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

A Computational Study of Using Black-box QR Solvers for Large-scale Sparse-dense Linear Least Squares Problems

JENNIFER SCOTT, STFC Rutherford Appleton Laboratory and University of Reading, UK
MIROSLAV TŮMA, Charles University, Czech Republic

Large-scale overdetermined linear least squares problems arise in many practical applications. One popular solution method is based on the backward stable QR factorization of the system matrix A. This article focuses on sparse-dense least squares problems in which A is sparse except from a small number of rows that are considered dense. For large-scale problems, the direct application of a QR solver either fails because of insufficient memory or is unacceptably slow. We study several solution approaches based on using a sparse QR solver without modification, focussing on the case that the sparse part of A is rank deficient. We discuss partial matrix stretching and regularization and propose extending the augmented system formulation with iterative refinement for sparse problems to sparse-dense problems, optionally incorporating multi-precision arithmetic. In summary, our computational study shows that, before applying a black-box QR factorization, a check should be made for rows that are classified as dense and, if such rows are identified, then A should be split into sparse and dense blocks; a number of ways to use a black-box QR factorization to exploit this splitting are possible, with no single method found to be the best in all cases.

CCS Concepts: • **Mathematics of computing → Mathematical analysis**; **Numerical analysis**;

Additional Key Words and Phrases: Sparse matrices, linear least-squares problems, dense rows, QR factorization

ACM Reference format:
Jennifer Scott and Miroslav Tůma. 2022. A Computational Study of Using Black-box QR Solvers for Large-scale Sparse-dense Linear Least Squares Problems. *ACM Trans. Math. Softw.* 48, 1, Article 5 (February 2022), 24 pages.
https://doi.org/10.1145/3494527

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

Jennifer Scott was partially supported by the UK Engineering and Physical Sciences Research Council grant EP/M025179/1 and Miroslav Tůma by project 18-12719S of the Grant Agency of the Czech Republic.
Authors' addresses: J. Scott, 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; email: jennifer.scott@stfc.ac.uk; M. Tůma, Faculty of Mathematics and Physics, Charles University, Sokolovská 49/83, 186 75 Praha 8, Czech Republic; email: mirektuma@karlin.mff.cuni.cz.

0098-3500/2022/02-ART5
https://doi.org/10.1145/3494527

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 [24, 25] 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

$$A = \begin{bmatrix} A_s \\ A_d \end{bmatrix}, \; A_s \in \mathbb{R}^{m_s \times n}, \; A_d \in \mathbb{R}^{m_d \times n}, \; b = \begin{bmatrix} b_s \\ b_d \end{bmatrix}, \; b_s \in \mathbb{R}^{m_s}, \; b_d \in \mathbb{R}^{m_d}, \tag{1}$$

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

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

We assume that A has full column rank, in which case the solution of Equation (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. First, there is a potential loss of information in explicitly computing the $n \times n$ symmetric positive definite normal matrix C and the vector $A^T b$. Second, if A contains just a single dense row, then C is not sparse, and, thus, if n is large, then it cannot be stored or factorized by a direct solver that computes a Cholesky factorization. Third, there is the fact that the condition number of C is the square of that of A, so that an accurate solution may be difficult to compute if A is poorly conditioned. If A is not full rank, then 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

$$AP = Q \begin{bmatrix} R \\ 0 \end{bmatrix} \qquad \text{and} \qquad b = Q \begin{bmatrix} c \\ d \end{bmatrix}, \tag{3}$$

where $R \in \mathbb{R}^{n \times n}$ is upper triangular (and nonsingular 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 Equation (2) may be obtained by solving the system

$$RP^T x = c. \tag{4}$$

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 References [4, 12, 15, 47]). Unfortunately, for large problems of the form of Equation (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.

In the 1980s and 1990s, the difficulties that a (small) dense row block presents were studied by several authors, see References [2, 11, 21, 27, 43–45]. Recently, Scott and Tůma have revisited this problem and have considered a number of possible approaches:

- a block factorization method that processes the rows that are identified as dense separately within an iterative solver [39];
- a Schur complement approach that exploits the block structure within the augmented system formulation of the LS problem and can be combined with a direct or an iterative solver [40];
- new sparse stretching strategies that replace the dense row block by a sparser but larger block [41, 42].

The purpose of this article is to build on these ideas and to incorporate them with other ideas into looking at how we can use a black-box QR package to solve LS problems with a small number of dense rows both efficiently and robustly. Several approaches are considered and compared using problems from practical applications. This was lacking in earlier papers on using QR for sparse-dense LS problems, such as References [2, 27] that present algorithms with no numerical results to show how effective they are. 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.

