Science and Technology Facilities Council

Preprint
RAL-P-2020-004

A computational study of using black-box QR solvers for large-scale sparse-dense linear least squares problems

J Scott, M Tuma

October 2020

Submitted for publication in ACM Transactions on Mathematical Software

ISSN 1361-4762

A COMPUTATIONAL STUDY OF USING BLACK-BOX QR SOLVERS FOR LARGE-SCALE SPARSE-DENSE LINEAR LEAST SQUARES PROBLEMS

JENNIFER SCOTT* AND MIROSLAV TŮMA†

Abstract. Large-scale overdetermined linear least squares problems arise in many practical applications, both as subproblems of nonlinear least squares problems and in their own right. One popular solution method is based on the backward stable QR factorization of the system matrix A. This paper focuses on sparse-dense linear least squares problems, that is, problems where A is sparse except from a small number of rows that are considered to be dense. For large-scale problems, the direct application of a QR solver will fail because of a lack of memory or will be unacceptably slow. We study a number of approaches for solving such problems using a sparse QR solver without modification. We consider the case where the sparse part of A is rank-deficient and show that either preprocessing A using partial matrix stretching or using regularization and employing a direct-iterative approach can be seamlessly combined with a black-box QR solver. Furthermore, we propose extending the augmented system formulation with iterative refinement for sparse problems to sparse-dense problems, and demonstrate experimentally that multi-precision variants can be successfully used.

1. Introduction.

In recent years, there has been renewed interest in the development of efficient and reliable software packages for the solution of large sparse least squares (LS) problems using the QR algorithm [12, 15]. Although these are general-purpose packages that may be used as "black-box" solvers, they do not effectively tackle the not uncommon case of the system matrix containing a (small) number of dense rows (here a row is considered to be dense if it has significantly more entries than the other rows or leads to a large amount of fill in the R factor but it is not necessarily full). These rows can result in sparse QR solvers failing because of either a lack of memory or being unacceptably slow; numerical results included in the study of least squares solution techniques by Gould and Scott [22, 23] demonstrate this.

To introduce our notation, we assume throughout that the rows of the system matrix A that are to be treated as dense have been permuted to the end. With a conformal partitioning of the vector b (and omitting the row permutation matrix for simplicity of notation) we have

$$A = \begin{bmatrix} A_s \\ A_d \end{bmatrix}, \; A_s \in \mathbb{R}^{m_s \times n}, \; A_d \in \mathbb{R}^{m_d \times n}, \; b = \begin{bmatrix} b_s \\ b_d \end{bmatrix}, \; b_s \in \mathbb{R}^{m_s}, \; b_d \in \mathbb{R}^{m_d}, \tag{1.1}$$

where m_s and m_d denote the number of sparse and dense rows of A, respectively, with $m = m_s + m_d$, $m_s \geq n$ and $m_d \geq 1$ is small ($m_d \ll m_s$). A_s and A_d are referred to as the sparse and dense row blocks of A and the rows of A_d are termed dense rows. The linear LS problem that we are interested in solving is then

$$\min_x \|Ax - b\|_2^2 = \min_x \left\| \begin{bmatrix} A_s \\ A_d \end{bmatrix} x - \begin{bmatrix} b_s \\ b_d \end{bmatrix} \right\|_2^2 . \tag{1.2}$$

We assume that A has full column rank, in which case the solution of (1.2) is unique and is given by the solution to the system of normal equations

$$Cx = A^T b, \qquad C = A^T A.$$

It is well understood that there are a number of possible problems associated with the normal equations. Firstly, there is a potential loss of information in explicitly computing the $n \times n$ symmetric positive definite normal matrix C and the vector $A^T b$. Secondly, if A contains just a single dense row then C is not sparse and thus, if n is large, it cannot be stored or factorized by a direct solver that computes a Cholesky factorization. Then there is the fact that the condition number of C is the square of that of A, so that

* STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, Oxfordshire, OX11 0QX, UK and School of Mathematical, Physical and Computational Sciences, University of Reading, Reading RG6 6AQ, UK. Correspondence to: jennifer.scott@stfc.ac.uk. Partially supported by EPSRC grant EP/M025179/1.

† Department of Numerical Mathematics, Faculty of Mathematics and Physics, Charles University, Czech Republic. mirektuma@karlin.mff.cuni.cz. Supported by project 17-04150J of the Grant Agency of the Czech Republic.

an accurate solution may be difficult to compute if A is poorly conditioned. If A is not full rank, the Cholesky factorization of C breaks down; near rank degeneracy causes similar numerical problems in finite precision arithmetic. One way to try and lessen the numerical issues is to avoid computing C and to obtain its Cholesky factor R directly from A by computing its QR factorization. An orthogonal matrix Q is computed such that

$$AP = Q \begin{bmatrix} R \\ 0 \end{bmatrix} \qquad \text{and} \qquad b = Q \begin{bmatrix} c \\ d \end{bmatrix}, \tag{1.3}$$

where $R \in \mathbb{R}^{n\times n}$ is upper triangular (and non singular if A is of full rank) and $P \in \mathbb{R}^{n\times n}$ is a permutation matrix that performs column interchanges to limit the fill in R. Since the Euclidean norm is invariant under orthogonal transformation, the solution to (1.2) may be obtained by solving the system

$$RP^T x = c. \tag{1.4}$$

Over the years, there has been significant work on QR factorizations for solving large sparse LS problems (see, for example, the book by Björck [11] and the references therein as well as [4, 12, 15, 44]). Unfortunately, for large problems of the form (1.2), a straightforward application of the QR algorithm will fail because, as already observed, the block A_d of dense rows causes the factor R to fill in, limiting the usefulness of black-box sparse QR solvers.

The difficulties that a (small) number of dense rows presents has been studied in the literature; see, for example, [2, 5, 11, 21, 25, 36, 37, 38, 39, 40, 41, 42]. The purpose of this paper is to consider how we can use a black-box QR package to solve such systems both efficiently and robustly. A number of different approaches are proposed and compared using problems from practical applications. A key challenge is that the sparse row block A_s is often rank deficient so that a QR package cannot be applied directly to it. In Section 2, we recall the use of updating [25] to handle dense rows. Then, in Section 3, we consider two preprocessing approaches that extend the applicability of updating when A_s is rank deficient: partial matrix stretching [39] and regularization [34]. Both avoid break down of the QR algorithm by enlarging the problem that it is applied to. In Section 4, we discuss using a QR factorization of the sparse row block combined with an iterative solver. Numerical results are presented in Section 5. These illustrate that updating works well if A_s is of full rank, offering significant savings compared to applying the QR solver with no special handling of dense rows. For rank-deficient A_s, we show that both preprocessing strategies are effective. If stretching is used, the solution of the original problem can be directly extracted while if regularization is used, it can be recovered by employing a preconditioned iterative solver. An alternative approach based on the augmented system formulation of the LS problem is considered in Section 6. We extend this approach to sparse-dense LS problems. This extension allows, in particular, the incorporation of iterative refinement with a preconditioned Krylov subspace solver. We demonstrate how this may be used, including with the incorporation of multi-precision arithmetic, which reduces the cost of the direct solver part of the solution process (in terms of memory and potentially time), while still returning the requested accuracy.

2. The updating approach.

Updating procedures are used in least squares applications when new observations are added to a previously solved problem; they can also handle dense rows by omitting such rows from the QR factorization and then updating the solution (but not the factorization) to incorporate the effects of the omitted rows. The approach is described in the 1982 paper of Heath [25]. Assume that the sparse block A_s is of full rank and start by computing the QR factorization

$$A_s P_s = Q_s \begin{bmatrix} R_s \\ 0 \end{bmatrix} \qquad \text{and} \qquad b_s = Q_s \begin{bmatrix} c_s \\ d_s \end{bmatrix}. \tag{2.1}$$

The solution y of the sparse LS problem

$$\min_y \|A_s y - b_s\|_2^2, \tag{2.2}$$

is then found by solving

$$R_s P_s^T y = c_s. \tag{2.3}$$

Let the solution of (1.2) be $x = y + z$; it remains to determine z. Using (2.1) and (2.3),

$$b_s - A_s x = b_s - A_s P_s (P_s^T y + P_s^T z) = Q_s \begin{bmatrix} c_s \\ d_s \end{bmatrix} - Q_s \begin{bmatrix} R_s \\ 0 \end{bmatrix} (P_s^T y + P_s^T z) = Q_s \begin{bmatrix} -R_s P_s^T z \\ d_s \end{bmatrix}.$$

Let $r_d = b_d - A_d y$. Then $b_d - A_d x = r_d - A_d z$ and we see that z is given by the solution of the LS problem

$$\min_z \left\| \begin{bmatrix} R_s P_s^T \\ A_d \end{bmatrix} z - \begin{bmatrix} 0 \\ r_d \end{bmatrix} \right\|_2^2. \tag{2.4}$$

Let $K_d^T \in \mathbb{R}^{n \times m_d}$ be the solution of the linear system $P_s R_s^T K_d^T = A_d^T$. Using the change of variables $u = R_s P_s^T z$ and $v = r_d - A_d z = r_d - K_d R_s P_s^T z = r_d - K_d u$, problem (2.4) becomes that of finding the minimum-norm solution of the underdetermined $m_d \times (n + m_d)$ system

$$\begin{bmatrix} K_d & I_{m_d} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = r_d. \tag{2.5}$$

Here, and elsewhere, for $k \geq 1$, I_k denotes the $k \times k$ identity matrix. This leads to Algorithm 1 for solving (1.2) (see Algorithm 3 of [25]).

Algorithm 1 QR with updating for solving the sparse-dense LS problem (1.2)

```
1: Compute the sparse QR factorization $A_s P_s = Q_s \begin{bmatrix} R_s \\ 0 \end{bmatrix}$ and set $\begin{bmatrix} c_s \\ d_s \end{bmatrix} = Q_s^T b_s$.
2: Solve $R_s P_s^T y = c_s$.
3: Form $r_d = b_d - A_d y$.
4: Solve $P_s R_s^T K_d^T = A_d^T$.
5: Compute the minimum norm solution of (2.5).
6: Solve $R_s P_s^T z = u$.
7: Set $x = y + z$.
```

The sparse triangular factor R_s is used to solve the large linear systems in this algorithm (Steps 2, 4 and 6); once (2.1) is performed, Q is not needed, unless there is a requirement to solve for further vectors b. Steps 3 and 4 involve dense linear algebra; in particular, the solution of (2.5) can be efficiently computed using the LAPACK routine `_getsls`.

