Science & Technology Facilities Council

Preprint

RAL-P-2015-010 (corrected)

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

NIM Gould, J Scott,

November 2015 (corrected November 2016)

Submitted for publication in ACM Transactions in Mathematical Software (accepted)

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.

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 the development of the principle of least squares by Gauss in 1795 [23], 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 they are always consistent and positive semidefinite (positive definite if A is of full column rank). 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 [10]. 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 methods 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 [12], "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 [12]. 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 10 and finally, in Section 11, 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 (serial) or Intel MKL (serial vs parallel)

the direct solvers (see Section 3) are run in parallel, using four processors. Our initial experiments on iterative methods (those in Sections 4 and 8) are run on a single processor, although where BLAS are used, these may take advantage of parallelism. Later, when comparing iterative and direct approaches (in Sections 9 and 10), we repeat the calculations on 4 processors for the methods we have found to be best. Here sparse matrix-vector products and sparse triangular solves (if any) required by the preconditioner are performed in parallel using Intel Mathematics Kernel Library (MKL) routines; no attempt is made to parallelize any of the iterative methods themselves, nor the software for constructing the preconditioners.

Throughout this study, all reported times are elapsed times in seconds measured using the Fortran `system_clock`. For each solver and each problem, a 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 10^7. 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). In order to obtain close-to-repeatable times when running in parallel on 4 processors, we specify which processors are to be used via the Linux `numactl -C 0,1,2,3` command.

2.1 Test problems

The problems used in our study are all taken from the CUTEst linear programme set [26] and the UFL Collection [18]. 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. If values for the matrix entries are not supplied, we generate random values in the range $(-1, 1)$.

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 (when run on a single processor). 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 [30] for details of the full set).

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 $r(x) = b - Ax$ 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 [24] in their 2010 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. We note that new results on estimating backward errors for least-squares problems have been derived by a number of authors, including [31, 42].

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 [19], 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 [29]. 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 10, 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 different 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 (1.2), a positive definite solver that computes a sparse Cholesky factorization can be used, such as CHOLMOD [14] or `HSL_MA87` [37]. Alternatively, there are sparse packages that can factorize both positive definite and indefinite systems. These include a number of HSL codes (notably, `MA57` [20], `HSL_MA86` [39], and `HSL_MA97` [38]) as well as MUMPS [50], WSMP [33], PARDISO [56] and `SPRAL_SSIDS` [36]. Comparisons of some of these packages for solving sparse linear systems may be found in [25, 28]. When used to solve the augmented system (1.3), 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 conduct a detailed comparison of the performance of direct solvers on least-squares problems; instead we opt to use `HSL_MA87` (Version 2.4.0) for the normal equations and `HSL_MA97` (Version 2.3.0) for the augmented system in our comparisons with iterative methods. This choice was made since `HSL_MA87` and `HSL_MA97` are recent state-of-the-art packages that are designed for multicore machines, and, because we are responsible for their development, we find them 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 [17] and qr_mumps [13]. Here we use SuiteSparseQR version 4.4.4 (for which we have written a Fortran interface).

Although a straightforward application of a direct method to (1.2) or (1.3) is usually successful, computer rounding can have a profound effect. In particular, for problems that are (or are close to) rank deficient, simply forming the (theoretically) positive semidefinite normal matrix C may result in a matrix that is slightly indefinite and a Cholesky factorization will breakdown. Likewise, the symmetric, indefinite factorization of the augmented matrix K may produce a numerical inertia (i.e., a count of the numbers of positive, negative and zero eigenvalues) that is impossible had the matrix been factorized exactly. Thus, in addition to applying the appropriate factorization routine to our test problems, we also consider employing a "scale-and-shift" strategy that aims to reduce the impact of poor conditioning. In particular, we modify the normal equations and augmented system so that the required solution x is $x = Sz$, where z is the solution of the system

$$\bar{C}z = \bar{A}^T b, \qquad \bar{C} = \bar{A}^T\bar{A} + \delta_C I_n, \tag{3.1}$$

or

$$\bar{K}\bar{y} = c, \qquad \bar{K} = \begin{bmatrix} I_m & \bar{A} \\ \bar{A}^T & -\delta_K I_n \end{bmatrix}, \qquad y = \begin{bmatrix} r(x) \\ z \end{bmatrix}, \qquad c = \begin{bmatrix} b \\ 0 \end{bmatrix}, \tag{3.2}$$

where $\bar{A} = AS$ and the diagonal matrix S scales the columns of A so that each has a unit 2-norm, and the scalars $0 < \delta_C, \delta_K \ll 1$. In our experiments, we have found that $\delta_C = 10^{-12}$ and $\delta_K = 10^{-10}$ are generally suitable choices, and although they necessarily perturb the computed solution, our experience is that the perturbation is sufficiently small to be acceptable. Similar regularizations have been proposed by many authors, e.g., [63, 65]. On our test set $\mathcal{T}$, `HSL_MA87` and `HSL_MA97` both failed to solve 26 of the 83 problems without prescaling and shifting (the solvers used their own default scalings of C and K, respectively), and this reduced to 15 and 20, respectively, with prescaling and shifting (see Tables 4.17–4.20 in [30] for details). Thus, in what follows, we use the HSL direct solvers combined with the scale-and-shift approach in all remaining comparisons.

A time performance profile comparing SuiteSparseQR (denoted by SPQR), `HSL_MA87` applied to the normal equations (MA87 normal equations) and `HSL_MA97` applied to the the augmented system (MA97

augmented system) is given in Figure 3.1. In our experiments, one step of iterative refinement was used. We see that using HSL_MA87 for the normal equations leads to the smallest number of failures while it is the fastest approach for more than half of the problems. The failures are for some of the largest problems and are because of insufficient memory (see Tables 4.17–4.21 in [30] for details). In addition, for SPQR there are three problems (f855_mat9, mri1 and mri2) for which no error is flagged but the returned residuals are clearly too large when compared with those computed by the other solvers. Although a direct solver such as HSL_MA77 [57] 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.

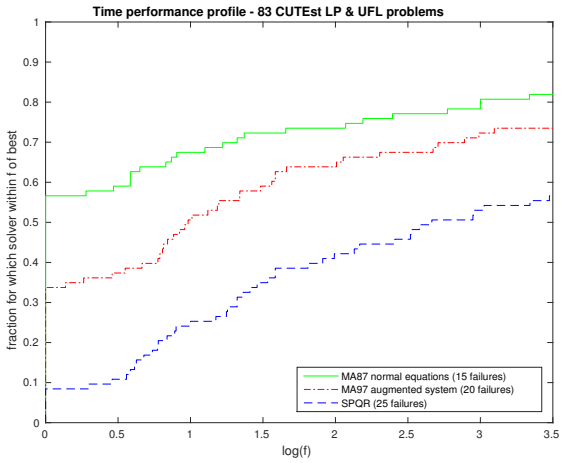

Figure 3.1: Time performance profile for the direct solvers HSL_MA87, HSL_MA97 and SuiteSparseQR (SPQR) for test set $\mathcal{T}$.

4 LSQR vs LSMR

CGLS (or CGNR) [35] 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 [53, 54] is algebraically identical to CGLS and, as shown in [11], both achieve similar final accuracy consistent with numerical stability. 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 [24] is similar to LSQR in that it too is based on Golub-Kahan bidiagonalization of A. However, in exact arithmetic LSMR is equivalent to MINRES [52] 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.

The implementation of LSMR used in this paper is a slightly modified version of the 2010 one of Fong and Saunders. 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

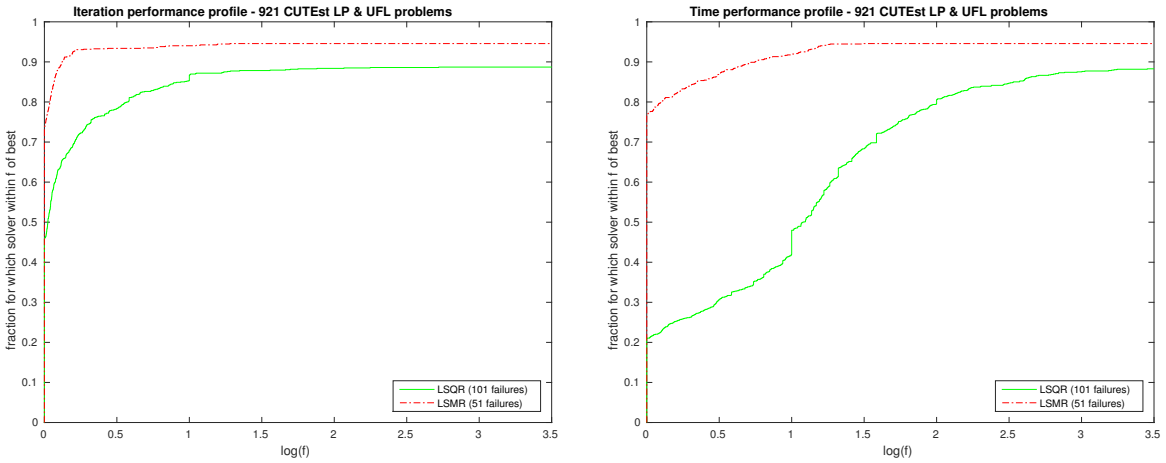

Figure 4.1: Iteration performance profile (left) and time performance profile for LSMR and LSQR (right) for the complete CUTEst and UFL test set of 921 eligible problems with no preconditioning.

modifications are made to LSQR for our tests. Both the modified version of LSMR and the Fong and Saunders code are available from `http://web.stanford.edu/group/SOL/download.html`.

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 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 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 `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 solve

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

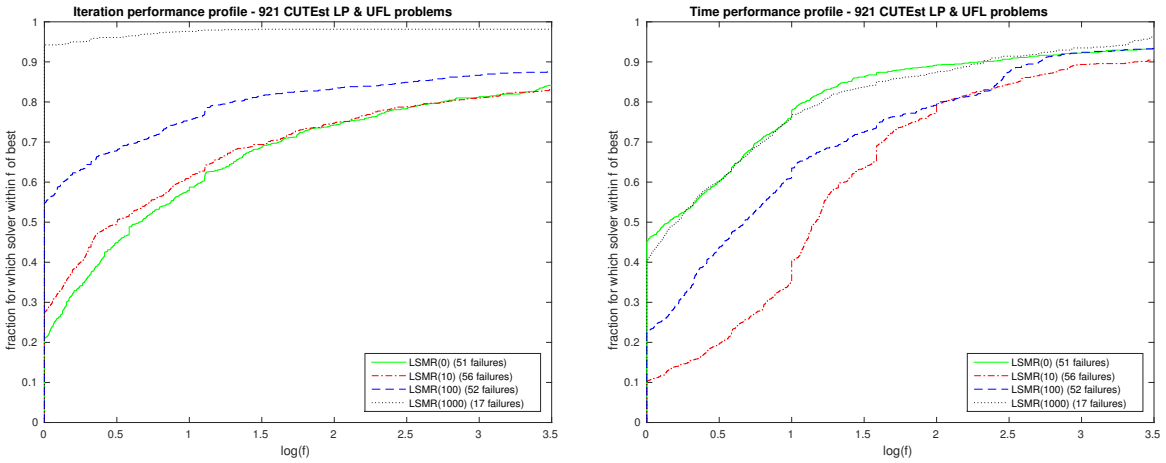

Figure 4.2: Iteration performance profile (left) and time performance profile for LSMR with a range of values of `localSize` for the complete CUTEst and UFL test set of 921 eligible problems with no preconditioning.

where S is a diagonal matrix that scales the columns of A to give each 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, [7, 60, 67] and the long lists of references therein. Since (if A has full column rank) the coefficient matrix C of the normal equations (1.2) is positive definite, an obvious choice for the preconditioner is an IC factorization of 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 preconditioned normal equations become

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

Performing preconditioning operations involves solving triangular systems with L and L^T. Over the years, a wealth of different variants have been proposed, including structure-based methods, those based on dropping entries below a prescribed threshold and others that prescribe the maximum number of entries allowed in L. We employ the recent limited memory approach of Scott and Tůma [66, 67] that exploits ideas from the positive semidefinite Tismenetsky-Kaporin modification scheme [44, 71]. The basic scheme employs 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

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

for details, see [66, 67]. The user specifies the maximum number of entries in each column of L and $\hat{L}$. At each step of the factorization, the largest entries are kept in the current column of L, the next largest

in the current column of $\hat{L}$, and the remainder are dropped. In practice, C is optionally preordered and scaled and, if necessary, shifted to avoid breakdown of the factorization (which occurs if a non positive pivot is encountered) [47]. Thus the LL^T incomplete factorization of the matrix

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

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.

