.44E+2	4.44E+2	4.44E+2	-	4.44E+2	4.44E+2	4.44E+2	4.44E+2	4.44E+2	4.44E+2
Maragal_6	1.06E+1	1.06E+1	1.06E+1	-	1.06E+1	1.06E+1	1.07E+1	1.06E+1	1.06E+1	1.07E+1
Maragal_7	1.36E+1	1.36E+1	1.36E+1	1.36E+1	1.36E+1	1.36E+1	1.37E+1	1.36E+1	1.36E+1	1.37E+1
Maragal_8	-	-	-	-	-	-	2.39E+2	2.37E+2	2.37E+2	2.37E+2
mri1	2.67E+1	2.67E+1	2.67E+1	2.67E+1	2.67E+1	2.67E+1	2.67E+1	2.67E+1	2.67E+1	8.09E+13
mri2	1.41E+2	1.41E+2	1.41E+2	-	1.41E+2	1.41E+2	1.41E+2	-	-	9.24E+24
NotreDame_actors	-	-	-	-	-	-	5.19E+2	-	-	-
psse0	6.63E+1	6.63E+1	6.63E+1	6.63E+1	6.63E+1	6.63E+1	6.64E+1	6.63E+1	6.63E+1	6.63E+1
psse1	3.59E+1	3.59E+1	3.59E+1	-	3.59E+1	3.59E+1	3.59E+1	3.59E+1	3.59E+1	3.59E+1
psse2	7.85E+1	7.85E+1	7.85E+1	-	7.85E+1	7.85E+1	7.86E+1	7.85E+1	7.85E+1	7.85E+1
rel9	1.54E+3	1.54E+3	1.54E+3	-	1.54E+3	1.54E+3	-	-	-	-
relat9	3.05E+3	3.05E+3	3.05E+3	-	3.05E+3	3.05E+3	-	-	-	-
Rucci1	7.27E+2	7.27E+2	-	7.27E+2	-	-	7.28E+2	7.27E+2	7.27E+2	7.27E+2
sls	1.30E-5	1.23E-5	1.27E-5	-	2.09E-4	1.05E-5	1.74E-5	9.31E-14	1.41E-13	-
TF14	5.57E-7	5.58E-7	5.50E-7	5.55E-7	5.37E-9	5.61E-7	6.24E-3	6.63E-15	6.20E-15	5.82E-14
TF15	8.78E-7	8.78E-7	-	8.74E-7	-	8.77E-7	1.13E-2	1.07E-14	9.84E-15	1.31E-13
TF16	1.38E-6	-	-	-	-	1.38E-6	1.52E-2	1.75E-14	1.60E-14	3.09E-13
TF17	-	-	-	-	-	-	2.17E-2	-	-	-
TF18	-	-	-	-	-	-	3.21E-2	-	-	-
TF19	-	-	-	-	-	-	4.26E-2	-	-	-
tomographic1	4.20E+1	4.19E+1	-	6.84E+1	-	4.19E+1	4.19E+1	4.18E+1	4.18E+1	4.18E+1
Trec14	1.12E+2	1.12E+2	1.12E+2	-	1.12E+2	1.12E+2	1.12E+2	1.12E+2	1.12E+2	1.12E+2
wheel_601	-	-	-	-	-	-	4.23E+2	-	4.22E+2	-