3. Updating when A_s has some null columns.

In practice, A_s frequently has one or more null columns or is close to being rank deficient [39]. In this case, the QR factorization is backward stable but the computed R factor is ill conditioned. This usually leads to the computed LS solution having a very large norm. In our case, there will be problems with Steps 2, 4 and 6 of Algorithm 1, leading to an inaccurate x. This also happens if the R factor is used as a preconditioner for an iterative solver: the solver terminates after a very few iterations with a solution that has a huge norm. Thus we want to avoid ill conditioning in R. Avron, Ng and Toledo [5] propose a strategy involving adding singleton rows to the matrix A. They do this dynamically (during the factorization) and then, once the factorization is finished, a check is made of the conditioning of R and, if necessary, further rows are added and rotated into R using Givens rotations. Their aim is to ensure R is not ill conditioned while doing as few modifications as possible (a low rank modification). Because our objective is to use an existing QR package as a black box solver (without modification), we are unable to dynamically add rows during the factorization. We consider two alternative approaches for handling null columns in A_s that allow us to use an unmodified QR package: matrix stretching and regularization.

3.1. Sparse and partial matrix stretching. Matrix stretching aims to split each dense row (that is, each row of A_d) into a number of sparser rows and to formulate a (larger) modified problem from which the solution to the original LS problem can be derived. The idea was proposed by Grcar [24] and was subsequently used in a number of different contexts for solving linear systems [3, 6, 18, 19]. Stretching has also been applied to LS problems [1, 2]. Stretching treats the dense rows one by one. Standard stretching splits the row indices of the non zero entries in each dense row into sets of (almost) equal contiguous segments; an extra row and column is added to the matrix with entries corresponding to each of these sets. However, this splitting can result in significant fill in the normal matrix for the resulting stretched matrix (and also in its factors). Simply increasing the number of segments each dense row is split into does not necessarily alleviate the problem (the stretched system increases in size with the number of parts) and may adversely effect the conditioning. This led us to introduce a new approach, which we termed sparse stretching [38]. It aims to choose the splitting so as to limit the fill in the stretched normal matrix. It does this by considering the pattern of the normal matrix $A_s^T A_s$ corresponding to the sparse row block and, for each row in A_d, chooses the subsets of row indices so as to minimise the number of entries in the normal matrix for the stretched matrix. While numerical experiments have shown that sparse stretching successfully reduces the fill in compared to standard stretching, it can result in the stretched system being much larger than the original system (particularly in the case where A_s is highly sparse, for example, close to diagonal) and the cost of the factorization (in terms of time and memory) may still be prohibitive.

To circumvent this, we recently proposed partial stretching [39]. The idea here is to select a small subset of the rows of A_d that cause A_s to have null columns and to apply sparse stretching just to these rows, adding them to an enlarged sparse row block $\hat{A}_s$ and moving the remaining dense rows to a block $\hat{A}_d$ that has fewer rows than A_d. The result is a partially stretched matrix with no null columns that, in general, is smaller than would result from stretching all the dense rows. It may still contain some dense rows and these can be handled by applying the updating Algorithm 1 to the partially stretched problem. In the event that $\hat{A}_s$ is rank deficient (or highly ill conditioned) after partial stretching, further rows of A_d may be stretched before updating is employed.

3.2. Regularization. An alternative approach to handling null columns in A_s is to use regularization (see, for example, [34]). This increases the row dimension by replacing (1.2) with the regularized (or damped) LS problem

$$\min_x \|Ax - b\|_2^2 + \|\alpha x\|_2^2 = \min_x \left\| \begin{bmatrix} A_s \\ \alpha I_n \\ A_d \end{bmatrix} x - \begin{bmatrix} b_s \\ 0 \\ b_d \end{bmatrix} \right\|_2^2 = \min_x \left\| \tilde{A}x - \tilde{b} \right\|_2^2, \tag{3.1}$$

where $\alpha > 0$ is the regularization (or damping) parameter. A key advantage of regularization is that appending αI_n to A_s gives a full rank sparse block and thus Algorithm 1 can be successfully applied to the regularized problem. The regularized matrix $\tilde{A}$ is of size $(m_s + n) \times n$, but it involves only n additional entries compared to the original A. An important issue is how to choose the parameter α: too little regularization ("small" α) can lead to the QR factorization having numerical difficulties while for excessive regularization ("large" α), the computed objective value may be unacceptably different from the optimum for the original problem. Saunders [34] looks at linear programming problems (without dense rows) from the Netlib test set (`http://www.netlib.org/`). For these, provided the problem has been prescaled, he reports that $\alpha = 10^{-4}$ gives satisfactory solutions. More generally, Saunders recommends $\alpha \geq 10^{-5}\|A\|_2$. In Section 5.2, we include results for regularization combined with updating using a range of values of α.

4. QR with an iterative solver. In their paper on QR factorizations for LS problems, Avron et al [5] propose ignoring the dense rows and combining the use of a direct solver with an iterative one. Specifically, they suggest computing the QR factorization of the sparse row block A_s and then using the computed R factor as a preconditioner for an iterative method (such as LSQR [31] or LSMR [20]) applied to the original problem. Again, success of the QR step requires A_s to be of full rank. If A_s contains null columns, we can extend the applicability of the approach, either by first applying partial stretching or, for more general rank-deficient A_s, by employing regularization before the QR step.

In [5], limited numerical experiments are reported and the authors state that the updating approach of Heath is sometimes more efficient than using their preconditioned solver method but not always. Further details are not given. If we compare the two approaches under the assumption that A_s is of full rank then both compute the QR factorization of A_s and this is typically the most expensive step. The updating Algorithm 1 then involves a solve with R_s at Steps 2 and 6 (single right hand side) and a solve with R_s^T at Step 4 with m_d right-hand sides (recall that m_d is the number of rows in A_d), thus a total of $m_d + 2$ triangular solves are performed. In addition, there is a `_gemv` operation at Step 4 and the application of the dense routine `_getsls` at Step 5. By comparison, each iteration of an iterative solver with R_s as the preconditioner requires a matrix-vector product with A and with A^T plus a solve with R_s and a solve with R_s^T. Thus, unless the number of iterations required is very small and products with A_s and A_s^T very cheap, the QR direct-iterative approach of Avron *et al.* will be more expensive than the updating of Heath. The QR direct-iterative method is, however, potentially attractive if the LS problem needs to be solved for different vectors b because it is not necessary to store the Q factor, whereas the Heath approach requires $c_s = Q_s^T b_s$ to be computed for each b_s and so sufficient information on Q_s must be held to allow this. Note that because Q_s is typically much denser than A_s, it may be prohibitively expensive to store Q_s explicitly. To avoid this, the Householder transformations used to compute it can be stored.

Using an iterative solver may be necessary for very large problems for which memory limitations mean a QR factorization of A_s is not possible but it could possibly be replaced by an incomplete factorization. There have been a number of approaches to computing a LS preconditioner based on incomplete orthogonal factorizations (including [7, 8, 9, 27, 28, 32, 33, 43]); no comparisons have been made between them. This is probably because, with the exception of the MIQR package of Li and Saad [28] there is (as far as we are aware) no software available (and the implementations are non trivial). In their study of preconditioners for LS problems, Gould and Scott [22, 23] included MIQR. They reported that it was generally not competitive with other preconditioners (most notably, the limited memory incomplete Cholesky factorization of the normal equations available within the HSL mathematical software library [26] as package `HSL_MI35` [35, 37]) and thus we do not experiment with it here. Developing and implementing an algorithm for efficient and reliable incomplete orthogonal factorization preconditioners remains an open question.

5. Numerical experiments.

In this section, we look at the performance of the approaches that we have discussed so far when applied to practical applications. The serial HSL QR package `MA49` was developed in the 1990s [4]; more recently, there is SuiteSparseQR of Davis [15][1] and qr_mumps of Buttari [12][2]. These are general-purpose multifrontal sparse QR packages, and the latter two are designed to exploit parallelism. It is not our intention to try and compare these codes or to assess their efficiency for solving general sparse LS problems: our interest is in using an existing sparse QR package without modification to solve mixed sparse-dense LS problems. In our experiments, unless stated otherwise, we employ SuiteSparseQR with default settings and COLAMD ordering for sparsity [16]. The Fortran interface that we use here was also used in [23]. We note that SuiteSparseQR appears in MATLAB as QR and as `x = A\b` for rectangular systems and thus is widely used.

We have developed Fortran code for performing sparse and partial stretching. The iterative solver we use is a Fortran implementation of LSMR [20] with a reverse communication interface[3]. The initial solution guess is taken to be $x^{(0)} = 0$ and, following [23], we require the computed residual $r = b - Ax$ to satisfy either $\|r\|_2 < \delta_1$ or $ratio < \delta_2$, where

$$ratio = \frac{\|A^T r\|_2 / \|r\|_2}{\|A^T b\|_2 / \|b\|_2}. \tag{5.1}$$

The convergence tolerances δ_1 and δ_2 are set to 10^{-8} and 10^{-6}, respectively. The vector b is taken to

[1] http://faculty.cse.tamu.edu/davis/suitesparse.html
[2] http://buttari.perso.enseeiht.fr/qr_mumps/.
[3] http://stanford.edu/group/SOL/software/lsmr/

be the vector of 1's. In our experiments, we prescale A by normalizing each of its columns. That is, we replace A by AD, where D is the diagonal matrix with entries D_{ii} satisfying $D_{ii}^2 = 1/\|Ae_i\|_2$ (e_i denotes the i-th unit vector). The entries of AD are at most one in absolute value.

Our test problems, which are given in Table 5.1, are from the SuiteSparse Matrix Collection[4]. If necessary, the matrix is transposed to give an overdetermined system ($m > n$). The m_d dense rows are identified using the variant of the approach of Meszaros [29] described in [39] (with the density parameter set to 0.05). n_{ds} is the number of null columns in A_s. In Table 5.1, we report the results of employing SuiteSparseQR to solve the LS problem with no special handling of the dense rows (for problem PDE1, the memory requirements cause an error return). The characteristics of the test machine that we use are summarized in Table 5.2.

Table 5.1
Test examples. m and n are the row and column dimensions of A, m_d is the number of dense rows, n_{ds} is the number of null columns in A_s after the removal of the block A_d of m_d dense rows from A. nnz(R) is the number of entries in the R factor of A and flops is the number of floating-point operations to compute it. ratio is given by (5.1). The SuiteSparseQR solution time (denoted Time) is in seconds. NS denotes insufficient memory to perform QR factorization.