This article offers the following novel contributions:

(a) it brings together different QR-based ideas in a single, cohesive presentation;
(b) it compares their performance on a range of practical applications;
(c) it proposes computing a QR factorization of the regularized problem and then using an iterative approach to recover the solution of the unregularized problem;
(d) it explores the use of mixed-precision computation to try to reduce the computational cost.

In Section 2, we discuss the software packages and test examples that are used in this study. We then commence our study by considering direct methods. Section 3 recalls the use of updating [27] to handle dense rows. We then consider two preprocessing approaches that extend the applicability of updating when A_s is rank deficient: partial matrix stretching [42] and regularization [36]. Both avoid break down of the QR algorithm by enlarging the problem that it is applied to. In Section 4, we discuss a hybrid approach that combines using a QR factorization of the (possibly enlarged) sparse row block with an iterative solver. An alternative method based on the augmented system formulation of the LS problem is considered in Section 5. Our proposed extension to sparse-dense LS problems allows the incorporation of iterative refinement with a preconditioned Krylov subspace solver. Numerical experiments to illustrate the performance of the different approaches are presented in Section 6. These show that updating works well if A_s is of full rank: it is straightforward to implement and robust and offers significant savings compared to applying the QR solver with no special handling of dense rows (even if the rows that are classified as dense are far from being full). For the more challenging case of rank-deficient A_s, we find that preprocessing using either regularization or partial stretching is effective, with the former being the easier to implement using existing software packages while the latter obtains more accurate solutions. A key attraction of the augmented system approach is that it can employ multi-precision arithmetic, which has the potential to reduce the solution cost (in terms of memory and/or time), while still returning the requested accuracy. In Section 7, we summarise our findings and propose possible future directions.

2 TEST ENVIRONMENT

We start by describing our test environment, the software packages we use, and our test examples. The characteristics of the test machine that we use in our numerical experiments are summarized in Table 1.

Table 1. Test Machine Characteristics

CPU	Two Intel Core i7-7700 3.6 GHz Quad Core processors
Memory	16 GB
Compiler	gfortran version 7.4.0 with options -O3 -fopenmp
BLAS/LAPACK	Intel MKL Version 2020.0.1

The serial HSL QR package MA49 [29] 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 numerical experiments, we employ SuiteSparseQR (Version 2.0.9) with default settings and COLAMD ordering for sparsity [16]. The Fortran interface that we use here was also used in Reference [25]. We note that, for sparse matrices, SuiteSparseQR appears in MATLAB as qr and as x = A\b for rectangular systems and thus is widely used. In Section 6.4, we report on using multi-precision arithmetic, and here we employ MA49, because (unlike SuiteSparseQR) it is available in both double and single precision versions.

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

$$ratio = \frac{\|A^T r\|_2/\|r\|_2}{\|A^T b\|_2/\|b\|_2}. \tag{5}$$

The convergence tolerances δ_1 and δ_2 are set to 10^{-8} and 10^{-6}, respectively. Note that for our test examples, the residual norm $\|r\|_2$ is not small and it is Equation (5) that controls the termination of LSMR.

When working with the augmented system, we employ preconditioned GMRES and MINRES. The implementations we use are MI24 (Version 1.3.1) and HSL_MI32 (Version 1.0.0) from the HSL mathematical software library, with the convergence tolerance for both codes set to 10^{-7}.

In our experiments, the vector b is taken to be the vector of 1's and 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 ith unit vector). The entries of AD are at most one in absolute value. For simplicity of notation, D is omitted from our discussions.

With the exception of PDE1 (which comes from the CUTEst linear program test set [23]), our test problems, which are given in Table 2, are from the SuiteSparse Matrix Collection.[4] If necessary, a test matrix is transposed to give an over determined system ($m > n$). The problems are a subset of those used in the study [25]. They were selected because they are real rectangular matrices of full rank that contain a small number of dense rows, they are sufficiently large and challenging to demonstrate the effectiveness of the algorithms considered in this article (all have an R factor with more than 10^6 entries), but, except for PDE1, they can still be solved in our test environment using QR applied to A. PDE1 was included, because it is a large problem that illustrates that a single dense row can cause real difficulties. The problems were also chosen to have a range of row