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 [4, 5, 41, 55, 59, 72]. Most recently, there is the Multilevel Incomplete QR (MIQR) factorization of Li and Saad [46].

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 [8, 9] 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

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

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

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

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 [8] 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 [9] 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, 12, 46, 48].

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 [61] applied to the normal equations. For problems for which convergence is slow and for very large problems for which storage is an issue, 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. [34, 48, 49]. In particular, for overdetermined least-squares problems, they propose using Jacobi- (Cimmino [16]) and SOR-type (Kaczmarz [43]) 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 [48, 49].

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 [48, 49] 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 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 [60], 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 [48, 49] 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, [51]). A number of possible approaches exist, including employing an incomplete factorization designed for general indefinite symmetric systems or a signed incomplete Cholesky factorization [68] designed specifically for systems of the form (1.3). Chow and Saad [15] considered the class of incomplete LU preconditioners for solving indefinite problems and later Li and Saad [45] integrated pivoting procedures with scaling and reordering. Building on this, Greif, He, and Liu [32] 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 [69] 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 [32]. 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 [68, 69]). 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 [68] 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 [68] 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 [61] 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 $\delta_3 = 10^{-6}$.

If we want to use a solver that is designed for symmetric indefinite systems, in place of GMRES we can use MINRES [52]. 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 [40]. This code is a modified version of `HSL_MI28` [66]. 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` [58] and

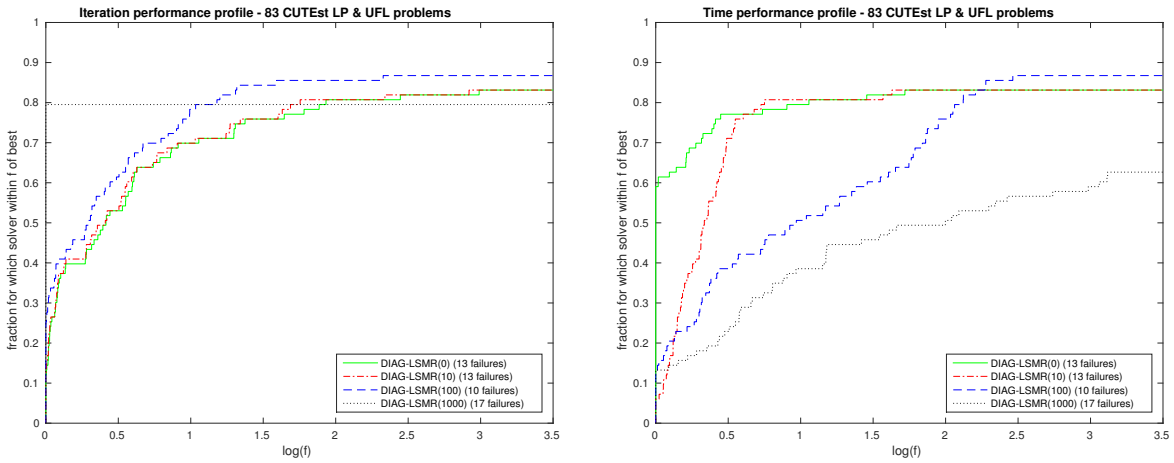

Figure 8.1: Iteration performance profile (left) and time performance profile (right) for LSMR with diagonal preconditioning using a range of values of `localSize` for test set $\mathcal{T}$.

`MC64` [21, 22], respectively, require C explicitly. The default scaling is l_2 scaling, in which column j of 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. Here and elsewhere, the time used for the time performance profile are the total solution time (that is, the time to compute the preconditioner plus the time to run preconditioned LSMR). 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`.

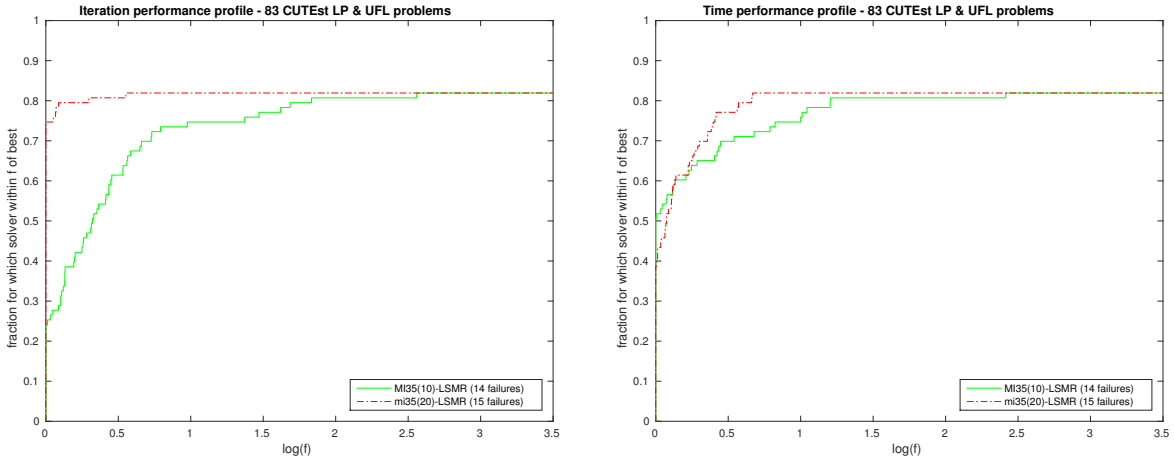

Figure 8.2: Iteration performance profile (left) and time performance profile (right) for LSMR preconditioned by `HSL_MI35` with `lsize` = `rsize` = 10 and `lsize` = `rsize` = 20 for test set $\mathcal{T}$.

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`.

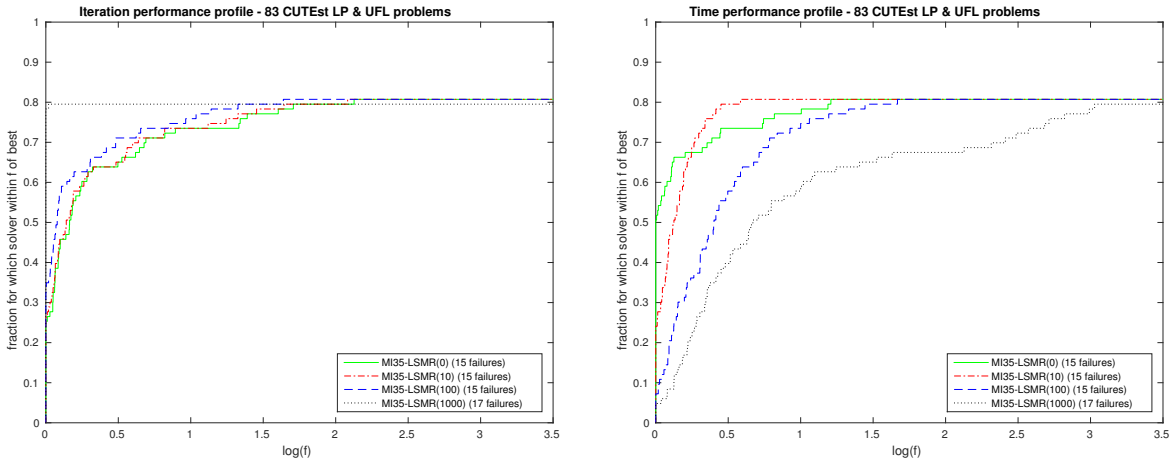

Figure 8.3: Iteration performance profile (left) and time performance profile (right) for LSMR preconditioned by `HSL_MI35` using a range of values of `localSize` for test set $\mathcal{T}$.

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 [27]. This is essentially a translation of Li and Saad [46]'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.

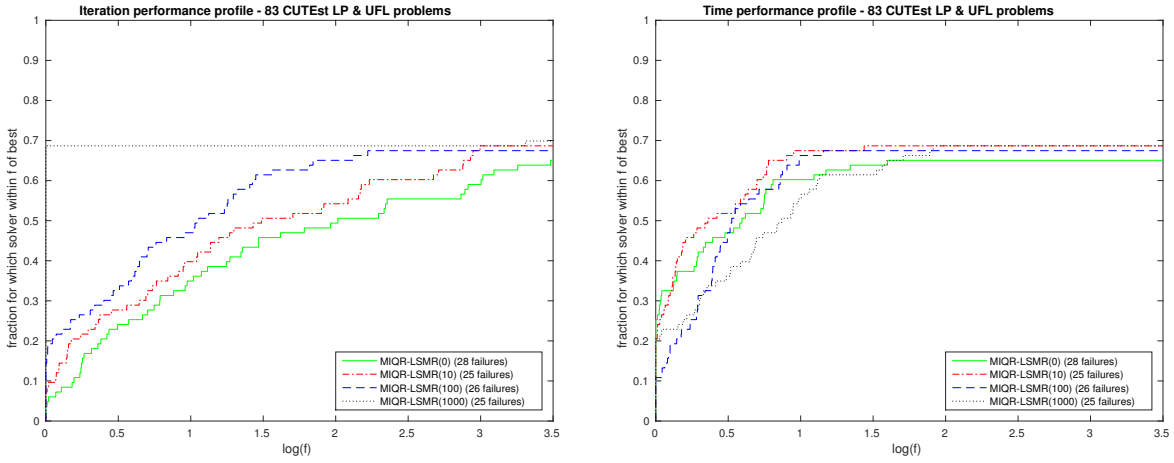

Figure 8.4: Iteration performance profile (left) and time performance profile (right) for LSMR with MIQR preconditioning using a range of values of `localSize` for test set $\mathcal{T}$.

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 (see [70] for details of the right- and left-looking variants). The latter works only with A and A^T and has the advantage that the required memory can be computed before the factorization begins using a symbolic preprocessing step [70]. 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`. There is no uniformly best value; in the rest of our experiments, we set `localSize` to 10.