Identifier	m	n	m_d	n_{ds}	$nnz(R)$	$flops$	$\|x\|_2$	$\|r\|_2$	$ratio$	Time
lp_fit2p	13525	3000	25	0	4.502×10^{6}	5.687×10^{10}	1.689×10^{1}	1.105×10^{2}	3.841×10^{-9}	0.86
sctap1-2b	33858	15390	34	0	8.532×10^{6}	7.572×10^{10}	8.171×10^{1}	1.237×10^{2}	3.375×10^{-13}	1.67
sctap1-2r	63426	28830	34	0	2.952×10^{7}	4.973×10^{11}	1.117×10^{2}	1.694×10^{2}	5.603×10^{-13}	14.0
south31	36321	18425	5	0	1.618×10^{8}	3.839×10^{12}	2.748×10^{1}	1.881×10^{2}	3.008×10^{-13}	48.4
PDE1	271792	270595	1	0	NS					
aircraft	7517	3754	17	4	3.108×10^{6}	8.287×10^{9}	2.000	8.660×10^{1}	1.514×10^{-14}	0.15
sc205-2r	62423	35213	8	1	2.331×10^{7}	1.374×10^{11}	3.478×10^{2}	2.034×10^{2}	8.560×10^{-14}	3.00
scagr7-2b	13847	9743	7	1	2.388×10^{6}	8.961×10^{9}	1.387×10^{2}	6.068×10^{1}	3.561×10^{-13}	0.05
scagr7-2r	46679	32847	7	1	2.650×10^{7}	3.414×10^{11}	5.693×10^{2}	1.132×10^{2}	1.665×10^{-12}	1.13
scrs8-2r	27691	14364	22	7	2.218×10^{7}	1.989×10^{11}	6.436×10^{3}	1.354×10^{2}	9.289×10^{-12}	6.47
scsd8-2r	60550	8650	50	5	7.900×10^{6}	9.999×10^{10}	2.837	2.461×10^{2}	1.210×10^{-14}	1.78

Table 5.2
Test machine characteristics

CPU	two Intel Core i7-7700 3.6Ghz Quad Core processors
Memory	16 GB
Compiler	gfortran version 7.4.0 with options -O3 -fopenmp
BLAS	Intel MKL

5.1. A_s full rank.

We first present results for the problems in the top half of Table 5.1 for which A_s has no null columns (and so regularization is not employed). We compare three approaches:

- Updating (Algorithm 1).
- Sparse stretching (each row of A_d is stretched).
- The QR direct-iterative method of Section 4.

In our tables of results, these approaches are termed Updating, Stretching, and QR+LSMR, respectively. Updating and QR+LSMR compute the same QR factorization of A_s while Stretching computes the QR factorization of the stretched matrix. In Table 5.3, we report statistics for the QR factorization as well as the norm of the computed solution $\|x\|_2$ and residual $\|r\|_2$ (and for QR+LSMR, the number of iterations required to achieve convergence is given); timings are presented in Table 5.4. We give timings for each phase of the solution process together with the total time. For Updating, Solve is the time for solving the triangular linear systems in Steps 2, 4 and 6 of Algorithm 1 plus the time for solving (2.5). For QR+LSMR, Solve is the time to run preconditioned LSMR and, for Stretching, it is the time to taken to solve (1.4) with the R factor for the stretched system.

[4]`https://sparse.tamu.edu/`

Table 5.3

A performance comparison of QR-based approaches when A_s is full rank. $\tilde{m}$ and $\tilde{n}$ are the row and column dimensions of the matrix that is factorized using SuiteSparseQR. $nnz(R_s)$ is the number of entries in the R factor and flops is the number of floating-point operations to compute it. its denotes the number of LSMR iterations performed (QR+LSMR only). ratio is given by (5.1).

Identifier	Approach	$\tilde{m}$	$\tilde{n}$	$nnz(R_s)$	*flops*	*its*	$\|x\|_2$	$\|r\|_2$	*ratio*
lp_fit2p	Updating	13500	3000	3.000×10^3	4.050×10^4		1.689×10^1	1.105×10^2	5.570×10^{-11}
	Stretching	50284	39759	7.478×10^6	2.782×10^{11}		1.689×10^1	1.105×10^2	3.821×10^{-9}
	QR+LSMR	13500	3000	3.000×10^3	4.050×10^4	44	1.689×10^1	1.105×10^2	9.734×10^{-8}
sctap1-2b	Updating	33826	15390	1.111×10^5	4.063×10^6		8.171×10^1	1.237×10^2	3.348×10^{-12}
	Stretching	45172	26704	2.522×10^6	6.467×10^9		8.171×10^1	1.237×10^2	8.210×10^{-14}
	QR+LSMR	33826	15390	1.111×10^5	4.063×10^6	43	8.171×10^1	1.237×10^2	8.420×10^{-7}
sctap1-2r	Updating	63392	28830	2.084×10^5	7.618×10^6		1.116×10^2	1.694×10^2	1.427×10^{-11}
	Stretching	84596	50000	5.571×10^6	1.943×10^{10}		1.116×10^2	1.694×10^2	1.677×10^{-13}
	QR+LSMR	63392	28830	2.084×10^5	7.618×10^6	40	1.116×10^2	1.694×10^2	8.459×10^{-7}
south31	Updating	36317	18425	1.985×10^6	9.400×10^8		2.748×10^1	1.881×10^2	7.095×10^{-15}
	Stretching	36810	18914	5.905×10^6	1.888×10^{10}		2.748×10^1	1.881×10^2	4.675×10^{-15}
	QR+LSMR	36317	18425	1.985×10^6	9.400×10^8	6	2.748×10^1	1.881×10^2	9.704×10^{-15}
PDE1	Updating	271791	270595	1.392×10^7	7.813×10^9		4.282×10^2	3.030×10^2	1.337×10^{-11}
	Stretching	362388	361191	3.349×10^7	3.336×10^{10}		4.282×10^2	3.030×10^2	1.054×10^{-10}
	QR+LSMR	271791	270595	1.392×10^7	7.813×10^9	2	4.282×10^2	3.030×10^2	2.122×10^{-9}

Table 5.4

A comparison timings (in seconds) for QR-based approaches when A_s is full rank. Stretch denotes the time to perform stretching. Symbolic and Numeric are the times for the symbolic analysis and numerical factorization phases of SuiteSparseQR, respectively. Solve is the solution time after the QR factorization has been computed. Total denotes the total solution time.

Identifier	Approach	Stretch	Symbolic	Numeric	Solve	Total
lp_fit2p	Updating		0.001	0.001	0.002	0.004
	Stretching	0.443	0.055	1.166	0.006	1.670
	QR+LSMR		0.001	0.001	0.013	0.015
sctap1-2b	Updating		0.005	0.005	0.011	0.021
	Stretching	0.397	0.019	0.152	0.003	0.571
	QR+LSMR		0.009	0.006	0.048	0.063
sctap1-2r	Updating		0.012	0.009	0.025	0.046
	Stretching	1.158	0.042	0.460	0.008	1.668
	QR+LSMR		0.013	0.010	0.096	0.119
south31	Updating		0.006	0.071	0.018	0.095
	Stretching	0.019	0.010	0.397	0.006	0.432
	QR+LSMR		0.006	0.069	0.037	0.112
PDE1	Updating		0.150	0.485	0.057	0.692
	Stretching	21.17	0.215	1.269	0.035	22.69
	QR+LSMR		0.159	0.472	0.103	0.734

We see that each approach is successful in computing x and r with the same norm. As expected, Updating is faster than QR+LSMR, with the additional cost dependent on the number of LSMR iterations. For Updating, Solve can account for more than half the total solution time (for example, the sctap1 test cases) but for problems for which the numerical factorization is expensive, it adds little overhead. Observe, in particular, problem PDE1 that has only one row that is classified as dense and this is 67% full (which is much denser than rows in the other examples that are classified as dense). For this example, the majority of the Updating run time is taken by the QR step whereas for Stretching, the initial stretching of A dominates the time. In general, Stretching is expensive, both in terms of time and storage requirements because the stretched matrix can be much larger than the original one. Problem lp_fit2p is an extreme example of this because, in this instance, the sparsity pattern of A_s is close to diagonal, causing the sparse stretching algorithm to split each dense row into a large number of parts, which determines the dimensions of the stretched system. We note that our software that implements stretching has not been optimised and it may be possible to improve its efficiency. Nevertheless, the time to stretch the matrix clearly adds a significant overhead. Overall, our experiments suggest that if A_s is full rank then using Algorithm 1 (Updating) is the best of the approaches considered and offers very significant savings compared to the results of Table 5.1 that did not handle the dense rows separately. Note, in particular, the reduction in the total solution time for problem south31 (which has just 5 dense rows) from 48 seconds to less than 0.1 second.

5.2. A_s **rank deficient.** We now move to the more challenging case in which the sparse row block A_s is rank deficient. Here and elsewhere, we use $\tilde{A}_s$ to denote either the sparse row block of the partially stretched A or the regularized sparse matrix $\begin{bmatrix} A_s \\ \alpha I_n \end{bmatrix}$ and $\tilde{R}_s$ is its R factor.

Our first experiment compares applying Updating (Algorithm 1) directly to the original problem with applying it to the problem that results from first performing partial stretching to give a sparse row block $\tilde{A}_s$ that has no null columns. Results are given in Table 5.5. Here n_{ds} is the number of rows that are stretched, which is equal to the number of null columns in A_s. We see that, without stretching, *ratio* is large, $\|r\|_2$ is greater than for partial stretching and $\|x\|_2$ is very different, confirming that a direct application of Algorithm 1 does not yield the required result for rank deficient A_s. Partial stretching generally leads to a modest increase in the size of the matrix that is factorized (the exception is problem aircraft because for this example A_s is highly sparse). The only problem for which partial stretching is

Table 5.5
A performance comparison of approach Updating when A_s is rank deficient with and without partial stretching. n_{ds} denotes the number of rows that are stretched ($n_{ds} = 0$ denotes no stretching); $\tilde{m}$ and $\tilde{n}$ are the row and column dimensions of the matrix that is factorized using SuiteSparseQR. $nnz(\tilde{R}_s)$ is the number of entries in $\tilde{R}_s$ and flops is the number of floating-point operations needed to compute it. ratio is given by (5.1).