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

Table 2. Test Examples

Identifier	m	n	m_d	n_{ds}	*dense*	$\kappa(A)$	$\kappa(A_s)$
lp_fit2p	13,525	3,000	25	0	1.000	5.021×10^4	7.200×10^2
sctap1-2b	33,858	15,390	34	0	0.005	1.152×10^3	1.842×10^2
sctap1-2r	63,426	28,830	34	0	0.005	1.570×10^3	1.842×10^2
south31	36,321	18,425	5	0	0.951	5.062×10^5	7.102×10^4
PDE1	271,792	270,595	1	0	0.670	NS	
aircraft	7,517	3,754	17	4	0.200	2.265×10^3	7.681×10^3
sc205-2r	62,423	35,213	8	1	0.005	1.089×10^3	1.493×10^5
scagr7-2b	13,847	9,743	7	1	0.184	1.849×10^3	1.843×10^5
scagr7-2br	46,679	32,847	7	1	0.184	5.978×10^3	1.780×10^5
scrs8-2r	27,691	14,364	22	7	0.143	9.628×10^3	1.383×10^5
scsd8-2r	60,550	8,650	50	5	0.100	7.246×10^2	1.198×10^5

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. $\kappa(A)$ is the condition number of (scaled) A. For the examples in the top part of the table, $\kappa(A_s)$ is the condition number of A_s; for the other examples it is the condition number of the regularized matrix $\begin{bmatrix} A_s \\ \alpha I_n \end{bmatrix}$ with $\alpha = 1.0^{-5}$. NS indicates insufficient memory to compute the condition numbers.

densities. In our experiments, the m_d dense rows are identified using the variant of the approach of Meszaros [32] described in Reference [42] (with the density parameter set to 0.05). In Table 2, n_{ds} denotes the number of null columns in the sparse row block A_s. *dense* is the density of the densest row in A (that is, the ratio of the largest number of entries in a row of A to n); this ranges from 0.05 to 1. $\kappa(A)$ is the condition number of A (after scaling). For the examples with $n_{ds} = 0$, $\kappa(A_s)$ is the condition number of A_s; otherwise, it is the condition number of the regularized matrix $\begin{bmatrix} A_s \\ \alpha I_n \end{bmatrix}$ with $\alpha = 10^{-5}$. Here, and elsewhere, for $n \geq 1$, I_n denotes the $n \times n$ identity matrix. The condition numbers are computed using MATLAB (the R factor is first computed `R = qr(A)` and then the largest and smallest singular values of the R are determined using `svds`).

In Table 3, we report the results of employing SuiteSparseQR on our test machine to solve the LS problems with no special handling of the dense rows. `PDE1` is omitted, because the memory requirements cause an error return. These results are included for later comparison purposes.

3 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 (with no computational results) in the 1982 paper of Heath [27]. Assume that the sparse block A_s is of full rank and start by computing the QR factorization

$$A_s P_s = Q_s \begin{bmatrix} R_s \\ 0 \end{bmatrix} \qquad \text{and} \qquad b_s = Q_s \begin{bmatrix} c_s \\ d_s \end{bmatrix}. \tag{6}$$

The solution y of the sparse LS problem

$$\min_y \|A_s y - b_s\|_2^2, \tag{7}$$

is then found by solving

$$R_s P_s^T y = c_s. \tag{8}$$

Table 3. QR Results for the Test Problems

Identifier	$nnz(R)$	$flops$	$\|x\|_2$	$\|r\|_2$	$ratio$	Time
lp_fit2p	4.502×10^6	5.687×10^{10}	1.689×10^1	1.105×10^2	3.841×10^{-9}	0.86
sctap1-2b	8.532×10^6	7.572×10^{10}	8.171×10^1	1.237×10^2	3.375×10^{-13}	1.67
sctap1-2r	2.952×10^7	4.973×10^{11}	1.117×10^2	1.694×10^2	5.603×10^{-13}	14.0
south31	1.618×10^8	3.839×10^{12}	2.748×10^1	1.881×10^2	3.008×10^{-13}	48.4
aircraft	3.108×10^6	8.287×10^9	2.000	8.660×10^1	1.514×10^{-14}	0.15
sc205-2r	2.331×10^7	1.374×10^{11}	3.478×10^2	2.034×10^2	8.560×10^{-14}	3.00
scagr7-2b	2.388×10^6	8.961×10^9	1.387×10^2	6.068×10^1	3.561×10^{-13}	0.05
scagr7-2br	2.650×10^7	3.414×10^{11}	5.693×10^2	1.132×10^2	1.665×10^{-12}	1.13
scrs8-2r	2.218×10^7	1.989×10^{11}	6.436×10^3	1.354×10^2	9.289×10^{-12}	6.47
scsd8-2r	7.900×10^6	9.999×10^{10}	2.837	2.461×10^2	1.210×10^{-14}	1.78

$nnz(R)$ is the number of entries in the R factor of A and $flops$ is the number of floating-point operations to compute it using SuiteSparseQR. *ratio* is given by Equation (5). The SuiteSparseQR solution time (denoted Time) is in seconds.