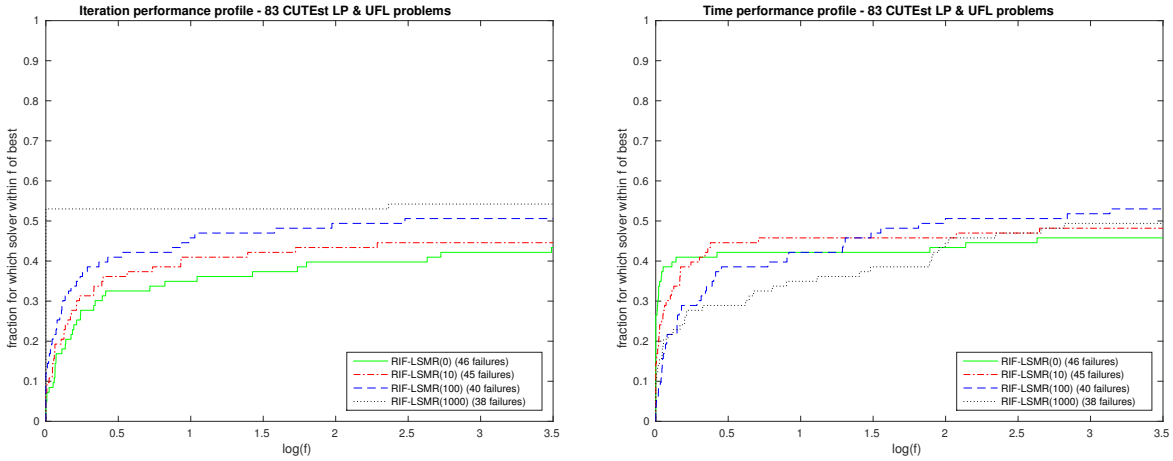

Figure 8.5: Iteration performance profile (left) and time performance profile (right) for LSMR with RIF preconditioning using a range of values of `localSize` for test set $\mathcal{T}$.

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

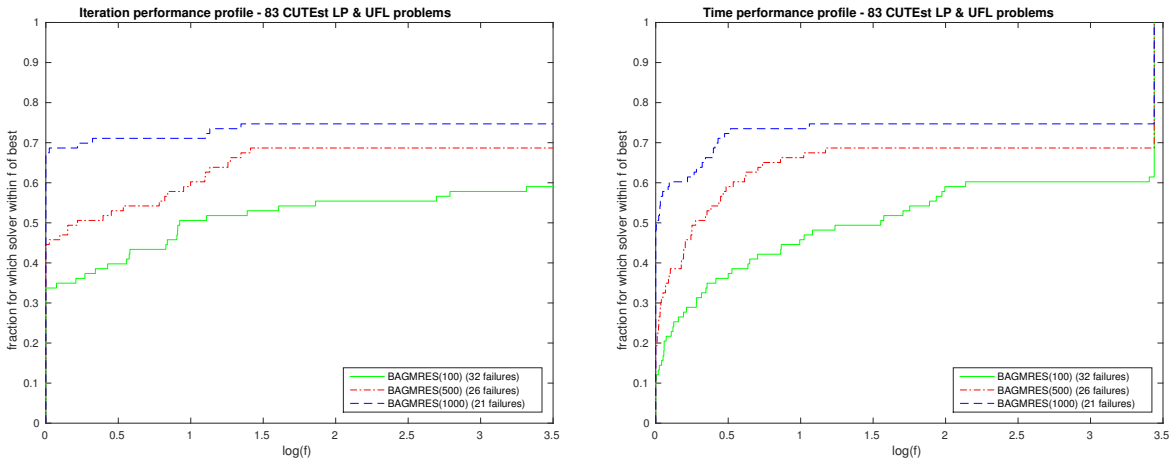

Figure 8.6: Iteration performance profile (left) and time performance profile for BA-GMRES with different restart parameters for test set $\mathcal{T}$.

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 [48, 49]. We use NR-SOR inner iteration with automatic parameter selection in our tests.

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 [68]. 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 Benefits of simple parallelism

Having selected what we consider to be good parameter settings, a natural question is how much the methods in question might benefit from simple parallelism? Figure 9.1 illustrates the improvements possible simply by performing sparse matrix-vector products and triangular preconditioning solves on 4 rather than a single processor. Each preconditioner gains from parallelism, and perhaps unsurprisingly the most significant gains are for those methods for which (MKL-assisted) matrix-vector products and triangular solves dominate the solution time. Note that the serial experiments were necessarily repeated to obtain this data, since here we use MKL BLAS while our earlier experiments used the open BLAS. On the basis of these findings, our remaining comparisons use data from these runs on 4 processors.

10 Solver comparison results

10.1 Performance comparison for preconditioning LSMR

Figure 10.1 presents iteration and time performance profiles for LSMR run both without preconditioning and with diagonal, MIQR, RIF and IC (`HSL_MI35`) preconditioning run on 4 processors. Here we chose `localSize` = 0 for no preconditioning and diagonal preconditioning and `localSize` = 10 for MIQR,

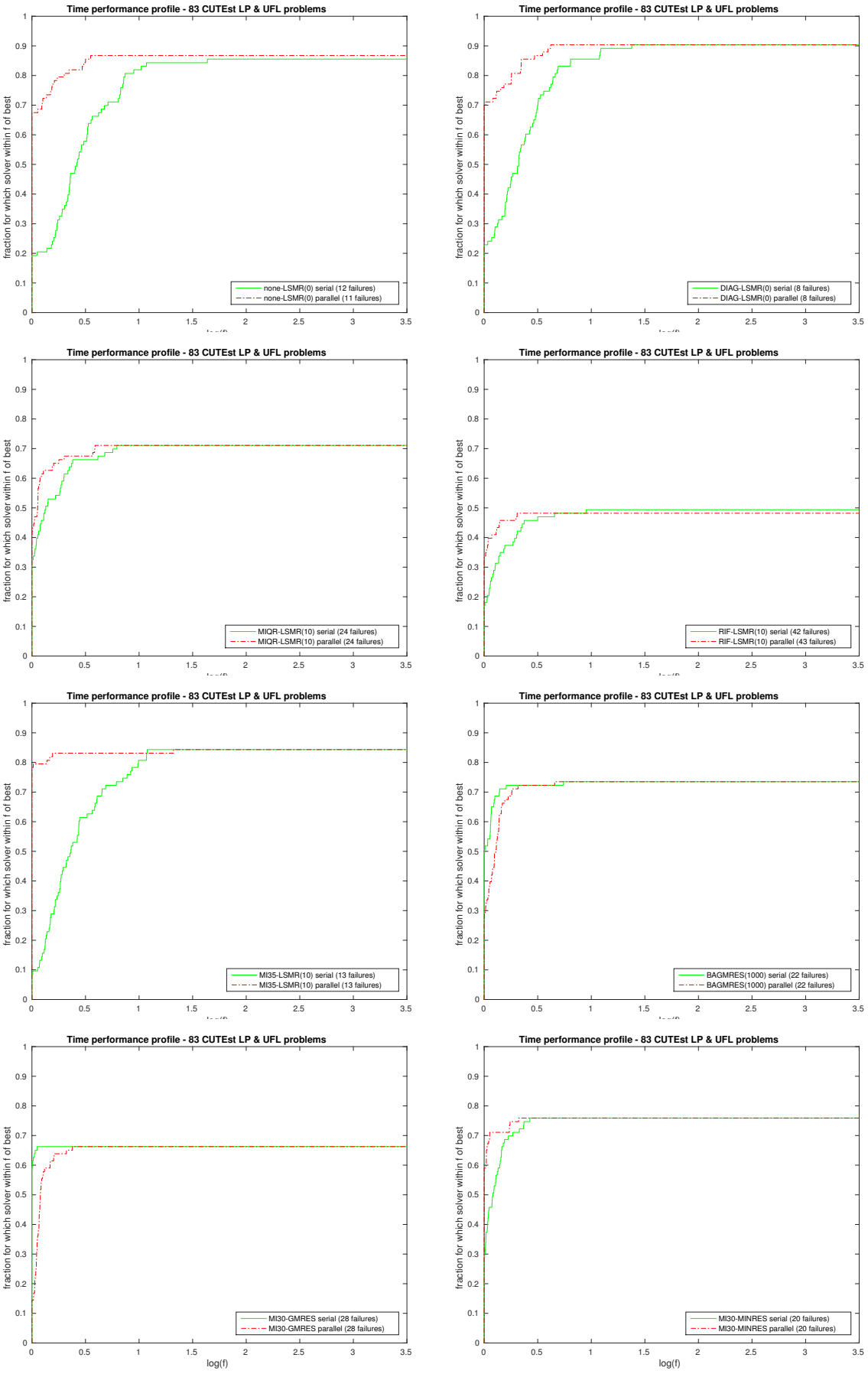

Figure 9.1: Time performance profiles for serial and parallel execution of the methods described in Section 8 for test set $\mathcal{T}$.

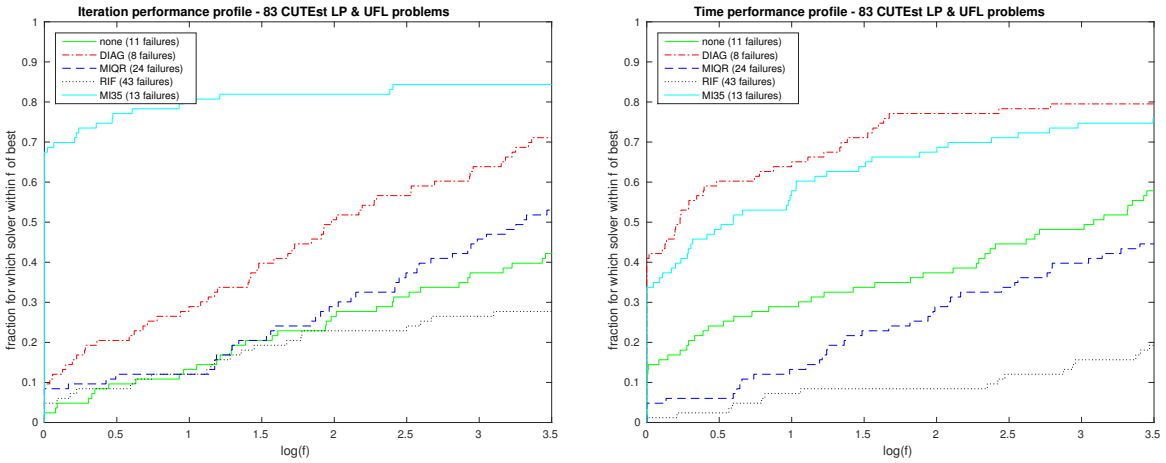

Figure 10.1: Iteration performance profile (left) and time performance profile for different preconditioners used with LSMR for test set $\mathcal{T}$.

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 10.2. We observe that Morikuni and Hayami [48] also found diagonal preconditioning to give the fastest solution times in some of their tests.

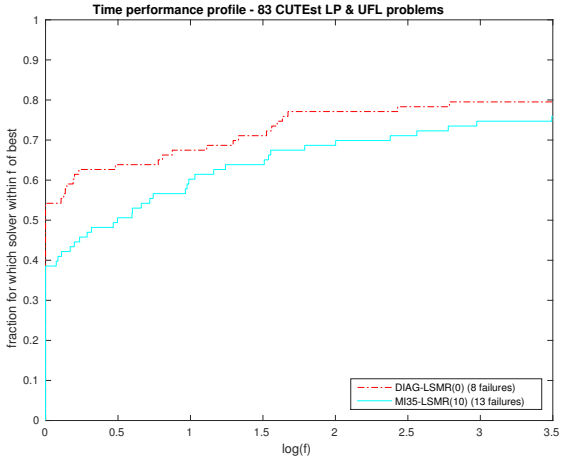

Figure 10.2: Time performance profile for diagonal and IC preconditioners used with LSMR for test set $\mathcal{T}$.

In Figure 10.3 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 10.4 when RIF is removed from the picture.