Identifier	n_{ds}	$\tilde{m}$	$\tilde{n}$	$nnz(\tilde{R}_s)$	*flops*	$\|x\|_2$	$\|r\|_2$	*ratio*
aircraft	0	7517	3754	3.750×10^{3}	2.250×10^{4}	4.964×10^{-3}	8.661×10^{1}	3.521×10^{-4}
	4	10517	6754	4.719×10^{4}	9.333×10^{5}	2.000	8.660×10^{1}	2.879×10^{-14}
sc205-2r	0	62423	35213	2.704×10^{5}	6.394×10^{6}	8.770×10^{1}	2.038×10^{2}	1.877×10^{-3}
	1	64023	36813	3.175×10^{5}	8.253×10^{6}	3.478×10^{2}	2.033×10^{2}	6.728×10^{-11}
scagr7-2b	0	13847	9743	6.026×10^{4}	2.417×10^{6}	1.104×10^{2}	6.069×10^{1}	7.784×10^{-5}
	1	15127	11023	1.186×10^{5}	5.029×10^{6}	1.387×10^{2}	6.068×10^{1}	8.845×10^{-12}
scagr7-2r	0	46679	32847	2.273×10^{5}	7.715×10^{6}	1.818×10^{2}	1.133×10^{2}	8.859×10^{-5}
	1	50999	37167	4.673×10^{5}	1.881×10^{7}	5.693×10^{2}	1.132×10^{2}	4.343×10^{-11}
scrs8-2r	0	27691	14364	8.200×10^{4}	1.171×10^{6}	7.961×10^{1}	1.459×10^{2}	3.966×10^{-3}
	7	32820	19493	4.242×10^{5}	4.655×10^{7}	6.435×10^{3}	1.354×10^{2}	2.457×10^{-12}
scsd8-2r	0	60550	8650	9.073×10^{4}	2.660×10^{7}	1.091×10^{-1}	2.461×10^{2}	4.055×10^{-2}
	5	62710	10810	2.421×10^{5}	3.880×10^{7}	2.373	2.461×10^{2}	2.237×10^{-3}

unsuccessful is scsd8-2r. For this example, A_s has 5 null columns but its rank deficiency is 6. Thus after stretching $n_{ds} = 5$ rows, the sparse part $\tilde{A}_s$ has rank deficiency 1 and Updating leads to large *ratio*. This can be remedied by stretching additional rows to make $\tilde{A}_s$ full rank.

Regularization requires the selection of an appropriate regularization parameter α. Table 5.6 illustrates that, if regularization is combined with Updating (that is, Algorithm 1 applied to (3.1)), then as α increases, the deviation of the computed solution from the desired solution may be unacceptable; it is also unacceptable for very small α. The computed solution is for the regularized problem. However, if we compute the QR factorization of the regularized sparse matrix, then we can use $\tilde{R}_s$ as a preconditioner for LSMR applied to the original system and thus compute the solution of the (unregularized) problem. That is, within the LSMR algorithm, matrix-vector products are with the original A and the stopping criteria are applied to the original system. In Figure 5.1, for $10^{-7} \le \alpha \le 1$ we plot the number of LSMR iterations for this approach applied to problems scagr7-2b and scrs8-2r. As we expect, the iteration count

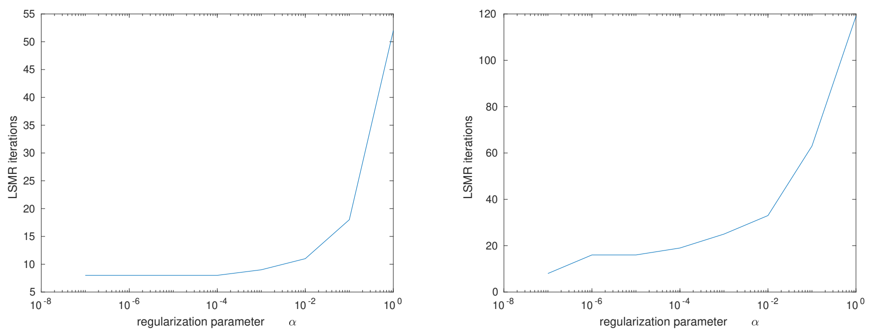
Table 5.6

Results for regularization combined with Updating for a range of values of the regularization parameter α. ratio is given by (5.1).

α	scagr7-2b			scrs8-2r		
	$\|x\|_2$	$\|r\|_2$	*ratio*	$\|x\|_2$	$\|r\|_2$	*ratio*
1.0×10^{-11}	1.1045×10^{2}	6.0686×10^{1}	7.7845×10^{-5}	7.9612×10^{1}	1.4595×10^{2}	3.9660×10^{-3}
1.0×10^{-10}	1.1045×10^{2}	6.0686×10^{1}	7.7845×10^{-5}	7.9612×10^{1}	1.4595×10^{2}	3.9660×10^{-3}
1.0×10^{-9}	1.3868×10^{2}	6.0677×10^{1}	9.8318×10^{-8}	6.4355×10^{3}	1.3537×10^{2}	3.5548×10^{-8}
1.0×10^{-8}	1.3868×10^{2}	6.0677×10^{1}	5.8658×10^{-8}	6.4355×10^{3}	1.3537×10^{2}	5.5789×10^{-9}
1.0×10^{-7}	1.3868×10^{2}	6.0677×10^{1}	6.9576×10^{-9}	6.4355×10^{3}	1.3537×10^{2}	5.6097×10^{-10}
1.0×10^{-6}	1.3868×10^{2}	6.0677×10^{1}	2.4533×10^{-10}	6.4355×10^{3}	1.3537×10^{2}	5.2442×10^{-11}
1.0×10^{-5}	1.3868×10^{2}	6.0677×10^{1}	3.0287×10^{-10}	6.4354×10^{3}	1.3537×10^{2}	1.6714×10^{-8}
1.0×10^{-4}	1.3867×10^{2}	6.0677×10^{1}	2.9832×10^{-8}	6.4250×10^{3}	1.3537×10^{2}	1.6677×10^{-6}
1.0×10^{-3}	1.3836×10^{2}	6.0677×10^{1}	2.9826×10^{-6}	5.5945×10^{3}	1.3539×10^{2}	1.5535×10^{-4}
1.0×10^{-2}	1.2140×10^{2}	6.0679×10^{1}	2.9486×10^{-4}	3.1728×10^{3}	1.3587×10^{2}	1.0847×10^{-2}
1.0×10^{-1}	8.7196×10^{1}	6.0969×10^{1}	2.7369×10^{-2}	8.9678×10^{2}	1.4093×10^{2}	1.5157×10^{-1}
1.0	2.8011×10^{1}	8.4479×10^{1}	5.9665×10^{-1}	1.8850×10^{1}	1.5274×10^{2}	5.0195

increases with α but, for a range of values, there is little variation in the count; it is only once $\alpha > 10^{-3}$ that a large number of iterations is needed. The lack of sensitivity is encouraging because it implies that, provided the problem is well scaled, the precise choice of the regularization parameter is not important.

Fig. 5.1. *LSMR iteration counts for a range of values of the regularization parameter α for problems scar7_2b (left) and scrs8-2r (right).*



We next compare partial stretching (denoted by PStretching) with regularization. Results are given in Tables 5.7 and 5.8. For approach PStretching, we apply partial stretching, compute the QR factorization of $\tilde{A}_s$, use the R factor $\tilde{R}_s$ as a preconditioner for LSMR applied to the stretched problem and then recover the solution of the original problem. For the approach Regular, we perform regularization (with $\alpha = 10^{-5}$), and use the computed R factor to precondition LSMR applied to the original problem. In each case, *ratio* is for the original problem. We see that both approaches are successful and the iteration counts are similar. However, *ratio* is consistently smaller for PStretching. To try to reduce *ratio* further for approach Regular, additional LSMR iterations can be performed. But in our experiments we observed that, in general, Regular was unable to produce values of *ratio* as small as those obtained with PStretching. The disadvantages of PStretching are that the number of entries in the R factor and the flop count to compute it are generally greater and, again, it is expensive to perform the partial stretching.

Finally, we consider how the number of dense rows effects the performance of preconditioned LSMR. We have already seen in Table 5.4 that the first three problems, which contain more dense rows than the final two, have a higher iteration count. We now explore varying the number of dense rows using the test problem south31. If the density threshold is set to 0.01 in the algorithm used for dense row detection, then 381 rows are classified as dense (with varying numbers of entries). We select dense rows in a random order and solve a modified problem in which we take A_s to be the matrix with the 381 dense rows discarded and

TABLE 5.7

A performance comparison of employing $\tilde{R}_s$ computed using partial stretching and regularization as a preconditioner for LSMR when A_s is rank deficient. $\tilde{m}$ and $\tilde{n}$ are the row and column dimensions of the matrix that is factorized using SuiteSparseQR. $nnz(\tilde{R}_s)$ is the number of entries in the factor $\tilde{R}_s$ and flops is the number of floating-point operations needed to compute it. its denotes the number of LSMR iterations performed. ratio is given by (5.1).

Identifier	Approach	$\tilde{m}$	$\tilde{n}$	$nnz(\tilde{R}_s)$	*flops*	*its*	$\|x\|_2$	$\|r\|_2$	*ratio*
aircraft	PStretching	10517	6754	4.719×10^{4}	9.333×10^{5}	9	2.000	8.660×10^{1}	4.947×10^{-12}
	Regularization	11271	3754	3.754×10^{3}	3.376×10^{4}	7	2.000	8.660×10^{1}	5.476×10^{-7}
sc205-2r	PStretching	64023	36813	3.175×10^{5}	8.253×10^{6}	6	3.478×10^{2}	2.033×10^{2}	9.983×10^{-9}
	Regularization	97636	35213	2.704×10^{5}	1.209×10^{7}	7	3.478×10^{2}	2.033×10^{2}	1.002×10^{-8}
scagr7-2b	PStretching	15127	11023	1.186×10^{5}	5.029×10^{6}	7	1.387×10^{2}	6.068×10^{1}	2.129×10^{-12}
	Regularization	23590	9743	6.027×10^{4}	3.667×10^{6}	8	1.387×10^{2}	6.068×10^{1}	3.979×10^{-9}
scagr7-2r	PStretching	50999	37167	4.673×10^{5}	1.881×10^{7}	7	5.693×10^{2}	1.132×10^{2}	1.046×10^{-10}
	Regularization	79526	32847	2.273×10^{5}	1.275×10^{7}	8	5.693×10^{2}	1.132×10^{2}	3.470×10^{-8}
scrs8-2r	PStretching	32820	19493	4.242×10^{5}	4.655×10^{7}	7	6.435×10^{3}	1.354×10^{2}	3.311×10^{-12}
	Regularization	42055	14364	8.200×10^{4}	2.835×10^{6}	16	6.435×10^{3}	1.354×10^{2}	3.532×10^{-7}

TABLE 5.8

A comparison of timings (seconds) of employing $\tilde{R}_s$ computed using partial stretching and regularization as a preconditioner for LSMR when A_s is rank deficient. Stretch denotes the time to perform stretching. Symbolic and Numeric are the times for the symbolic analysis and numerical factorization phases of SuiteSparseQR, respectively. Solve is the solution time after the QR factorization has been computed. Total denotes the total solution time.