Let the solution of Equation (2) be $x = y + z$; it remains to determine z. Using Equations (6) and (8),

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

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

$$\min_z \left\| \begin{bmatrix} R_s P_s^T \\ A_d \end{bmatrix} z - \begin{bmatrix} 0 \\ r_d \end{bmatrix} \right\|_2^2. \tag{9}$$

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 (9) becomes that of finding the minimum-norm solution of the under determined $m_d \times (n + m_d)$ system

$$\begin{bmatrix} K_d & I_{m_d} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = r_d. \tag{10}$$

This leads to Algorithm 1 for solving Equatin (2) (see Algorithm 3 of Reference [27]).

ALGORITHM 1: QR with updating for solving the sparse-dense LS problem (2)

1: Compute the sparse QR factorization $\begin{bmatrix} A_s P_s & b_s \end{bmatrix} = Q_s \begin{bmatrix} R_s & c_s \\ 0 & d_s \end{bmatrix}$.
2: Solve $P_s R_s^T K_d^T = A_d^T$.
3: Solve $R_s P_s^T y = c_s$.
4: Form $r_d = b_d - A_d y$.
5: Compute the minimum norm solution of (10).
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, 3, and 6); once Equation (6) is performed, Q_s 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 Equation (10) can be efficiently computed using the LAPACK routine `_getsls`. Observe that the minimum norm problem (10) is the same as the following problem of size $(m_d + n) \times m_d$:

$$\min_u \left\| \begin{bmatrix} K_d \\ I_{m_d} \end{bmatrix} v - \begin{bmatrix} 0 \\ r_d \end{bmatrix} \right\|_2^2, \quad u = K_d^T v$$

(see, for example, References [28, 37]); both problems have the normal equations $(I_{m_d} + K_d K_d^T)v = r_d$.

3.1 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 [42]. 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, 3, 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. In the following two subsections, we consider two approaches for handling null columns in A_s that allow us to use an unmodified QR package: matrix stretching and regularization.

3.1.1 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 [26] 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 nonzero 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 [41]. 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 [42]. The idea here is to select a small subset of the rows of A_d that contain entries in the columns of A_s that are null. Sparse stretching

is applied to each of the rows in this subset. The stretched rows are sparse so they are added to an enlarged sparse row block $\widetilde{A}_s$ (which incorporates A_s and the stretched rows) while the remaining dense rows are moved to a dense block $\widetilde{A}_d$ 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. The dense rows in $\widetilde{A}_d$ can be handled by applying the updating Algorithm 1 to the partially stretched problem. In the event that $\widetilde{A}_s$ is rank deficient (or highly ill conditioned) after partial stretching, further rows of A_d may be stretched before updating is employed.

3.1.2 Regularization. An alternative approach to handling null columns in A_s is to use regularization (see, for example, Reference [36]). This increases the row dimension by replacing Equation (2) with the regularized (or damped) LS problem,

$$\min_x \|Ax - b\|_2^2 + \|\alpha x\|_2^2 = \min_x \left\| \begin{bmatrix} A_s \\ \alpha I_n \\ A_d \end{bmatrix} x - \begin{bmatrix} b_s \\ 0 \\ b_d \end{bmatrix} \right\|_2^2 = \min_x \left\| \widetilde{A}x - \widetilde{b} \right\|_2^2, \tag{11}$$

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 $\widetilde{A}$ 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 [36] 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 6.2, we include results for regularization combined with updating using a range of values of α.

4 HYBRID APPROACH COMBINING QR WITH AN ITERATIVE SOLVER

So far, we have considered purely direct approaches. We now consider a hybrid approach that involves a QR factorization and an iterative solver. In their paper on QR factorizations for LS problems, Avron et al. [5] 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 [33] or LSMR [20]) applied to the original problem. Again, success of the QR step requires A_s to be of full rank and not too ill conditioned. When A_s has full rank, fast convergence of LSQR preconditioned with R_s can be expected, because the preconditioned matrix has only m_d distinct singular values. In exact arithmetic, convergence should be in at most $m_d + 1$ iterations. If A_s contains null columns, then we can extend the applicability of the hybrid approach, either by first applying partial stretching or, for more general rank-deficient A_s, by employing regularization before the QR step.