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

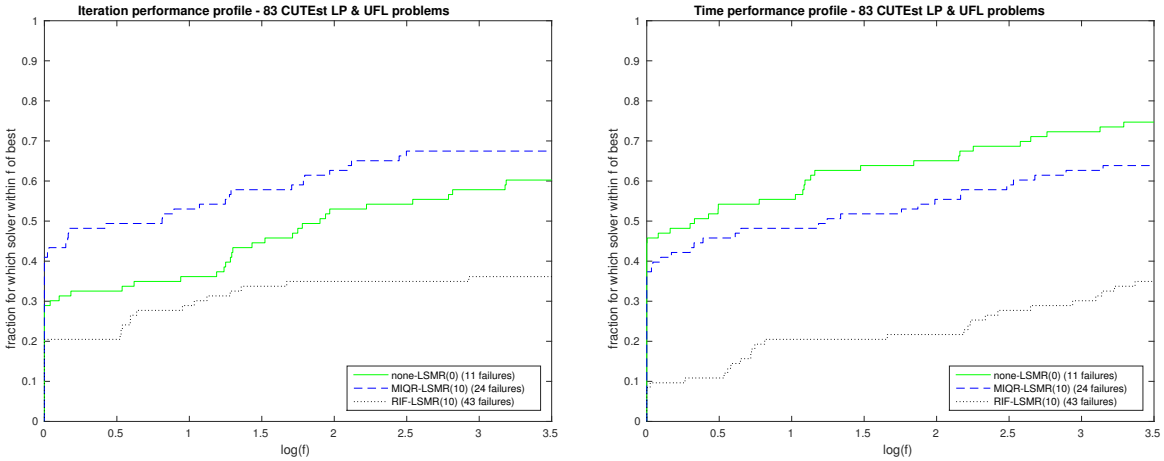

Figure 10.3: Iteration and time performance profiles for LSMR with no preconditioning and MIQR and RIF preconditioning for test set $\mathcal{T}$.

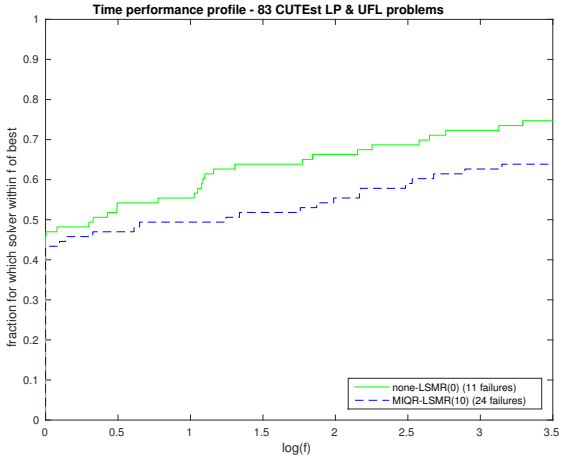

Figure 10.4: Time performance profile for LSMR with no preconditioning and MIQR preconditioning for test set $\mathcal{T}$.

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 [70]. 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 [48]. 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.

10.2 Performance comparison with BA-GMRES

Time performance profiles for BA-GMRES are given in Figure 10.5. 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 [30]) 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.

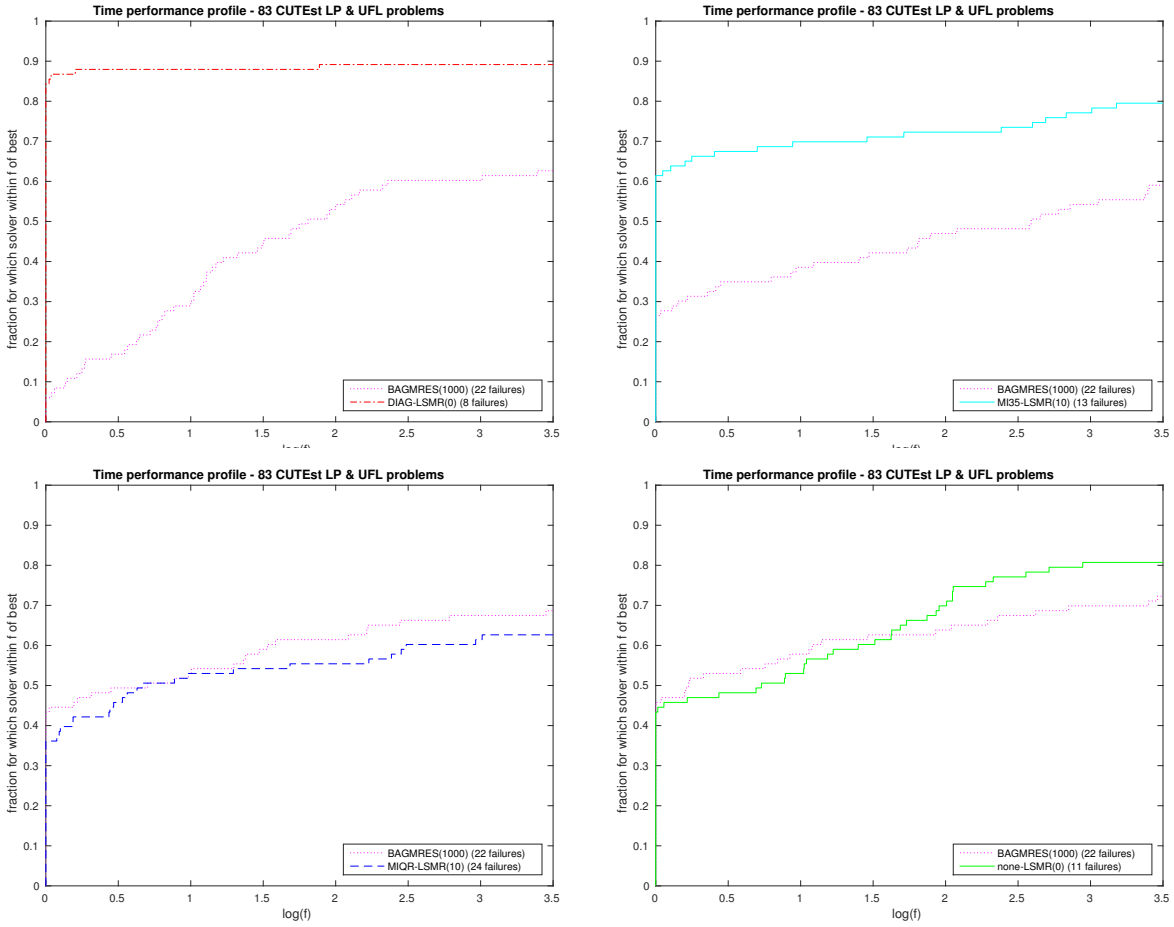

Figure 10.5: Time performance profile for BA-GMRES(1000) and LSMR with diagonal, IC (HSL_MI35), MIQR and no preconditioning for test set $\mathcal{T}$.

10.3 Performance comparison with signed incomplete factorization

In Figure 10.6, 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.

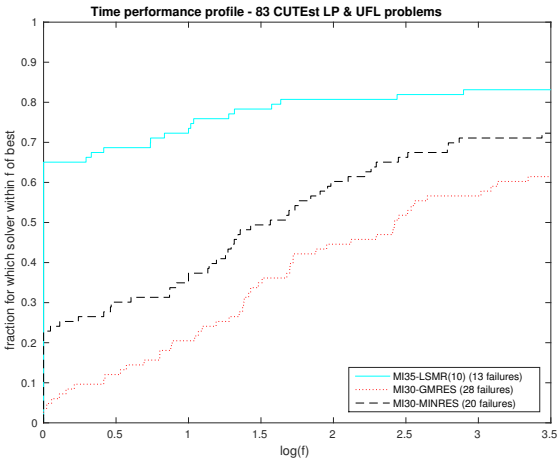

Figure 10.6: Time performance profile for LSMR with IC (`HSL_MI35`) preconditioning and GMRES(1000) and MINRES with signed IC (`HSL_MI30`) preconditioning for test set $\mathcal{T}$.

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} then either C1 or C2 is satisfied (see Tables 3.25 and 3.26 in [30]). 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 to solve problem IMDB to the required accuracy (with our time and iteration count limits) using any of the direct solvers or preconditioners in this study, while problems NotreDame_actors, TF17, TF18, TF19 and wheel_601 proved impossible to all but a few solvers.

10.4 Performance comparison with a direct solver

In Figure 10.7, we present time performance profiles for the direct solver `HSL_MA87`, applied to the (prescaled and slightly shifted) normal equations (3.1), and for diagonal and IC (`HSL_MI35`) preconditioned LSMR. We see that for the set $\mathcal{T}$ the direct solver is the fastest for almost 70% of the problems, but it is unable to solve 18% of the problems for which there was insufficient memory. If we look at the results for individual

problems given in Table A.6, we see that both diagonal and IC preconditioned LSMR solve some problems that HSL_MA87 fail on, including SPAL_004 and the GL7dxx examples.

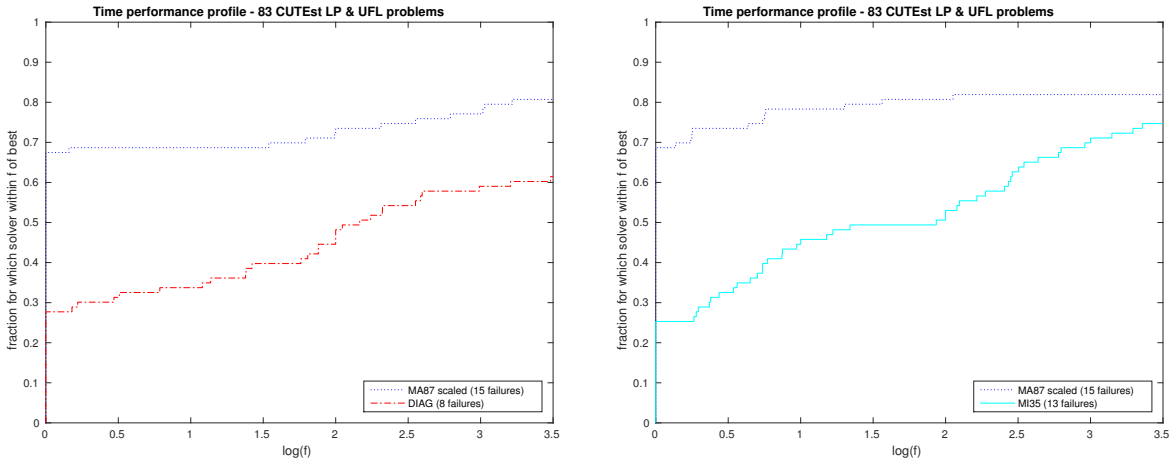

Figure 10.7: Time performance profile for direct solver HSL_MA87 and diagonal and IC (HSL_MI35) preconditioned LSMR for test set $\mathcal{T}$.

10.5 Summary tables

In Tables A.2–A.6, 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_MA87 (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 [30]. 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 (using 4 processors) and report the computed least squares residual. 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. Similarly, we omit columns for the space and times required to obtain factors when no preconditioning is used in Tables A.2 and A.4, respectively, as well as the factorization times for BA-GMRES from the later, as the computation of this preconditioner is integrated within the overall algorithm. A − indicates that the run was unsuccessful; again, for full details the reader is referred to [30].

11 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 [30]. 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 [24] 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 [6] and the balanced incomplete factorization (BIF) preconditioner of Bru, Marín Mas and Tůma [12]. LU preconditioning, which was discussed by Saunders [62] in 1979 (see also Section 7.5.3 of the book by Björck [10]), has also received renewed attention (see the 2015 paper by Arioli and Duff [3] and presentation by Saunders [64]). 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 [6], experimental results are reported using MATLAB codes. Unfortunately, the recent Fortran results reported by Saunders [64] 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. In particular, we recommend that future comparisons of linear least-squares software includes the test examples PDE1, IMDB, GLRD17–21, NotreDame_actors, TF17–19 and wheel_601, since these challenge many of the methods we have considered here.