Identifier	Approach	Stretch	Symbolic	Numeric	Solve	Total
aircraft	PStretching	0.058	0.002	0.002	0.004	0.066
	Regularization		0.001	0.001	0.001	0.003
sc205-2r	PStretching	0.091	0.016	0.011	0.021	0.139
	Regularization		0.015	0.010	0.017	0.042
scagr7-2b	PStretching	0.017	0.005	0.005	0.006	0.033
	Regularization		0.005	0.003	0.007	0.015
scagr7-2r	PStretching	0.179	0.016	0.017	0.024	0.236
	Regularization		0.017	0.012	0.022	0.051
scrs8-2r	PStretching	0.128	0.008	0.016	0.013	0.165
	Regularization		0.004	0.004	0.013	0.021
scsd8-2r	Regularization		0.010	0.007	0.041	0.058

then choose m_d to lie in the range 1 to 381 (the other $381 - m_d$ rows identified as dense are discarded). We perform regularization combined with LSMR and, in Table 5.9, we report the number of iterations for each choice of m_d. As expected, the number of iterations increases steadily with m_d, confirming that the approach is most suited to problems with a limited number of dense rows.

TABLE 5.9

The LSMR iteration count for the (modified) problem south31 as the number m_d of dense rows increases.

m_d	1	10	20	50	100	150	200	250	300	350	381
Iterations	3	11	23	41	44	44	61	137	166	175	181

6. Sparse-dense augmented system approach with iterative refinement.

In this section, we extend the augmented system formulation of LS problems to sparse-dense LS problems. In particular, we propose a QR-based approach for handling dense rows in A that enables the incorporation of iterative refinement. As already noted, LS problems can be ill conditioned, and so rounding errors may result in an insufficiently accurate solution; accuracy may be improved by employing iterative refinement. Before describing our extension, we recall the standard augmented system LS formulation with iterative refinement.

The idea was first suggested by Björck [10] in 1967. He proposed applying iterative refinement to the mathematically equivalent $(m+n)\times(m+n)$ augmented system

$$\begin{bmatrix} I_m & A \\ A^T & 0 \end{bmatrix}\begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \qquad r = b - Ax. \tag{6.1}$$

Given an initial solution $x^{(0)}$ and $r^{(0)} = b - Ax^{(0)}$, the $(i+1)$st refinement step proceeds as follows.

1. Compute the residual vector for the augmented system

$$\begin{bmatrix} f^{(i)} \\ g^{(i)} \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} - \begin{bmatrix} I_m & A \\ A^T & 0 \end{bmatrix}\begin{bmatrix} r^{(i)} \\ x^{(i)} \end{bmatrix} = \begin{bmatrix} b - r^{(i)} - Ax^{(i)} \\ -A^T r^{(i)} \end{bmatrix}. \tag{6.2}$$

2. Solve for the corrections

$$\begin{bmatrix} I_m & A \\ A^T & 0 \end{bmatrix}\begin{bmatrix} \delta r^{(i)} \\ \delta x^{(i)} \end{bmatrix} = \begin{bmatrix} f^{(i)} \\ g^{(i)} \end{bmatrix}. \tag{6.3}$$

3. Update the solution to the augmented system

$$\begin{bmatrix} r^{i+1} \\ x^{i+1} \end{bmatrix} = \begin{bmatrix} \delta r^{(i)} \\ \delta x^{(i)} \end{bmatrix} + \begin{bmatrix} r^{(i)} \\ x^{(i)} \end{bmatrix}. \tag{6.4}$$

In this way, the solution $x^{(i)}$ and residual $r^{(i)}$ are simultaneously refined. If the QR factorization of A has been computed, Björck showed that (6.3) can be solved by reusing the factors. To introduce our notation, consider the augmented system

$$\begin{bmatrix} I_m & A \\ A^T & 0 \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} w \\ t \end{bmatrix}. \tag{6.5}$$

Using the QR factorization (1.3), we have

$$\begin{bmatrix} I_m & AP \\ P^TA^T & 0 \end{bmatrix}\begin{bmatrix} u \\ P^Tv \end{bmatrix} = \begin{bmatrix} Q & \\ & I_n \end{bmatrix}\begin{bmatrix} I_n & 0 & R \\ 0 & I_{m-n} & 0 \\ R^T & 0 & 0 \end{bmatrix}\begin{bmatrix} Q^T & \\ & I_n \end{bmatrix}\begin{bmatrix} u \\ P^Tv \end{bmatrix} = \begin{bmatrix} w \\ P^Tt \end{bmatrix},$$

so that

$$\begin{bmatrix} I_n & 0 & R \\ 0 & I_{m-n} & 0 \\ R^T & 0 & 0 \end{bmatrix}\begin{bmatrix} e \\ f \\ P^Tv \end{bmatrix} = \begin{bmatrix} c \\ d \\ P^Tt \end{bmatrix},$$

where

$$\begin{bmatrix} c \\ d \end{bmatrix} = Q^Tw \qquad \text{and} \qquad u = Q\begin{bmatrix} e \\ f \end{bmatrix} = Q\begin{bmatrix} e \\ d \end{bmatrix}.$$

The component e is found by solving

$$PR^Te = t,$$

and finally v is the solution of

$$RP^Tv = c - e.$$

Thus a solve with R and with R^T plus one multiplication with Q and one with Q^T are required. The augmented system approach has been used by Demmel, Hida, Riedy, and Li [17] and, very recently, by Carson, Higham and Pranesh [14] in their work on multi-precision iterative refinement algorithm for LS problems.

Consider now the sparse-dense LS problem in which A is of the form (1.1). Assume a conformal partitioning of u and w. After block permuting, the augmented system (6.5) becomes

$$\begin{bmatrix} I_{m_s} & A_s & 0 \\ A_s^T & 0 & A_d^T \\ 0 & A_d & I_{m_d} \end{bmatrix} \begin{bmatrix} u_s \\ v \\ u_d \end{bmatrix} = \begin{bmatrix} w_s \\ t \\ w_d \end{bmatrix}. \tag{6.6}$$

Using the QR factorization (2.1) of the sparse block A_s, we obtain

$$\begin{bmatrix} I_n & 0 & R_s & 0 \\ 0 & I_{m_s-n} & 0 & 0 \\ R_s^T & 0 & 0 & P_s^T A_d^T \\ 0 & 0 & A_d P_s & I_{m_d} \end{bmatrix} \begin{bmatrix} e_s \\ f_s \\ P_s^T v \\ u_d \end{bmatrix} = \begin{bmatrix} c_s \\ d_s \\ P_s^T t \\ w_d \end{bmatrix}, \tag{6.7}$$

where P_s is the permutation from (2.1) and

$$\begin{bmatrix} c_s \\ d_s \end{bmatrix} = Q_s^T w_s \qquad \text{and} \qquad u_s = Q_s \begin{bmatrix} e_s \\ f_s \end{bmatrix} = Q_s \begin{bmatrix} e_s \\ d_s \end{bmatrix}.$$

Thus to solve (6.5) when A contains dense rows, we need to compute e_s, v and u_d. It is convenient to rewrite the system (6.7) as

$$\begin{bmatrix} \widehat{R}_s & \widehat{A}_d^T \\ \widehat{A}_d & I_{m_d} \end{bmatrix} \begin{bmatrix} \hat{v} \\ u_d \end{bmatrix} = \begin{bmatrix} \hat{t} \\ w_d \end{bmatrix}, \tag{6.8}$$

with

$$\widehat{R}_s = \begin{bmatrix} I_n & 0 & R_s \\ 0 & I_{m_s-n} & 0 \\ R_s^T & 0 & 0 \end{bmatrix}, \tag{6.9}$$

and

$$\widehat{A}_d = \begin{bmatrix} 0 & 0 & A_d P_s \end{bmatrix}, \quad \hat{v} = \begin{bmatrix} e_s \\ f_s \\ P_s^T v \end{bmatrix} \quad \hat{t} = \begin{bmatrix} c_s \\ d_s \\ P_s^T t \end{bmatrix}. \tag{6.10}$$

Consider the block factorization

$$\begin{bmatrix} \widehat{R}_s & \widehat{A}_d^T \\ \widehat{A}_d & I_{m_d} \end{bmatrix} = \begin{bmatrix} I_{m_s+n} & 0 \\ B & I_{m_d} \end{bmatrix} \begin{bmatrix} \widehat{R}_s & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} I_{m_s+n} & B^T \\ 0 & I_{m_d} \end{bmatrix} = \begin{bmatrix} \widehat{R}_s & \widehat{R}_s B^T \\ B\widehat{R}_s & S + B\widehat{R}_s B^T \end{bmatrix}. \tag{6.11}$$

Equating terms yields

$$\widehat{R}_s B^T = \widehat{A}_d^T \tag{6.12}$$

and

$$S = I_{m_d} - B\widehat{R}_s B^T. \tag{6.13}$$

Let $B = \begin{bmatrix} B_1 & B_2 & B_3 \end{bmatrix}$ with $B_1, B_3 \in \mathbb{R}^{m_d \times n}$ and $B_2 \in \mathbb{R}^{m_d \times m_s - n}$. Then from (6.10) and (6.12), we see that B_1^T is the solution of the system

$$P_s R_s^T B_1^T = A_d^T, \tag{6.14}$$

$B_2 = 0$, and B_3 satisfies

$$B_1^T + R_s B_3^T = 0. \tag{6.15}$$

It follows that

$$S = I_{m_d} - B_3 P_s^T A_d^T = I_{m_d} - B_3 R_s^T B_1^T = I_{m_d} + B_1 B_1^T. \tag{6.16}$$