In Reference [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 3 and 6 (single right-hand side) and a solve with R_s^T at Step 2 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 (m_d is 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. If A is nearly square, then memory requirements can reduced by storing the Householder transformations used to compute Q_s. However, if m_s is much larger than n, then this does not offer significant savings [22].

5 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

$$\begin{bmatrix} I_m & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \qquad r = b - Ax. \tag{12}$$

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

$$\begin{bmatrix} f^{(i)} \\ g^{(i)} \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix} - \begin{bmatrix} I_m & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} r^{(i)} \\ x^{(i)} \end{bmatrix} = \begin{bmatrix} b - r^{(i)} - Ax^{(i)} \\ -A^T r^{(i)} \end{bmatrix}. \tag{13}$$

(2) Solve for the corrections

$$\begin{bmatrix} I_m & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} \delta r^{(i)} \\ \delta x^{(i)} \end{bmatrix} = \begin{bmatrix} f^{(i)} \\ g^{(i)} \end{bmatrix}. \tag{14}$$

(3) Update the solution to the augmented system

$$\begin{bmatrix} r^{(i+1)} \\ x^{(i+1)} \end{bmatrix} = \begin{bmatrix} \delta r^{(i)} \\ \delta x^{(i)} \end{bmatrix} + \begin{bmatrix} r^{(i)} \\ x^{(i)} \end{bmatrix}. \tag{15}$$

In this way, the solution $x^{(i)}$ and residual $r^{(i)}$ are simultaneously refined. If the QR factorization of A has been computed, then Björck showed that Equation (14) can be solved by reusing the factors. To introduce our notation, consider the augmented system

$$\begin{bmatrix} I_m & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} w \\ t \end{bmatrix}. \tag{16}$$

Using the QR factorization (3), we have

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

so that

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

where

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

The component e is found by solving

$$PR^T e = t,$$

and finally v is the solution of

$$RP^T v = c - e.$$

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). Assume a conformal partitioning of u and w. Performing the QR factorization (6) of the sparse block A_s, we obtain

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

where P_s is the permutation from Equation (6) and

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

Setting $z = R_s P_s^T v$ and $P_s R_s^T K_d^T = A_d^T$, we have

$$\begin{bmatrix} I_n & 0 & I_n \\ 0 & I_{m_d} & K_d \\ I_n & K_d^T & 0 \end{bmatrix} \begin{bmatrix} e_s \\ u_d \\ z \end{bmatrix} = \begin{bmatrix} c_s \\ w_d \\ p \end{bmatrix},$$

where $P_s R_s^T p = t$. The first block gives $z = c_s - e_s$ and eliminating e_s from the above equation gives

$$\begin{bmatrix} I_{m_d} & K_d \\ K_d^T & -I_n \end{bmatrix} \begin{bmatrix} u_d \\ z \end{bmatrix} = \begin{bmatrix} w_d \\ p - c_s \end{bmatrix}.$$

Finally, eliminating z we obtain

$$(I_{m_d} + K_d K_d^T) u_d = w_d + K_d (p - c_s).$$

These are the normal equation of the LS problem

$$\min_{u_d} \left\| \begin{bmatrix} K_d^T \\ I_{m_d} \end{bmatrix} u_d - \begin{bmatrix} p - c_s \\ w_d \end{bmatrix} \right\|_2^2. \tag{17}$$

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

$$\begin{bmatrix} I_{m_s} & 0 & A_s \\ 0 & I_{m_d} & A_d \\ A_s^T & A_d^T & 0 \end{bmatrix} \begin{bmatrix} u_s \\ u_d \\ v \end{bmatrix} = \begin{bmatrix} w_s \\ w_d \\ t \end{bmatrix}, \tag{18}$$

as Algorithm 2.

ALGORITHM 2: Solve the sparse-dense augmented system (18)

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 $P_s R_s^T K_d^T = A_d^T$.
3: Solve $P_s R_s^T p = t$.
4: Solve the LS problem (17) to obtain u_d.
5: Form $e_s = p - K_d^T u_d$.
6: Solve $R_s P_s^T v = c_s - e_s$.
7: Set $u_s = Q_s \begin{bmatrix} e_s \\ d_s \end{bmatrix}$.
8: Return u_s, u_d and v.