Acknowledgements

We are grateful to Michael Saunders for a number of discussions related to his LSQR and LSMR software packages and for making our reverse communication implementation of LSMR available on his web page. We thank Miroslav Tůma for help with understanding and employing his RIF code. We would also like to thank four anonymous reviewers for their constructive feedback.

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] Z.-Z. Bai, I. S. Duff, and A. J. Wathen, *A class of incomplete orthogonal factorization methods. I: Methods and theories*, BIT Numerical Mathematics, 41 (2001), pp. 53–70.

[5] Z.-Z. Bai, I. S. Duff, and J.-F. Yin, *Numerical study on incomplete orthogonal factorization preconditioners*, J. of Computational and Applied Mathematics, 226 (2009), pp. 22–41.

[6] 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.

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

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

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

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

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

[12] 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.

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

[14] 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.

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

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

[17] 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.

[18] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Transactions on Mathematical Software, 38 (2011), pp. 1:1–1:25.

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

[20] 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.

[21] 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.

[22] ———, *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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] ———, *The state-of-the-art of preconditioners for sparse linear least squares problems: the complete results*, Technical Report RAL-TR-2015-009 (revision 1), Rutherford Appleton Laboratory, 2016.

[31] J. F. Grcar, M. A. Saunders, and Z. Su, *Estimates of optimal backward perturbations for linear least squares problems*, Technical Report SOL 2007-1, Department of Management Science and Engineering, Stanford University, Stanford, 2007.

[32] 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/`.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

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

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

[41] 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.

[42] P. Jiranek and D. Titley-Peloquin, *Estimating the backward error in LSQR*, SIAM J. on Matrix Analysis and Applications, 31 (2010), pp. 2055–2074.

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

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

[45] 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.

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

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

[48] 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.

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

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

[51] 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.

[52] 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.

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

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

[55] 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.

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

[57] 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.

[58] 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.

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

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

[61] 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.

[62] 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.

[63] ———, *Cholesky-based methods for sparse least squares: The benefits of regularization*, in Linear and Nonlinear Conjugate Gradient-Related Methods, L. Adams and J. L. Nazareth, eds., Philadelphia, USA, 1996, SIAM, pp. 92–100.

[64] ———, *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`.

[65] J. A. Scott, *On using Cholesky-based factorizations for solving rank-deficient sparse linear least-squares problems*, Technical Report RAL-P-2016-006, Rutherford Appleton Laboratory, 2016.

[66] 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.

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

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

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

[70] ———, *Preconditioning of linear least squares by RIF for implicitly held normal equations*, Technical Report RAL-TR-2016-P-001, Rutherford Appleton Laboratory, 2016.

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

[72] 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 * indicates a right-hand side vector b was supplied and − 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	6.75E-2	8.87E-2
BAXTER	30733	27441	111576	2993	1.68E-3	1.63E-3
BCDOUT	7078	5412	67344	2	1.55E-1	6.86E-2
CO9	22924	10789	109651	0	2.60E-3	2.14E-3
CONT11_L	1961394	1468599	5382999	0	4.77E-6	8.38E-6
DBIR1	45775	18804	1077025	103	1.19E-2	6.89E-3
DBIR2	45877	18906	1158159	101	1.23E-2	7.38E-3
D2Q06C	5831	2171	33081	0	1.57E-2	1.19E-2
DELF000	5543	3128	13741	0	2.88E-3	2.74E-3
GE	16369	10099	44825	0	3.56E-3	1.10E-3
LARGE001	7176	4162	18887	0	2.64E-3	2.46E-3
LPL1	129959	39951	386218	44	4.00E-4	3.39E-4
MOD2	66409	34774	199810	0	4.60E-4	5.00E-4
MODEL10	16819	4400	150372	0	3.86E-3	1.51E-2
MPSBCD03	7078	5412	66210	2	1.55E-1	6.82E-2
NSCT2	37563	23003	697738	287	2.73E-2	1.57E-2
NSIR2	10057	4453	154939	0	5.28E-2	2.39E-2
PDE1	271792	270595	990587	-	6.70E-1	-
PDS-100	514577	156016	1096002	227	1.92E-5	6.04E-5
PDS-90	475448	142596	1014136	227	2.10E-5	6.71E-5
PILOT-JA	2267	940	14977	0	5.85E-2	3.36E-2
PILOTNOV	2446	975	13331	0	4.10E-2	2.65E-2
RAIL2586	923269	2586	8011362	0	4.64E-3	7.05E-2
RAIL4284	1096894	4284	11284032	0	2.80E-3	1.19E-1
SPAL_004	321696	10203	46168124	0	1.65E-2	4.99E-1
STAT96V2	957432	29089	2852184	0	4.13E-4	4.17E-4
STAT96V3	1113780	33841	3317736	0	3.55E-4	3.58E-4
STAT96V4	63076	3173	491336	0	2.84E-3	5.43E-3
STORMG21K	1377306	526185	3459881	0	1.93E-3	3.00E-4
WATSON_1	386992	201155	1055093	0	4.47E-5	4.79E-5
WATSON_2	677224	352013	1846391	0	4.26E-5	2.74E-5
WORLD	67147	34506	198883	0	4.64E-4	4.89E-4
UF Sparse Matrix Collection examples						
12month1	872622	12471	22624727	-	2.74E-1	6.87E-1
162bit	3606	3476	37118	16	4.03E-3	1.95E-2
176bit	7441	7150	82270	40	2.24E-3	1.03E-2
192bit	13691	13093	154303	82	1.22E-3	5.73E-3
208bit	24430	23191	299756	199	7.76E-4	3.56E-3
beaflw	500	492	53403	4	8.13E-1	8.94E-1

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	5.30E-1	8.12E-1
connectus	394707	458	1127525	0	1.59E-1	1.58E-1
ESOC	327062	37349	6019939	0	5.09E-4	5.16E-3
EternityII_Etilde	204304	10054	1170516	0	6.96E-4	1.70E-2
f855_mat9	2511	2456	171214	0	3.38E-1	7.44E-1
GL7d16	955127	460260	14488881	-	1.39E-4	9.46E-4
GL7d17	1548649	955127	25978098	-	7.22E-5	4.43E-4
GL7d18	1955309	1548645	35590540	-	4.71E-5	2.54E-4
GL7d19	1955296	1911130	37322725	-	2.83E-5	1.97E-4
GL7d20	1911124	1437546	29893084	-	2.99E-5	2.23E-4
GL7d21	1437546	822922	18174775	-	4.37E-5	3.26E-4
GL7d22	822906	349443	8251000	-	7.44E-5	6.28E-4
GL7d23	349443	105054	2695430	-	1.81E-4	1.65E-3
graphics	29493	11822	117954	0	3.38E-4	5.91E-4
HFE18_96_in	2372	2371	933343	0	5.07E-1	9.91E-1
IG5-15	11369	6146	323509	0	1.95E-2	1.52E-1
IG5-16	18846	9519	588326	0	1.26E-2	1.28E-1
IG5-17	30162	14060	1035008	0	8.53E-3	1.14E-1
IG5-18	47894	20818	1790490	0	5.76E-3	9.91E-2
IMDB	896302	303617	3782463	-	5.24E-3	1.51E-3
kneser_10_4_1	349651	330751	992252	-	4.84E-5	8.89E-5
landmark	71952	2673	1146848	2	5.99E-3	1.68E-2
LargeRegFile	2111154	801374	4944201	0	4.99E-6	9.93E-6
Maragal_6*	21251	10144	537694	516	5.86E-1	7.49E-1
Maragal_7*	46845	26525	1200537	2046	3.60E-1	3.10E-1
Maragal_8*	60845	33093	1308415	7107	5.03E-2	3.56E-2
mri1	114637	65536	589824	603	3.66E-3	2.57E-4
mri2	104597	63240	569160	-	6.60E-2	7.84E-3
NotreDame_actors	383640	127823	1470404	-	5.05E-3	2.52E-3
psse0*	26722	11028	102432	0	3.63E-4	5.88E-4
psse1*	14318	11028	57376	0	1.63E-3	6.67E-4
psse2*	28634	11028	115262	0	2.54E-3	7.68E-4
rel9	5921786	274667	23667183	-	1.46E-5	5.09E-4
relat9	9746232	274667	38955420	-	1.46E-5	5.09E-4
Rucci1	1977885	109900	7791168	0	3.64E-5	8.07E-4
sls	1748122	62729	6804304	0	6.38E-5	1.20E-3
TF14	3159	2644	29862	0	4.92E-3	3.12E-2
TF15	7741	6334	80057	0	2.21E-3	1.63E-2
TF16	19320	15437	216173	0	9.72E-4	8.17E-3
TF17	48629	38132	586218	-	4.20E-4	3.98E-3
TF18	123867	95368	1597545	-	1.78E-4	1.88E-3
TF19	317955	241029	4370721	-	7.47E-5	8.70E-4
tomographic1*	59360	45908	647495	3436	3.27E-4	8.68E-4
Trec14	15904	3159	2872265	0	7.91E-1	9.32E-1
wheel_601	902103	723605	2170814	-	8.32E-4	4.22E-4

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

name	m	n	$nz(A)$	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN	MA87
BAS1LP	9825	5411	587775	5411	154335	58422	6419411	113414	858406	2525076
BAXTER	30733	27441	111576	27441	166672	254733	27441	218513	709648	6763027
BCDOUT	7078	5412	67344	5412	227118	38953	5412	97350	214193	2155189
C09	22924	10789	109651	10789	139168	52621	11802789	122520	346055	1233194
CONT11_L	1961394	1468599	5382999	1468599	>6823917	4883216	149807602	10208898	22208667	128552555
DBIR1	45775	18804	1077025	18804	306833	92947	7055662	264644	1977702	4589933
DBIR2	45877	18906	1158159	18906	311225	94898	17102778	172316	1876024	4846892
D2Q06C	5831	2171	33081	2171	22404	13660	700243	30258	100726	203218
DELF000	5543	3128	13741	3128	9959	11667	4134128	11314	33260	136175
GE	16369	10099	44825	10099	98350	49164	8719998	134483	248210	710847
LARGE001	7176	4162	18887	4162	14243	16930	5169162	16000	46375	204012
LPL1	129959	39951	386218	39951	184639	143744	28994143	541699	1605246	7939663
MOD2	66409	34774	199810	34774	559518	229986	20505898	527770	1300573	4521661
MODEL10	16819	4400	150372	4400	34670	24083	1863308	78015	378047	659462
MPSBCD03	7078	5412	66210	5412	228815	38965	5412	98584	220678	2055166
NSCT2	37563	23003	697738	23003	561767	109287	4781258	165302	1407946	8399426
NSIR2	10057	4453	154939	4453	98551	27196	984313	54142	305406	635304
PDE1	271792	270595	990587	270595	>7750326	>996368	270595	-	-	-
PDS-100	514577	156016	1096002	156016	1027112	618440	12019236	2891511	7696832	57854930
PDS-90	475448	142596	1014136	142596	950792	570713	10557652	2626289	7233396	51441581
PILOT-JA	2267	940	14977	940	9650	5780	427458	10870	47073	105608
PILOTNOV	2446	975	13331	975	9320	5958	449095	11317	44818	98127
RAIL2586	923269	2586	8011362	2586	6235	24447	495112	51833	9613272	1503568
RAIL4284	1096894	4284	11284032	4284	6345	45055	958072	89219	15880038	6776270
SPAL_004	321696	10203	46168124	10203	18683	>6911	20410	213972	40754070	46763011
STAT96V2	957432	29089	2852184	29089	38807	81055	12573814	276290	4210897	1702787
STAT96V3	1113780	33841	3317736	33841	45145	94273	14216653	319109	4901099	1977377
STAT96V4	63076	3173	491336	3173	11493	10513	415798	47295	512268	149650
STORMG21K	1377306	526185	3459881	526185	6933837	>1850507	526185	7505682	34433755	853778245
WATSON_1	386992	201155	1055093	201155	1005650	519977	11469139	2634818	6106788	11698276
WATSON_2	677224	352013	1846391	352013	2808070	1171936	17251085	4648375	10588235	17953039
WORLD	67147	34506	198883	34506	535778	224464	20065186	511951	1288095	4555752