Using the block factorization (6.11), we can solve (6.8) by solving the lower triangular system

$$\begin{bmatrix} I_{m_s+n} & 0 \\ B & I_{m_d} \end{bmatrix} \begin{bmatrix} \hat{y} \\ z \end{bmatrix} = \begin{bmatrix} \hat{t} \\ w_d \end{bmatrix}, \tag{6.17}$$

followed by

$$\begin{bmatrix} \widehat{R}_s & 0 \\ 0 & S \end{bmatrix} \begin{bmatrix} \hat{p} \\ q \end{bmatrix} = \begin{bmatrix} \hat{y} \\ z \end{bmatrix}, \tag{6.18}$$

and finally the upper triangular system

$$\begin{bmatrix} I_{m_s+n} & B^T \\ 0 & I_{m_d} \end{bmatrix} \begin{bmatrix} \hat{v} \\ u_d \end{bmatrix} = \begin{bmatrix} \hat{p} \\ q \end{bmatrix}. \tag{6.19}$$

From (6.17), $\hat{y} = \hat{t}$ and using (6.10) and (6.15),

$$z = w_d - B\hat{t} = w_d - (B_1 c_s + B_3 P_s^T t) = w_d - B_1(c_s - t^*)$$

where

$$R_s^T t^* = P_s^T t. \tag{6.20}$$

From (6.18), if $\hat{p}^T = \begin{bmatrix} p_1^T & p_2^T & p_3^T \end{bmatrix}$ with p_1, $p_3 \in \mathbb{R}^n$ and $p_2 \in \mathbb{R}^{m_s - n}$,

$$\begin{bmatrix} I_n & 0 & R_s \\ 0 & I_{m_s-n} & 0 \\ R_s^T & 0 & 0 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} c_s \\ d_s \\ P_s^T t \end{bmatrix}, \tag{6.21}$$

so that $p_2 = d_s$ and $R_s^T p_1 = P_s^T t$. It follows from (6.20) that $t^* = p_1$ and so

$$z = w_d - B_1(c_s - p_1)$$

Employing (6.18) and (6.19), u_d is the solution of the $m_d \times m_d$ symmetric positive definite system

$$S u_d = z.$$

Finally, from (6.19),

$$\hat{v} + B^T u_d = \hat{p}.$$

That is,

$$\begin{bmatrix} e_s \\ f_s \\ P_s^T v \end{bmatrix} + \begin{bmatrix} B_1^T \\ 0 \\ B_3^T \end{bmatrix} u_d = \begin{bmatrix} p_1 \\ d_s \\ p_3 \end{bmatrix}.$$

Thus we have

$$e_s = p_1 - B_1^T u_d$$

and

$$P_s^T v = p_3 - B_3^T u_d.$$

Premultiplying by R_s, using (6.15) and since from (6.21) $R_s p_3 = c_s - p_1$, it follows that v is the solution of the permuted triangular system

$$R_s P_s^T v = c_s - p_1 + B_1^T u_d = c_s - e_s.$$

We summarize the solution steps needed to solve the sparse-dense augmented system

$$\begin{bmatrix} I_{m_s} & 0 & A_s \\ 0 & I_{m_d} & A_d \\ A_s^T & A_d^T & 0 \end{bmatrix} \begin{bmatrix} u_s \\ u_d \\ v \end{bmatrix} = \begin{bmatrix} w_s \\ w_d \\ t \end{bmatrix}, \tag{6.22}$$

as Algorithm 2. The algorithm requires one solve with R_s, m_d+1 solves with R_s^T plus one multiplication

Algorithm 2 Solve the sparse-dense augmented system (6.22)

```
1: Compute the sparse QR factorization $A_s P_s = Q_s \begin{bmatrix} R_s \\ 0 \end{bmatrix}$ and set $\begin{bmatrix} c_s \\ d_s \end{bmatrix} = Q_s^T w_s$.
2: Solve $P_s R_s^T p_1 = t$.
3: Solve $P_s R_s^T B_1^T = A_d^T$.
4: Form $z = w_d - B_1(c_s - p_1)$.
5: Form $S = I_{m_d} + B_1 B_1^T$ and factorize it.
6: Use the factors of $S$ to solve $S u_d = z$.
7: Form $e_s = p_1 - B_1^T u_d$.
8: Solve $R_s P_s^T v = c_s - e_s$.
9: Set $u_s = Q_s \begin{bmatrix} e_s \\ d_s \end{bmatrix}$.
10: Return $u_s$, $u_d$ and $v$.
```

with Q_s and one with Q_s^T. Thus the main additional costs compared to the case when A has no dense rows are m_d solves with R_s^T plus dense linear algebra operations that also depend on m_d. In the special case of the sparse-dense LS augmented system, the system is

$$A_{aug} y = b_{aug}, \tag{6.23}$$

where

$$A_{aug} = \begin{bmatrix} I_{m_s} & 0 & A_s \\ 0 & I_{m_d} & A_d \\ A_s^T & A_d^T & 0 \end{bmatrix}, \qquad y = \begin{bmatrix} r_s \\ r_d \\ x \end{bmatrix}, \qquad b_{aug} = \begin{bmatrix} b_s \\ b_d \\ 0 \end{bmatrix}, \tag{6.24}$$

and Algorithm 2 simplifies to Algorithm 3, which reduces the number of solves with R_s^T to m_d. Observe that Step 2 of Algorithm 3 (the computation of B_1) and Step 4 (the computation and factorization of S) are independent of the right-hand side vector. Thus if Algorithm 3 is used to compute an initial LS solution $x^{(0)}$ and then Algorithm 2 is employed at each refinement iteration to compute the corrections $\delta x^{(i)}$ and $\delta r^{(i)}$, Steps 3 and 5 of Algorithm 2 have already been performed. The additional work for each refinement step is thus one solve with R_s, one with R_s^T plus one multiplication with Q_s and one with Q_s^T, together with the dense linear algebra operations in Steps 4, 6 and 7 of Algorithm 2. For small m_d it is clear that the work per refinement iteration for the sparse-dense case is essentially the same as for A with no dense rows.

A comparison of Algorithm 3 with the updating approach Algorithm 1 shows that $B_1 = K_d$. The factorization of S and subsequent solve in Algorithm 3 can be performed using the LAPACK routines `_potrf` and `_potrs`.

Algorithm 3 Solve the sparse-dense LS problem (1.2) using the augmented system (6.23)

```
1: Compute the sparse QR factorization $A_s P_s = Q_s \begin{bmatrix} R_s \\ 0 \end{bmatrix}$ and set $\begin{bmatrix} c_s \\ d_s \end{bmatrix} = Q_s^T b_s$.
2: Solve $P_s R_s^T B_1^T = A_d^T$.
3: Form $z = b_d - B_1 c_s$.
4: Form $S = I_{m_d} + B_1 B_1^T$ and factorize it.
5: Use the factors of $S$ to solve $S r_d = z$.
6: Form $e_s = -B_1^T r_d$.
7: Solve $R_s P_s^T x = c_s - e_s$.
8: Set $r_s = Q_s \begin{bmatrix} e_s \\ d_s \end{bmatrix}$.
9: Return $r_s$, $r_d$ and $x$.
```

6.1. Augmented approach with Krylov subspace refinement. When A_s is rank deficient, we can use the computed R factor ($\tilde{R}_s$) of the regularized sparse matrix $\tilde{A}_s$ to construct a preconditioner for a Krylov subspace solver such as GMRES or MINRES that we apply to the unregularized augmented system (6.23) to obtain the solution of the original problem. A straightforward choice is the simple $\tilde{R}_s$-block diagonal preconditioner

$$\begin{bmatrix} I_m & 0 \\ 0 & \tilde{R}_s^T \tilde{R}_s \end{bmatrix} = \begin{bmatrix} I_m & 0 \\ 0 & \tilde{R}_s^T \end{bmatrix} \begin{bmatrix} I_m & 0 \\ 0 & \tilde{R}_s \end{bmatrix} = M^T M. \tag{6.25}$$

This could be used for left preconditioning or, using it as a split preconditioner to retain symmetry, gives the preconditioned augmented matrix

$$M^{-T} \begin{bmatrix} I_m & A \\ A^T & 0 \end{bmatrix} M^{-1} = \begin{bmatrix} I_m & A\tilde{R}_s^{-1} \\ \tilde{R}_s^{-T} A^T & 0 \end{bmatrix}. \tag{6.26}$$

In the case of no dense rows ($m_d = 0$), (6.26) has three distinct non zero eigenvalues, 1 and $\frac{1}{2}(1 \pm \sqrt{5})$ [30]. Each application of the preconditioner (and thus each iteration of the Krylov subspace solver) requires a solve with $\tilde{R}_s$ and with $\tilde{R}_s^T$.

An alternative approach is to use a hybrid method that first solves the augmented system corresponding to the regularized LS problem (Algorithm 3) and then uses a preconditioned Krylov subspace solver as a refinement algorithm to recover the solution of the original system. This is summarized as Algorithm 4, where we use the notation (6.24) and define

$$y^{(i)} = \begin{bmatrix} r_s^{(i)} \\ r_d^{(i)} \\ x^{(i)} \end{bmatrix}, \qquad \delta y^{(i)} = \begin{bmatrix} \delta r_s^{(i)} \\ \delta r_d^{(i)} \\ \delta x^{(i)} \end{bmatrix}, \qquad s^{(i)} = \begin{bmatrix} f_s^{(i)} \\ f_d^{(i)} \\ g^{(i)} \end{bmatrix}. \tag{6.27}$$

Observe that the Q factor need not be retained for the refinement.