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

name	m	n	$nz(A)$	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN	MA87
12month1	872622	12471	22624727	12471	73375	>64026	764511	260889	34096963	72985572
162bit	3606	3476	37118	3476	193542	28550	1215038	70512	166083	2844518
176bit	7441	7150	82270	7150	388870	56269	5121390	145649	349258	10258362
192bit	13691	13093	154303	13093	685467	96275	14109093	267084	654954	29643489
208bit	24430	23191	299756	23191	1202798	168273	24217191	471420	1177025	85199435
beaflw	500	492	53403	492	28073	5307	452608	9814	71656	114801
c8_mat11	5761	4562	2462970	4562	78255	49640	4562	95477	2621770	10254646
connectus	394707	458	1127525	458	478	1744	1850	5185	1527818	29414
ESOC	327062	37349	6019939	37349	698366	>265992	37349	776570	12993333	37844216
EternityII_Etilde	204304	10054	1170516	10054	24368	103380	4019398	205375	4202504	5074041
f855_mat9	2511	2456	171214	2456	136743	26961	2456	51366	264054	2893837
GL7d16	955127	460260	14488881	460260	9346036	>320334	4602708	9665201	36508148	-
GL7d17	1548649	955127	25978098	955127	-	>328160	8596231	20057165	-	-
GL7d18	1955309	1548645	35590540	1548645	-	>352792	17035225	32520973	-	-
GL7d19	1955296	1911130	37322725	1911130	-	>550233	19111408	40133312	-	-
GL7d20	1911124	1437546	29893084	1437546	-	>527911	8625316	30188087	-	-
GL7d21	1437546	822922	18174775	822922	-	>526636	5760508	17281075	-	-
GL7d22	822906	349443	8251000	349443	12616485	>535015	2446155	7337962	30115416	-
GL7d23	349443	105054	2695430	105054	1721600	>558111	735432	2205827	10424076	-
graphics	29493	11822	117954	11822	31960	30470	11822	24901	194127	466587
HFE18_96_in	2372	2371	933343	2371	10978	26016	3376371	49576	1028976	2810425
IG5-15	11369	6146	323509	6146	189538	62626	580318	128833	643103	13575427
IG5-16	18846	9519	588326	9519	279931	97466	1470142	199678	1115361	31235491
IG5-17	30162	14060	1035008	14060	409603	145262	1758938	295028	1843157	69381526
IG5-18	47894	20818	1790490	20818	587680	216979	2260114	436954	2992054	155805819
IMDB	896302	303617	3782463	303617	15909318	>487530	303617	5858164	13438747	>4216225900
kneser_10_4_1	349651	330751	992252	330751	6394519	-	330751	6759341	11405272	362087118
landmark	71952	2673	1146848	2673	11307	17778	75654	26909	873043	378177
LargeRegFile	2111154	801374	4944201	801374	3615761	>461966	6411062	4106048	24923112	15941094
Maragal_6	21251	10144	537694	10144	246737	71907	5289072	212144	679769	50574842
Maragal_7	46845	26525	1200537	26525	662167	166881	5156765	553856	1641051	139030659
Maragal_8	60845	33093	1308415	33093	1702715	229119	33093	597971	1392093	88830058
mri1	114637	65536	589824	65536	519051	325965	62016508	636290	1776829	8157680
mri2	104597	63240	569160	63240	1486341	392461	47669568	781491	2575072	35347277
NotreDame_actors	383640	127823	1470404	127823	6768455	>708376	127823	2506265	6151175	1179796747
psse0	26722	11028	102432	11028	31603	23833	12042028	35197	180414	371305
psse1	14318	11028	57376	11028	40288	28925	12042028	35771	133438	381958
psse2	28634	11028	115262	11028	38437	34657	11028	40814	216020	394467
rel9	5921786	274667	23667183	274667	498966	>313371	3570851	5764774	-	-
relat9	9746232	274667	38955420	274667	337475	>263449	3845546	5763434	-	-
Rucci1	1977885	109900	7791168	109900	638743	932200	109900	2306811	19056222	136755107
sls	1748122	62729	6804304	62729	71400	108831	3013342	1226997	13636543	110829629
TF14	3159	2644	29862	2644	150764	28727	3649644	55249	148306	2209059
TF15	7741	6334	80057	6334	379663	69250	6334	132747	368065	11277723
TF16	19320	15437	216173	15437	933646	169320	15437	323886	928743	64337506
TF17	48629	38132	586218	38132	2304372	418887	38132	800458	2359904	360007876
TF18	123867	95368	1597545	95368	5699962	>808291	95368	2002386	6080905	>2147378630
TF19	317955	241029	4370721	241029	14260905	>811388	241029	5061262	15718547	>8629432746
tomographic1	59360	45908	647495	45908	988212	280726	45908	906437	2247216	31002372
Trec14	15904	3159	2872265	3159	11942	34692	2661039	66099	3251107	4909829
wheel_601	902103	723605	2170814	723605	7796970	4253762	723605	14201674	25831230	-

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	14838	7625	48418	>741606	4406	7977	11618
BAXTER	30733	27441	111576	172315	>940092	101	>875045	>18992	>322351	>144158
BCDOUT	7078	5412	67344	129755	122022	>452954	>1774726	>73955	142047	154786
C09	22924	10789	109651	24953	5078	3612	6990	2619	379	194
CONT11_L	1961394	1468599	5382999	206	206	-	60	101	22	19
DBIR1	45775	18804	1077025	1450	2228	31060	>224685	367	1858	>59884
DBIR2	45877	18906	1158159	19116	2211	33561	>400186	864	793	>60617
D2Q06C	5831	2171	33081	58912	1597	472	12219	284	209	18
DELF000	5543	3128	13741	298427	26446	691	30615	4665	60	58
GE	16369	10099	44825	69740	6244	573	832	799	28	81
LARGE001	7176	4162	18887	52556	26777	53702	2233527	7422	75	90
LPL1	129959	39951	386218	31755	3217	564	>171967	712	420	66
MOD2	66409	34774	199810	10674	1370	1515	46985	579	151	88
MODEL10	16819	4400	150372	34343	2229	5899	131196	388	744	202
MPSBCD03	7078	5412	66210	153952	149821	>450987	>1782418	>70943	145960	172717
NSCT2	37563	23003	697738	10090	1397	13476	>645872	205	610	>88277
NSIR2	10057	4453	154939	9622	1033	20198	>1646173	210	388	76281
PDE1	271792	270595	990587	903	943	-	-	>2291	-	-
PDS-100	514577	156016	1096002	682	342	228	203	76	90	64
PDS-90	475448	142596	1014136	639	331	216	195	73	88	74
PILOT-JA	2267	940	14977	137401	2346	61	26428	334	323	54
PILOTNOV	2446	975	13331	83019	1931	41	18021	340	214	20
RAIL2586	923269	2586	8011362	912	400	809	233	178	151	9
RAIL4284	1096894	4284	11284032	883	732	913	374	212	224	19
SPAL_004	321696	10203	46168124	>5212	3258	>5630	-	1	3764	>1397
STAT96V2	957432	29089	2852184	985	726	464	414	425	19	22
STAT96V3	1113780	33841	3317736	1054	765	484	433	414	20	27
STAT96V4	63076	3173	491336	4087	809	1757	449	125	17	24
STORMG21K	1377306	526185	3459881	1401	183	>7602	-	>919	2281	>3327
WATSON_1	386992	201155	1055093	2161	422	165	249	56	73	8
WATSON_2	677224	352013	1846391	1812	349	119	185	48	54	7
WORLD	67147	34506	198883	9814	1369	1084	24839	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	>9415	266	973	-	60	369	293
162bit	3606	3476	37118	29440	2540	729	1493	319	247	252
176bit	7441	7150	82270	147449	6538	2697	3892	655	454	981
192bit	13691	13093	154303	326126	12200	4800	7238	6005	1281	1519
208bit	24430	23191	299756	606439	17074	9741	13993	6219	2197	3132
beaflw	500	492	53403	43174	41103	38035	>5	469	33910	>2951677
c8_mat11	5761	4562	2462970	40930	38679	>176239	>193279	>7647	30090	>67344
connectus	394707	458	1127525	1753	7	114	7	3	6	3
ESOC	327062	37349	6019939	5596	15011	>20554	-	>2671	>21840	>9045
EternityII_Etilde	204304	10054	1170516	1356	1122	2098	883	384	585	81
f855_mat9	2511	2456	171214	19082	20175	267667	321572	>72874	12324	>832602
GL7d16	955127	460260	14488881	61	48	265	-	9	32	35
GL7d17	1548649	955127	25978098	58	48	-	-	8	28	-
GL7d18	1955309	1548645	35590540	79	64	-	-	10	40	-
GL7d19	1955296	1911130	37322725	204	53	-	-	9	46	-
GL7d20	1911124	1437546	29893084	137	31	-	-	5	28	-
GL7d21	1437546	822922	18174775	143	26	-	-	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	>1623343	302842	312745	177862	>44385	1908	221
HFE18_96_in	2372	2371	933343	30425	15101	30092	12419	1632	14635	16052
IG5-15	11369	6146	323509	4567	610	126	323	92	240	533
IG5-16	18846	9519	588326	7421	866	150	478	151	347	771
IG5-17	30162	14060	1035008	7243	828	169	411	123	329	705
IG5-18	47894	20818	1790490	7281	735	205	446	107	312	889
IMDB	896302	303617	3782463	>15970	>15288	>3043	-	>899	>8031	>3368
kneser_10_4_1	349651	330751	992252	17207	10782	>9032	-	>1755	3258	1866
landmark	71952	2673	1146848	19557	894	36	274	27	12	25
LargeRegFile	2111154	801374	4944201	784	54	168	-	7	12	21
Maragal_6	21251	10144	537694	5400	3254	10031	>579407	496	1016	1756
Maragal_7	46845	26525	1200537	7469	2683	5150	>1913	192	680	1112
Maragal_8	60845	33093	1308415	>134737	>131006	>43588	>103951	>8548	>91721	>52936
mri1	114637	65536	589824	6150	6103	8744	1603	932	2217	65
mri2	104597	63240	569160	11848	11853	4126	>1	744	2933	10317
NotreDame_actors	383640	127823	1470404	>64753	>63129	>8938	-	>3314	>26248	>9268
psse0	26722	11028	102432	190067	41438	1564	20222	40001	104	26
psse1	14318	11028	57376	190737	58420	5113	>1072010	51952	665	138
psse2	28634	11028	115262	191582	51664	7598	>887601	>51962	687	161
rel9	5921786	274667	23667183	110	81	107	-	12	37	-
relat9	9746232	274667	38955420	88	76	82	-	13	36	-
Rucci1	1977885	109900	7791168	>13607	8330	>13414	1822	>1813	>12647	555
sls	1748122	62729	6804304	610	188	619	-	47	68	13
TF14	3159	2644	29862	34774	25705	44459	12884	1758	11389	1091
TF15	7741	6334	80057	107320	81868	246138	43583	>63793	41012	1723
TF16	19320	15437	216173	323986	227796	>107703	142169	>29019	144257	1102
TF17	48629	38132	586218	>440453	>465868	>43370	>182890	>10465	>145969	1212
TF18	123867	95368	1597545	>146295	>130859	>17063	-	>3960	>50392	1469
TF19	317955	241029	4370721	>42857	>44132	>5160	-	>1478	>14317	1086
tomographic1	59360	45908	647495	65478	18912	>75149	>157297	>8915	1867	1837
Trec14	15904	3159	2872265	2005	1600	8542	>1	690	1598	16105
wheel_601	902103	723605	2170814	>21448	>21212	>4579	>7467	>762	>6083	>3554

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

name	m	n	$nz(A)$	diagonal	MIQR	RIF	MI35	MI30-MIN	MA87
BAS1LP	9825	5411	587775	0.00	0.66	7.30	1.35	2.07	0.95
BAXTER	30733	27441	111576	0.00	0.31	1.34	0.28	0.98	0.38
BCDOUT	7078	5412	67344	0.00	0.40	1.51	0.46	0.29	0.39
C09	22924	10789	109651	0.00	0.17	1.06	0.21	0.99	0.10
CONT11_L	1961394	1468599	5382999	0.02	>15.95	45.92	9.15	20.14	8.38
DBIR1	45775	18804	1077025	0.00	0.82	42.48	2.14	5.12	0.99
DBIR2	45877	18906	1158159	0.00	0.84	49.64	1.90	4.38	1.11
D2Q06C	5831	2171	33081	0.00	0.03	0.06	0.03	0.17	0.03
DELF000	5543	3128	13741	0.00	0.01	0.01	0.04	0.01	0.02
GE	16369	10099	44825	0.00	0.06	0.17	0.08	0.39	0.06
LARGE001	7176	4162	18887	0.00	0.01	0.01	0.01	0.02	0.02
LPL1	129959	39951	386218	0.00	0.26	0.44	0.92	3.59	0.38
MOD2	66409	34774	199810	0.00	0.43	2.23	0.98	1.31	0.29
MODEL10	16819	4400	150372	0.00	0.20	0.40	0.09	0.53	0.08
MPSBCD03	7078	5412	66210	0.00	0.39	1.57	0.42	0.37	0.40
NSCT2	37563	23003	697738	0.00	1.42	38.78	2.51	4.21	2.05
NSIR2	10057	4453	154939	0.00	0.21	2.49	0.21	0.18	0.15
PDE1	271792	270595	990587	0.00	>591.48	>600.02	>0.01	>600.82	>0.01
PDS-100	514577	156016	1096002	0.00	1.17	22.99	1.55	13.34	2.53
PDS-90	475448	142596	1014136	0.00	1.07	21.69	1.43	12.40	2.40
PILOT-JA	2267	940	14977	0.00	0.02	0.04	0.02	0.02	0.02
PILOTNOV	2446	975	13331	0.00	0.02	0.03	0.02	0.02	0.02
RAIL2586	923269	2586	8011362	0.01	2.33	110.81	2.17	3.83	1.25
RAIL4284	1096894	4284	11284032	0.02	4.75	352.77	4.18	11.91	2.46
SPAL_004	321696	10203	46168124	0.08	9.61	>601.39	71.76	43.20	73.46
STAT96V2	957432	29089	2852184	0.01	0.15	0.65	0.27	1.12	0.26
STAT96V3	1113780	33841	3317736	0.01	0.17	0.78	0.32	1.32	0.37
STAT96V4	63076	3173	491336	0.00	0.04	2.36	0.08	0.22	0.05
STORMG21K	1377306	526185	3459881	0.01	42.59	>600.04	30.21	62.51	113.37
WATSON_1	386992	201155	1055093	0.00	1.69	0.53	1.48	1.02	1.15
WATSON_2	677224	352013	1846391	0.01	5.04	4.22	2.85	1.97	2.21
WORLD	67147	34506	198883	0.00	0.41	2.17	0.74	3.73	0.29

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

name	m	n	$nz(A)$	diagonal	MIQR	RIF	MI35	MI30-MIN	MA87
12month1	872622	12471	22624727	0.04	15.03	>600.67	170.63	363.98	117.64
162bit	3606	3476	37118	0.00	0.43	1.80	0.13	0.17	0.14
176bit	7441	7150	82270	0.00	1.16	8.51	0.28	0.47	0.64
192bit	13691	13093	154303	0.00	2.40	30.75	0.46	0.75	2.54
208bit	24430	23191	299756	0.00	4.77	109.27	0.98	1.91	11.70
beaflw	500	492	53403	0.00	0.06	0.18	0.09	0.12	0.07
c8_mat11	5761	4562	2462970	0.01	4.37	66.65	30.65	8.03	23.21
connectus	394707	458	1127525	0.00	0.11	2.32	0.22	0.60	0.17
ESOC	327062	37349	6019939	0.01	6.34	>600.14	3.35	340.01	3.94
EternityII_Etilde	204304	10054	1170516	0.00	0.63	38.64	1.14	2.25	0.47
f855_mat9	2511	2456	171214	0.00	0.63	3.01	0.88	0.55	0.74
GL7d16	955127	460260	14488881	0.03	278.05	>600.74	74.74	229.19	>600.86
GL7d17	1548649	955127	25978098	0.06	>600.05	>601.73	311.42	>600.18	>600.11
GL7d18	1955309	1548645	35590540	0.08	>600.00	>602.63	>600.40	>600.43	>600.59
GL7d19	1955296	1911130	37322725	0.08	>600.32	>602.93	>600.46	>600.00	>600.45
GL7d20	1911124	1437546	29893084	0.07	>600.27	>602.04	412.34	>600.59	>600.22
GL7d21	1437546	822922	18174775	0.04	>600.30	>601.12	75.82	>600.79	>600.90
GL7d22	822906	349443	8251000	0.02	167.10	>600.39	22.34	198.70	>600.65
GL7d23	349443	105054	2695430	0.01	23.48	>600.09	5.26	98.90	>600.43
graphics	29493	11822	117954	0.00	0.09	0.17	0.04	0.08	0.06
HFE18_96_in	2372	2371	933343	0.00	1.13	14.67	5.80	0.76	3.31
IG5-15	11369	6146	323509	0.00	0.99	9.91	0.70	1.98	1.50
IG5-16	18846	9519	588326	0.00	1.84	27.11	1.68	1.59	3.75
IG5-17	30162	14060	1035008	0.00	3.26	70.73	2.72	7.52	11.09
IG5-18	47894	20818	1790490	0.00	5.88	182.75	6.82	10.52	38.25
IMDB	896302	303617	3782463	0.01	51.43	>600.23	40.43	131.77	>205.71
kneser_10_4_1	349651	330751	992252	0.00	9.22	>20.61	5.69	15.88	31.95
landmark	71952	2673	1146848	0.00	0.27	1.12	0.28	0.52	0.18
LargeRegFile	2111154	801374	4944201	0.01	10.46	>600.08	2.50	57.99	127.40
Maragal_6	21251	10144	537694	0.00	2.41	41.79	13.15	2.31	17.11
Maragal_7	46845	26525	1200537	0.00	8.64	145.90	45.06	8.91	56.28
Maragal_8	60845	33093	1308415	0.00	16.17	205.65	5.18	3.10	14.09
mri1	114637	65536	589824	0.00	1.25	95.20	2.23	2.07	0.53
mri2	104597	63240	569160	0.00	4.07	68.05	4.84	6.73	3.98
NotreDame_actors	383640	127823	1470404	0.01	11.45	>600.04	9.09	31.35	591.52
psse0	26722	11028	102432	0.00	0.07	0.04	0.02	0.07	0.14
psse1	14318	11028	57376	0.00	0.09	0.05	0.03	0.03	0.03
psse2	28634	11028	115262	0.00	0.08	0.13	0.04	0.06	0.04
rel9	5921786	274667	23667183	0.04	15.81	>601.16	56.51	>600.26	>600.42
relat9	9746232	274667	38955420	0.07	14.14	>602.27	62.22	>600.49	>600.26
Rucci1	1977885	109900	7791168	0.01	2.40	179.08	2.55	39.35	7.97
sls	1748122	62729	6804304	0.01	1.60	>601.94	6.62	14.26	24.23
TF14	3159	2644	29862	0.00	0.23	0.43	0.06	0.21	0.11
TF15	7741	6334	80057	0.00	0.76	2.66	0.20	0.46	0.59
TF16	19320	15437	216173	0.00	2.47	17.64	0.45	1.17	5.74
TF17	48629	38132	586218	0.00	7.66	120.76	1.21	4.23	77.31
TF18	123867	95368	1597545	0.00	24.45	>600.03	3.50	13.48	>21.69
TF19	317955	241029	4370721	0.01	84.67	>600.09	10.60	40.42	>224.19
tomographic1	59360	45908	647495	0.00	2.92	3.12	1.46	1.61	1.52
Trec14	15904	3159	2872265	0.00	2.39	58.89	26.88	11.27	19.05
wheel_601	902103	723605	2170814	0.01	20.63	39.95	30.90	49.67	>600.25

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	MA87
BAS1LP	9825	5411	587775	8.37	5.80	74.22	>600.33	92.27	8.77	49.65	0.99
BAXTER	30733	27441	111576	73.77	>600.00	0.51	>601.34	>600.01	>600.28	>600.33	0.40
BCDOUT	7078	5412	67344	15.18	18.57	>600.40	>601.51	>600.00	64.14	142.05	0.40
CO9	22924	10789	109651	6.59	1.69	4.68	5.62	32.24	0.53	1.53	0.10
CONT11_L	1961394	1468599	5382999	9.58	11.24	-	51.37	39.39	11.83	25.45	9.65
DBIR1	45775	18804	1077025	2.91	4.87	131.85	>600.49	8.02	8.02	>600.27	1.03
DBIR2	45877	18906	1158159	43.93	5.16	148.55	>600.67	26.50	4.40	>600.46	1.15
D2Q06C	5831	2171	33081	3.70	0.12	0.16	2.03	0.48	0.07	0.20	0.03
DELF000	5543	3128	13741	16.12	1.98	0.14	4.38	27.21	0.05	0.03	0.03
GE	16369	10099	44825	11.81	1.46	0.60	0.61	7.31	0.11	0.54	0.06
LARGE001	7176	4162	18887	3.65	2.51	13.35	427.11	38.54	0.03	0.06	0.03
LPL1	129959	39951	386218	62.09	6.99	2.76	>600.45	30.19	2.85	4.46	0.41
MOD2	66409	34774	199810	7.86	1.28	7.44	99.95	14.93	1.44	2.06	0.31
MODEL10	16819	4400	150372	6.23	0.48	3.06	47.21	1.55	0.44	0.87	0.08
MPSBCD03	7078	5412	66210	17.87	22.69	>600.39	>601.57	>600.00	65.92	158.56	0.41
NSCT2	37563	23003	697738	12.03	1.88	67.30	>600.79	2.90	3.84	>600.28	2.10
NSIR2	10057	4453	154939	2.19	0.17	16.27	>600.50	0.66	0.38	104.00	0.15
PDE1	271792	270595	990587	5.79	7.59	-	-	>600.05	-	-	-
PDS-100	514577	156016	1096002	7.75	3.88	5.51	25.92	3.82	3.53	17.60	2.73
PDS-90	475448	142596	1014136	6.45	3.56	4.80	24.24	3.47	3.12	16.83	2.57
PILOT-JA	2267	940	14977	4.00	0.08	0.04	1.71	0.17	0.08	0.04	0.02
PILOTNOV	2446	975	13331	2.35	0.07	0.03	1.17	0.14	0.04	0.03	0.02
RAIL2586	923269	2586	8011362	31.83	14.22	25.87	117.90	47.76	6.58	4.79	1.54
RAIL4284	1096894	4284	11284032	41.94	35.47	44.83	369.62	88.69	14.09	14.59	2.91
SPAL_004	321696	10203	46168124	>600.18	374.89	>600.81	-	4.76	466.94	>600.72	74.99
STAT96V2	957432	29089	2852184	12.98	9.55	6.27	6.09	16.31	0.55	2.39	0.35
STAT96V3	1113780	33841	3317736	16.34	11.98	7.63	7.46	18.75	0.65	3.13	0.48
STAT96V4	63076	3173	491336	3.48	0.69	1.64	2.75	1.53	0.10	0.32	0.06
STORMG21K	1377306	526185	3459881	33.31	5.09	>600.67	-	>600.94	167.08	>600.67	115.19
WATSON_1	386992	201155	1055093	14.62	3.25	4.28	3.27	3.77	2.83	1.40	1.25
WATSON_2	677224	352013	1846391	22.86	5.03	9.25	8.22	5.98	4.67	2.59	2.38
WORLD	67147	34506	198883	6.57	1.24	5.23	54.05	14.52	1.24	4.33	0.31