In Table 6.1, results are presented for our test problems for which A_s is rank deficient. Results are given for split preconditioned GMRES and MINRES applied to the augmented system (6.23) with initial solution $x^{(0)} = 0$ and for Algorithm 4 (with GMRES and MINRES as the iterative refinement solver). For the latter, we give the ratio (5.1) before refinement, that is,

$$ratio_{init} = \frac{\|A^T r^{(0)}\|_2 / \|r^{(0)}\|_2}{\|A^T b\|_2 / \|b\|_2}, \tag{6.28}$$

and after a single refinement step ($it_{max} = 1$). The GMRES and MINRES implementations from the HSL mathematical software library [26] are used with the convergence tolerance set to 10^{-7}. Timings are given in Table 6.2. Here "Augmented" denotes the time within Algorithm 4 for solving the augmented system

Algorithm 4 Solve the sparse-dense LS problem (1.2) using the augmented system approach with regularization and Krylov solver refinement

```
1: Apply Algorithm 3 to the regularized sparse-dense LS problem $\min_x \left\| \begin{bmatrix} A_s \\ \alpha I_n \\ A_d \end{bmatrix} x - \begin{bmatrix} b_s \\ 0 \\ b_d \end{bmatrix} \right\|_2^2$.
   Store the computed R factor $\tilde{R}_s$ and the solution $x_{aug}$.
2: Set $x^{(0)} = x_{aug}$ and compute $\begin{bmatrix} r_s^{(0)} \\ r_d^{(0)} \end{bmatrix} = \begin{bmatrix} b_s - A_s x^{(0)} \\ b_d - A_d x^{(0)} \end{bmatrix}$. Set $y^{(0)} = \begin{bmatrix} r_s^{(0)} \\ r_d^{(0)} \\ x^{(0)} \end{bmatrix}$.
3: for $i = 0 : it_{max} - 1$ do
4:     Compute the residual vector $s^{(i)} = b_{aug} - A_{aug} y^{(i)}$.
5:     Use the Krylov subspace solver with the $\tilde{R}_s$-block diagonal preconditioner to solve the correction
       system $A_{aug} \delta y^{(i)} = s^{(i)}$.
6:     Set $y^{(i+1)} = y^{(i)} + \delta y^{(i)}$.
7:     if converged then
8:         Return $x = x^{(i+1)}$, $r = \begin{bmatrix} r_s^{(i+1)} \\ r_d^{(i+1)} \end{bmatrix}$ stop
9:     end if
10: end for
```

(6.23) using the computed QR factorization of the regularized problem (that is, it is the time for steps 2 to 8 of Algorithm 3). Each approach is successful (the computed $\|x\|$ and $\|r\|$ are consistent with those given in Table 5.7), with the GMRES iteration counts being smaller than those for MINRES, resulting in lower times (the latter offers the advantage of requiring less memory). As we would expect, in general the iteration counts are higher for a larger regularization parameter α. Observe that, for $\alpha = 10^{-5}$ (the results given in the lower half of Table 6.1), the stopping criterion that we employed in Section 5 for tests with LSMR (namely, $ratio < \delta_2$ with $\delta_2 = 10^{-6}$) is satisfied by $ratio_{init}$ for all the examples except scsd8-2r. Thus, with the exception of this problem, refinement of the solution x_{aug} of the augmented system for the regularized problem is only required if we want to reduce $ratio$ further. Note also that the most expensive part of the solution process is the Krylov solver; the time could potentially be reduced by optimising the implementation of the matrix-vector products.

6.2. Augmented approach with multi-precision refinement. Motivated by the emergence of multi-precision capabilities in hardware, Carson, Higham and Pranesh [14] have recently studied multi-precision iterative refinement for LS problems using the augmented system formulation (6.1) [10]. They propose reducing the overall solution cost by performing the QR factorization using low precision arithmetic. Their GMRES-LSIR algorithm then solves (6.1) using GMRES preconditioned by a matrix based on the low precision QR factors to obtain the LS solution to working precision. Extending their work on multi-precision for square linear systems [13], they employ three precisions: u_f for the matrix factorization, u for the working precision, and u_r for the computation of residuals. Results are presented for u_f equal to half and to single precision, u equal to single or double precision and u_r equal to single, double or quad precision such that $u_f \geq u \geq u_r$. The reported numerical experiments [14] demonstrate that, provided the condition number of the system matrix is not too large, three-precision refinement using GMRES is able to solve a range of problems. Algorithm 5 extends the approach to sparse-dense LS problems with A_s of full rank. The notation is as defined in (6.24) and (6.27). As before, MINRES could be used in place of GMRES. If A_s is rank deficient, we can derive a multi-precision version of Algorithm 4 by choosing a regularization parameter α and replacing Steps 1 and 2 of Algorithm 5 as follows:

Table 6.1

A performance comparison of split-preconditioned GMRES and MINRES used directly to solve the augmented system (6.23) and for refinement (Algorithm 4 using GMRES and MINRES with $it_{max} = 1$). In each case, the sparse block A_s is rank deficient and the QR factorization of the regularized sparse matrix $\tilde{A}_s$ is computed. Results in the upper (respectively, lower) half of the table are for the regularization parameter $\alpha = 10^{-2}$ (respectively, $\alpha = 10^{-5}$). its denotes the number of GMRES or MINRES iterations performed. ratio is given by (5.1) and $ratio_{init}$ by (6.28).

Identifier	Approach	*its*	$ratio_{init}$	*ratio*
aircraft	GMRES / MINRES	18 / 24		$1.058{\times}10^{-14}$ / $4.682{\times}10^{-9}$
	Algorithm 4+GMRES / MINRES	18 / 22	$1.274{\times}10^{-7}$	$3.659{\times}10^{-16}$ / $1.751{\times}10^{-13}$
sc205-2r	GMRES / MINRES	20 / 28		$4.658{\times}10^{-9}$ / $1.376{\times}10^{-8}$
	Algorithm 4+GMRES / MINRES	20 / 30	$2.240{\times}10^{-4}$	$1.602{\times}10^{-12}$ / $1.835{\times}10^{-12}$
scagr7-2b	GMRES / MINRES	22 / 40		$8.761{\times}10^{-11}$ / $1.060{\times}10^{-10}$
	Algorithm 4+GMRES / MINRES	22 / 40	$2.949{\times}10^{-4}$	$8.961{\times}10^{-13}$ / $1.391{\times}10^{-13}$
scagr7-2r	GMRES / MINRES	22 / 40		$3.151{\times}10^{-10}$ / $1.824{\times}10^{-9}$
	Algorithm 4+GMRES / MINRES	22 / 40	$3.181{\times}10^{-4}$	$4.492{\times}10^{-11}$ / $3.442{\times}10^{-11}$
scrs8-2r	GMRES / MINRES	38 / 42		$3.229{\times}10^{-8}$ / $3.270{\times}10^{-8}$
	Algorithm 4+GMRES / MINRES	42 / 98	$1.246{\times}10^{-3}$	$2.999{\times}10^{-12}$ / $1.268{\times}10^{-11}$
scsd8-2r	GMRES / MINRES	38 / 70		$1.924{\times}10^{-9}$ / $2.706{\times}10^{-8}$
	Algorithm 4+GMRES / MINRES	40 / 88	$1.070{\times}10^{-4}$	$3.220{\times}10^{-14}$ / $3.264{\times}10^{-11}$
aircraft	GMRES / MINRES	18 / 20		$8.033{\times}10^{-15}$ / $2.437{\times}10^{-9}$
	Algorithm 4+GMRES / MINRES	20 / 24	$1.298{\times}10^{-13}$	$1.022{\times}10^{-17}$ / $1.010{\times}10^{-17}$
sc205-2r	GMRES / MINRES	14 / 30		$2.445{\times}10^{-9}$ / $2.404{\times}10^{-9}$
	Algorithm 4+GMRES / MINRES	16 / 30	$4.514{\times}10^{-7}$	$1.635{\times}10^{-12}$ / $1.635{\times}10^{-12}$
scagr7-2b	GMRES / MINRES	16 / 28		$3.425{\times}10^{-9}$ / $7.308{\times}10^{-9}$
	Algorithm 4+GMRES / MINRES	18 / 38	$2.985{\times}10^{-10}$	$1.282{\times}10^{-12}$ / $1.282{\times}10^{-12}$
scagr7-2r	GMRES / MINRES	16 / 28		$1.943{\times}10^{-8}$ / $5.031{\times}10^{-8}$
	Algorithm 4+GMRES / MINRES	18 / 46	$3.594{\times}10^{-10}$	$6.906{\times}10^{-12}$ / $6.905{\times}10^{-12}$
scrs8-2r	GMRES / MINRES	30 / 44		$1.958{\times}10^{-9}$ / $4.654{\times}10^{-8}$
	Algorithm 4+GMRES / MINRES	30 / 48	$2.429{\times}10^{-9}$	$4.039{\times}10^{-12}$ / $4.039{\times}10^{-12}$
scsd8-2r	GMRES / MINRES	36 / 98		$6.843{\times}10^{-13}$ / $7.359{\times}10^{-8}$
	Algorithm 4+GMRES / MINRES	36 / 100	$2.685{\times}10^{-6}$	$1.793{\times}10^{-14}$ / $8.231{\times}10^{-14}$

Table 6.2

Timings (in seconds) for split-preconditioned GMRES and MINRES used directly to solve the augmented system (6.23) and for refinement (Algorithm 4 using GMRES and MINRES with $it_{max} = 1$). Symbolic and Numeric are the times for the symbolic analysis and numerical factorization phases of SuiteSparseQR, respectively. Augmented is the time within Algorithm 4 after the QR factorization has been computed to solve the augmented system (6.23) and Iterative is the time for GMRES or MINRES. Total denotes the total solution time. The regularization parameter is $\alpha = 10^{-5}$.

Identifier	Approach	Symbolic	Numeric	Augmented	Iterative	Total
aircraft	GMRES / MINRES	0.001	0.001		0.006 / 0.006	0.008 / 0.008
	Algorithm 4+GMRES / MINRES	0.001	0.001	0.003	0.007 / 0.007	0.012 / 0.013
sc205-2r	GMRES / MINRES	0.013	0.010		0.037 / 0.086	0.060 / 0.106
	Algorithm 4+GMRES / MINRES	0.013	0.009	0.020	0.050 / 0.087	0.092 / 0.125
scagr7-2b	GMRES / MINRES	0.003	0.003		0.008 / 0.018	0.014 / 0.025
	Algorithm 4+GMRES / MINRES	0.004	0.003	0.008	0.012 / 0.023	0.027 / 0.040
scagr7-2r	GMRES / MINRES	0.014	0.012		0.045 / 0.082	0.071 / 0.107
	Algorithm 4+GMRES / MINRES	0.014	0.011	0.028	0.049 / 0.132	0.102 / 0.185
scrs8-2r	GMRES / MINRES	0.005	0.004		0.034 / 0.049	0.043 / 0.057
	Algorithm 4+GMRES / MINRES	0.005	0.004	0.011	0.031 / 0.065	0.051 / 0.094
scsd8-2r	GMRES / MINRES	0.008	0.007		0.079 / 0.180	0.094 / 0.195
	Algorithm 4+GMRES / MINRES	0.008	0.007	0.009	0.072 / 0.199	0.097 / 0.223

1: Apply Algorithm 3 to the regularized sparse-dense LS problem $\min_x \left\| \begin{bmatrix} A_s \\ \alpha I_n \\ A_d \end{bmatrix} x - \begin{bmatrix} b_s \\ 0 \\ b_d \end{bmatrix} \right\|_2^2$ using precision u_f. Store the computed R factor $\tilde{R}_s$ using precision u_f and the solution x_{aug} using precision u.

```
2: Set $x^{(0)} = x_{aug}$ and compute $\begin{bmatrix} r_s^{(0)} \\ r_d^{(0)} \end{bmatrix} = \begin{bmatrix} b_s - A_s x^{(0)} \\ b_d - A_d x^{(0)} \end{bmatrix}$. Set $y^{(0)} = \begin{bmatrix} r_s^{(0)} \\ r_d^{(0)} \\ x^{(0)} \end{bmatrix}$ using precision $u$.
```