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	MA87
12month1	872622	12471	22624727	>600.12	17.11	73.17	-	29.84	192.66	419.45	118.47
162bit	3606	3476	37118	2.77	0.33	1.24	2.18	0.67	0.18	0.34	0.15
176bit	7441	7150	82270	32.68	1.77	7.35	10.64	4.98	0.60	1.88	0.66
192bit	13691	13093	154303	145.33	6.80	23.16	38.23	107.45	2.19	5.50	2.61
208bit	24430	23191	299756	566.32	20.49	83.52	137.37	215.02	7.12	23.16	11.88
beaflw	500	492	53403	1.83	1.85	7.26	>0.18	0.50	2.61	>600.61	0.08
c8_mat11	5761	4562	2462970	123.49	116.55	>600.38	>600.66	>600.03	125.79	>600.84	23.29
connectus	394707	458	1127525	10.86	0.05	0.75	2.37	0.16	0.26	0.71	0.20
ESOC	327062	37349	6019939	152.70	398.52	>600.39	-	>600.23	>603.38	>600.05	4.21
EternityII_Etilde	204304	10054	1170516	6.88	5.81	10.79	43.13	7.94	4.16	4.36	0.52
f855_mat9	2511	2456	171214	2.08	2.54	219.75	77.67	>600.01	4.25	>600.44	0.75
GL7d16	955127	460260	14488881	7.11	5.82	333.57	-	7.04	80.55	246.11	-
GL7d17	1548649	955127	25978098	14.63	12.47	-	-	15.01	323.25	-	-
GL7d18	1955309	1548645	35590540	30.68	25.32	-	-	26.39	-	-	-
GL7d19	1955296	1911130	37322725	79.19	21.12	-	-	38.93	-	-	-
GL7d20	1911124	1437546	29893084	43.87	10.14	-	-	23.70	428.69	-	-
GL7d21	1437546	822922	18174775	24.30	4.82	-	-	10.40	83.70	-	-
GL7d22	822906	349443	8251000	14.51	1.63	191.82	-	3.51	25.00	210.17	-
GL7d23	349443	105054	2695430	4.70	0.43	27.15	-	0.97	5.88	101.50	-
graphics	29493	11822	117954	>600.00	131.90	266.73	117.27	>600.00	1.42	0.51	0.07
HFE18_96_in	2372	2371	933343	28.72	12.30	31.62	25.68	51.44	19.44	54.85	3.34
IG5-15	11369	6146	323509	2.51	0.31	1.19	10.19	0.88	0.91	3.44	1.54
IG5-16	18846	9519	588326	8.14	0.96	2.27	27.85	2.64	2.27	5.54	3.82
IG5-17	30162	14060	1035008	19.59	1.92	4.11	71.91	3.90	3.81	14.00	11.25
IG5-18	47894	20818	1790490	47.51	4.77	7.96	185.81	8.09	9.31	28.20	38.58
IMDB	896302	303617	3782463	>600.08	>600.06	>600.69	-	>600.35	>640.51	>600.87	-
kneser_10_4_1	349651	330751	992252	162.24	120.85	>600.28	-	>600.14	138.66	152.51	32.83
landmark	71952	2673	1146848	33.28	1.67	0.34	1.58	1.85	0.31	0.69	0.21
LargeRegFile	2111154	801374	4944201	42.50	3.00	24.82	-	3.08	3.66	67.27	127.83
Maragal_6	21251	10144	537694	3.78	2.11	24.19	>600.80	4.42	14.48	7.85	17.22
Maragal_7	46845	26525	1200537	16.32	6.08	41.25	>151.90	5.28	47.83	18.50	56.57
Maragal_8	60845	33093	1308415	>600.01	>600.01	>600.18	>600.66	>600.07	>605.19	>600.18	14.30
mri1	114637	65536	589824	13.39	15.33	60.08	102.66	68.69	16.27	2.99	0.58
mri2	104597	63240	569160	20.90	24.62	49.53	>68.07	43.48	22.23	156.20	4.07
NotreDame_actors	383640	127823	1470404	>600.02	>600.02	>600.52	-	>600.05	>609.12	>600.39	593.90
psse0	26722	11028	102432	49.79	13.48	1.28	11.99	358.86	0.09	0.12	0.14
psse1	14318	11028	57376	38.77	15.38	3.96	>600.05	509.79	0.41	0.19	0.04
psse2	28634	11028	115262	56.29	17.98	6.44	>600.13	>600.01	0.49	0.37	0.05
rel9	5921786	274667	23667183	25.00	16.49	38.17	-	24.04	66.04	-	-
relat9	9746232	274667	38955420	33.85	26.92	42.44	-	39.81	76.63	-	-
Rucci1	1977885	109900	7791168	>600.05	343.67	>600.47	261.20	>600.42	>602.63	133.04	8.51
sls	1748122	62729	6804304	30.31	8.59	30.25	-	9.41	10.11	16.99	24.94
TF14	3159	2644	29862	1.84	1.83	36.66	2.71	7.02	2.53	0.76	0.11
TF15	7741	6334	80057	14.57	14.46	513.65	21.79	>600.01	21.91	2.64	0.62
TF16	19320	15437	216173	110.15	99.55	>600.48	172.42	>600.00	195.85	5.27	5.87
TF17	48629	38132	586218	>600.00	>600.01	>600.68	>600.77	>600.01	>601.21	17.34	77.98
TF18	123867	95368	1597545	>600.01	>600.01	>600.49	-	>600.01	>603.52	70.49	-
TF19	317955	241029	4370721	>600.03	>600.03	>600.75	-	>600.03	>610.63	179.21	-
tomographic1	59360	45908	647495	119.00	33.46	>600.93	>600.18	>600.10	11.05	22.58	1.61
Trec14	15904	3159	2872265	7.24	5.53	32.63	>58.91	44.57	32.66	203.92	19.13
wheel_601	902103	723605	2170814	>600.05	>600.06	>600.77	>600.07	>600.40	>631.06	>600.76	-

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

name	no	diagonal	MIQR	RIF	BA-G	MI35	MI30-MIN	MA87	MA97	SPQR
BAS1LP	5.54E+1	5.54E+1	5.54E+1	-	5.54E+1	5.54E+1	5.54E+1	5.54E+1	5.54E+1	5.54E+1
BAXTER	9.76E+1	-	9.18E+1	-	-	-	-	5.93E+1	6.50E+1	5.92E+1
BCDOUT	3.54E+1	3.53E+1	-	-	-	3.53E+1	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	1.67E+2	1.66E+2	1.66E+2	-	1.66E+2	1.67E+2	-	1.66E+2	1.66E+2	1.66E+2
DBIR2	1.67E+2	1.66E+2	1.66E+2	-	1.66E+2	1.67E+2	-	1.66E+2	1.66E+2	1.66E+2
D2Q06C	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
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	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	6.06E+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	5.35E+1	5.36E+1	5.35E+1	5.35E+1	5.35E+1
MPSBCD03	3.53E+1	3.53E+1	-	-	-	3.53E+1	3.53E+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	-	1.83E+2	1.83E+2	1.83E+2
NSIR2	8.05E+1	8.04E+1	8.04E+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	3.32E+1	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	-	5.65E-6	-	-	3.24E-11	5.65E-6	-	5.31E-8	-	-
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	-	-
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	MA87	MA97	SPQR
12month1	-	9.27E+2	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	3.86E+1	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	-	4.33E+0	4.68E+0	-	4.37E+0	4.52E+0	4.16E+0
c8_mat11	2.21E+1	2.19E+1	-	-	-	2.19E+1	-	2.12E+1	2.14E+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	-	-	-	-	-	3.64E-1	1.51E+1	5.97E-9
EternityII_Etilde	4.47E-6	4.43E-6	4.37E-6	4.47E-6	5.33E-6	4.33E-6	1.19E-4	1.03E-7	1.03E-5	1.08E-12
f855_mat9	1.79E+1	1.79E+1	1.75E+1	1.67E+1	-	1.79E+1	-	1.34E+1	1.60E+1	1.90E+4
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	-	-	-	-
GL7d18	9.31E+2	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	-	-	-	-
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	3.55E+0	3.93E+1	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	-	-	-	-	-	-	-	-	-	-
kneser_10_4_1	1.62E+2	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	8.59E-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	9.39E+1	9.39E+1	9.38E+1	-	9.38E+1	9.39E+1	9.39E+1	9.38E+1	9.38E+1	9.48E+1
Maragal_7	1.33E+2	1.33E+2	1.33E+2	-	1.33E+2	1.33E+2	1.33E+2	1.33E+2	1.33E+2	1.34E+2
Maragal_8	-	-	-	-	-	-	-	2.38E+2	2.38E+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	3.56E+13
mri2	1.41E+2	1.41E+2	1.41E+2	-	1.41E+2	1.41E+2	1.41E+2	1.41E+2	-	1.25E+24
NotreDame_actors	-	-	-	-	-	-	-	5.18E+2	-	-
psse0	1.62E+2	1.62E+2	1.62E+2	1.62E+2	1.62E+2	1.62E+2	1.63E+2	1.62E+2	1.62E+2	1.62E+2
psse1	5.44E+1	5.43E+1	5.43E+1	-	5.43E+1	5.43E+1	5.44E+1	5.43E+1	5.43E+1	5.43E+1
psse2	1.65E+2	1.65E+2	1.65E+2	-	-	1.65E+2	1.66E+2	1.65E+2	1.65E+2	1.65E+2
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	-	-	7.28E+2	7.27E+2	7.27E+2	7.27E+2
sls	1.29E-5	1.23E-5	1.28E-5	1.18E-5	2.09E-4	1.01E-5	1.73E-5	1.06E-7	1.06E-5	-
TF14	5.57E-7	5.61E-7	5.50E-7	5.41E-7	5.37E-9	5.61E-7	6.23E-3	6.34E-8	6.29E-6	5.63E-14
TF15	8.78E-7	8.77E-7	8.79E-7	8.69E-7	-	8.77E-7	1.13E-2	2.40E-7	2.28E-5	1.35E-13
TF16	1.38E-6	1.38E-6	-	1.38E-6	-	1.38E-6	1.52E-2	9.42E-7	6.96E-5	3.16E-13
TF17	-	-	-	-	-	-	2.14E-2	3.66E-6	-	-
TF18	-	-	-	-	-	-	3.19E-2	-	-	-
TF19	-	-	-	-	-	-	4.26E-2	-	-	-
tomographic1	4.20E+1	4.19E+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	-	-	-	-	-	-	-	-	4.22E+2	-