Algorithm 5 Solve the sparse-dense LS problem (1.2) using GMRES-based iterative refinement with precisions $u_f \geq u \geq u_r$. A_s is assumed to be full rank.

```
1: Apply Algorithm 3 to the sparse-dense LS problem (1.2) using precision $u_f$. Store the computed R
   factor $R_s$ using precision $u_f$, and store the solution $x_{aug}$ and residual $r_{aug}$ using precision $u$.
2: Set $y^{(0)} = \begin{bmatrix} r_{aug} \\ x_{aug} \end{bmatrix}$ using precision $u$.
3: for $i = 0 : it_{max} - 1$ do
4:     Compute the residual vector $s^{(i)} = b_{aug} - A_{aug} y^{(i)}$ using precision $u_r$; round $s^{(i)}$ to precision $u$.
5:     Use GMRES with the $R_s$-block diagonal preconditioner to solve the correction system
       $A_{aug} \delta y^{(i)} = s^{(i)}$ using precision $u$, with matrix-vector products computed using precision $u_r$.
6:     Set $y^{(i+1)} = y^{(i)} + \delta y^{(i)}$ using precision $u$.
7:     if converged then
8:         Return $x = x^{(i+1)}$, $r = \begin{bmatrix} r_s^{(i+1)} \\ r_d^{(i+1)} \end{bmatrix}$ stop
9:     end if
10: end for
```

To illustrate the use of multi-precision arithmetic, we set u_f to single precision and $u = u_r$ to double precision. We employ the HSL code `MA49` for computing the QR factorization of A_s (or $\tilde{A}_s$) because it is available in both double and single precision versions. Results are presented in Table 6.3 for $it_{max} = 1$. In the upper part of the table, results are for the test problems for which A_s is of full rank and in the lower part, A_s is rank deficient and we set the regularization parameter to 10^{-3}. We found that for smaller values of α, Algorithm 3 can break down at Step 4 because S is found to be indefinite. The results show that the multi-precision approach is successful and offers another option.

Table 6.3

Multi precision results (Algorithm 5). The problems in the upper part do not use regularization while those in the lower part use regularization with $\alpha = 10^{-3}$. its denotes the number of GMRES iterations performed. $it_{max} = 1$. ratio is given by (5.1) and $ratio_{init}$ by (6.28).

Identifier	*its*	$\|x\|$	$\|r\|$	$ratio_{init}$	*ratio*
lp_fit2p	65	1.689×10	1.105×10^{2}	9.529×10^{-3}	1.023×10^{-12}
sctap1-2b	93	8.171×10	1.237×10^{2}	2.210×10^{-3}	1.068×10^{-14}
sctap1-2r	93	1.117×10^{2}	1.694×10^{2}	3.684×10^{-3}	4.780×10^{-14}
south31	48	2.748×10	1.881×10^{2}	2.675×10^{-6}	4.259×10^{-15}
aircraft	32	2.000	8.660×10	1.641×10^{-6}	6.739×10^{-17}
sc205-2r	32	3.478×10^{2}	2.034×10^{2}	1.049×10^{-2}	2.099×10^{-11}
scagr7-2b	34	1.387×10^{2}	6.068×10	1.045×10^{-3}	5.559×10^{-16}
scagr7-2r	33	5.693×10^{2}	1.132×10^{2}	1.843×10^{-2}	5.268×10^{-11}
scrs8-2r	63	6.436×10^{3}	1.354×10^{2}	1.334×10^{-3}	4.621×10^{-13}
scsd8-2r	70	2.837	2.461×10^{2}	1.686×10^{-1}	6.070×10^{-12}

7. Concluding remarks.

This computational study has explored how to employ black-box sparse QR solvers to efficiently and robustly solve sparse-dense LS problems. Particular emphasis has been on problems in which the sparse part A_s of the system matrix A is rank deficient. In this case, we propose combining basic approaches for sparse-dense LS problems with either partial matrix stretching or regularization. For the former, the solution of the original system can be obtained directly using the sparse QR solver combined with updating. However, when regularization is used, it may be necessary to use a direct-iterative approach to achieve the required accuracy for the original problem. Alternatively, a preconditioned Krylov solver can be applied to the augmented system formulation of the LS problem. We have demonstrated that, using this formulation, there is a significant potential for multi-precision solvers.

While partial matrix stretching offers an attractive way forward, more efficient implementations of the algorithm that performs the stretching are needed to make it really viable. When there is more than one row that requires stretching, in the future we plan to develop novel block stretching strategies in which rows in A_d with similar sparsity patterns are handled together. This could potentially significantly reduce the computational cost.

Finally, we observe that while we have focused our experiments on LS problems where A_d corresponds to dense rows in A, the proposed algorithms can also be applied when A_d represents rows (either sparse or dense) that are added to A after an initial QR factorization has been performed.

Acknowledgements. We are grateful to our colleagues Nick Gould for providing us with the Fortran interface to SuiteSparseQR that was used in our experiments and Iain Duff for revising `MA49` to add extra functionality to allow the R factor and associated row permutation to be explicitly obtained.

REFERENCES

[1] M. Adlers. Topics in sparse least squares problems. Technical Report, Department of Mathematics, Linköping University, Linköping, Sweden, 2000.

[2] M. Adlers and Å. Björck. Matrix stretching for sparse least squares problems. *Numerical Linear Algebra with Applications*, 7(2):51–65, 2000.

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

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

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

[6] C. Aykanat, A. Pinar, and Ü. V. Çatalyürek. Permuting sparse rectangular matrices into singly-bordered block-diagonal form for parallel solution of LP problems. Technical Report BU-CE-0203, Computer Engineering Department, Bilkent Univeristy, Ankara, Turkey, 2002.

[7] 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(1):53–70, 2001.

[8] Z.-Z. Bai, I. S. Duff, and J.-F. Yin. Numerical study on incomplete orthogonal factorization preconditioners. *J. Comput. Appl. Math.*, 226(1):22–41, 2009.

[9] Z.-Z. Bai and J.-F. Yin. Modified incomplete orthogonal factorization methods using Givens rotations. *Computing*, 86(1):53–69, 2009.

[10] Å. Björck. Iterative refinement of linear least squares solutions I. *BIT Numerical Mathematics*, 7:257–278, 1967.

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

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

[13] E. Carson and N. Higham. Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM J. on Scientific Computing*, 4(2):A817–A84, 2018.

[14] E. Carson, N. Higham, and S. Pranesh. Three-precision GMRES-based iterative refinement for least squares problems. Technical Report MIMS EPrint 2020.5, Manchester Institute for Mathematical Sciences, The University of Manchester, 2020.

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

[16] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng. Algorithm 836: Colamd, a column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software*, 30:377–380, 2004.

[17] J. Demmel, Y. Hida, E. J. Riedy, and X. S. Li. Extra-precise iterative refinement for overdetermined least squares problems. *ACM Transactions on Mathematical Software*, 35:28:1–28:32, 2009.

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

[19] M. C. Ferris and J. D. Horn. Partitioning mathematical programs for parallel solution. *Mathematical Programming*, 80:35–62, 1998.

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

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

[22] N. I. M. Gould and J. A. Scott. The state-of-the-art of preconditioners for sparse linear least squares problems: the complete results. Technical Report RAL-TR-2015-009, Rutherford Appleton Laboratory, 2015.

[23] N. I. M. Gould and J. A. Scott. The state-of-the-art of preconditioners for sparse linear least squares problems. *ACM Transactions on Mathematical Software*, 43(4):36:1–35, 2017.

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

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

[26] HSL. A collection of Fortran codes for large-scale scientific computation, 2018. `http://www.hsl.rl.ac.uk`.

[27] A. Jennings and M. A. Ajiz. Incomplete methods for solving $A^TAx = b$. *SIAM J. on Scientific and Statistical Computing*, 5(4):978–987, 1984.

[28] N. Li and Y. Saad. MIQR: A multilevel incomplete QR preconditioner for large sparse least-squares problems. *SIAM J. on Matrix Analysis and Applications*, 28(2), 2006.

[29] C. Meszaros. Detecting dense columns in interior point methods for linear programs. *Computational Optimization and Applications*, 36:309–320, 2007.

[30] M. F. Murphy, G. H. Golub, and A. J. Wathen. A note on preconditioning for indefinite linear systems. *SIAM J. on Scientific and Statistical Computing*, 21(6):1969–1972, 2000.

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

[32] 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(1):159–179, 2005.

[33] Y. Saad. Preconditioning techniques for nonsymmetric and indefinite linear systems. *J. of Computational and Applied Mathematics*, 24(1-2):89–105, 1988.

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

[35] J. A. Scott and M. Tůma. `HSL_MI28`: an efficient and robust limited-memory incomplete Cholesky factorization code. *ACM Transactions on Mathematical Software*, 40(4):24:1–19, 2014.

[36] J. A. Scott and M. Tůma. Solving mixed sparse-dense linear least-squares problems by preconditioned iterative methods. *SIAM J. on Scientific Computing*, 39(6):A2422–A2437, 2017.

[37] J. A. Scott and M. Tůma. A Schur complement approach to preconditioning sparse least-squares problems with some dense rows. *Numerical Linear Algebra with Applications*, 79:1147–1168, 2018. DOI: 10.1007/s11075-018-0478-2.

[38] J. A. Scott and M. Tůma. Sparse stretching for solving sparse-dense linear least-squares problems. *SIAM J. on Scientific Computing*, 41:A1604—A1625, 2019.

[39] J. A. Scott and M. Tůma. Strengths and limitations of stretching for least-squares problems with some dense rows. *ACM Transactions on Mathematical Software*, 2020. To appear.

[40] C. Sun. Dealing with dense rows in the solution of sparse linear least squares problems. Research Report CTC95TR227, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, 1995.

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

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

[43] X. Wang, K. A. Gallivan, and R. Bramley. CIMGS: an incomplete orthogonal factorization preconditioner. *SIAM J. on Scientific Computing*, 18(2):516–536, 1997.

[44] S. N. Yeralan, T. A. Davis, and S. Ranka. Algorithm 980: Sparse QR factorization on the GPU. *ACM Transactions on Mathematical Software*, 44:17:1–17:29, 2017.