It requires one solve with R_s, $m_d + 1$ solves with R_s^T plus one multiplication 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

$$A_{aug} y = b_{aug}, \tag{19}$$

where

$$A_{aug} = \begin{bmatrix} I_{m_s} & 0 & A_s \\ 0 & I_{m_d} & A_d \\ A_s^T & A_d^T & 0 \end{bmatrix}, \qquad y = \begin{bmatrix} r_s \\ r_d \\ x \end{bmatrix}, \qquad b_{aug} = \begin{bmatrix} b_s \\ b_d \\ 0 \end{bmatrix}. \tag{20}$$

Algorithm 2 simplifies, because $t = 0$ implies $p = 0$, reducing the number of solves with R_s^T to m_d. Observe that K_d is independent of the right-hand side vector. Thus, once an initial solution $y^{(0)}$ has been computed, K_d can be reused, limiting the amount of work needed to perform refinement. 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.

5.1 Augmented Approach with Krylov Subspace Refinement

Suppose now that A_s is rank deficient. We can use a hybrid method that first solves the augmented system corresponding to the regularized LS problem and then uses a preconditioned Krylov subspace solver (such as GMRES or MINRES) as a refinement algorithm to recover the solution of the original system. We emphasize that the regularization is used only in the construction of the preconditioner and starting vector for the iterative method, which is applied to the unregularized augmented system (Equation (19)) (note the iterative solver is not invariant with respect to the regularization). A straightforward choice is the simple $\widetilde{R}_s$-block diagonal preconditioner

$$\begin{bmatrix} I_m & 0 \\ 0 & \widetilde{R}_s^T \widetilde{R}_s \end{bmatrix} = \begin{bmatrix} I_m & 0 \\ 0 & \widetilde{R}_s^T \end{bmatrix} \begin{bmatrix} I_m & 0 \\ 0 & \widetilde{R}_s \end{bmatrix} = M^T M, \tag{21}$$

where $\widetilde{R}_s$ is the computed R factor of the regularized sparse matrix $\widetilde{A}_s$ This could be used for left preconditioning or, using it as a split preconditioner to retain symmetry, gives the preconditioned augmented matrix

$$M^{-T} \begin{bmatrix} I_m & A \\ A^T & 0 \end{bmatrix} M^{-1} = \begin{bmatrix} I_m & A\widetilde{R}_s^{-1} \\ \widetilde{R}_s^{-T} A^T & 0 \end{bmatrix}. \tag{22}$$

Each application of the preconditioner (and thus each iteration of the Krylov subspace solver) requires a solve with $\widetilde{R}_s$ and with $\widetilde{R}_s^T$. If there is no regularization, then the nonzero eigenvalues

of Equation (22) are 1 and $\frac{1}{2}(1 \pm \sqrt{1+4\sigma})$, where $n - m_d$ of the n the singular values σ of AR_s^{-1} are equal to 1, and so we can expect convergence of a Krylov method to require approximately $3 + 2m_d$ iterations. The numerical results for a small regularization parameter given in the lower half of Table 10 appear to confirm this dependence on m_d, in particular, see the last two matrices `scrs8-2r` and `scsd8-2r`.

The hybrid approach is summarized as Algorithm 3, where we use the notation (20) and define

$$y^{(i)} = \begin{bmatrix} r_s^{(i)} \\ r_d^{(i)} \\ x^{(i)} \end{bmatrix}, \qquad \delta y^{(i)} = \begin{bmatrix} \delta r_s^{(i)} \\ \delta r_d^{(i)} \\ \delta x^{(i)} \end{bmatrix}, \qquad s^{(i)} = \begin{bmatrix} f_s^{(i)} \\ f_d^{(i)} \\ g^{(i)} \end{bmatrix}. \tag{23}$$

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

ALGORITHM 3: Solve the sparse-dense LS problem (2) using the augmented system hybrid approach with regularization and Krylov solver refinement

1: Apply Algorithm 2 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 $\widetilde{R}_s$ 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 $\widetilde{R}_s$-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**

5.2 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 (12) [10]. They propose trying to reduce the overall solution cost by performing the QR factorization using low precision arithmetic. The reduction in cost may be in terms of time and/or memory requirements. Their GMRES-LSIR algorithm then solves (12) 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: the unit roundoff is 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,

ALGORITHM 4: Solve the sparse-dense LS problem (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 2 to the sparse-dense LS problem (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**

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 4 extends the approach to sparse-dense LS problems with A_s of full rank. The notation is as defined in Equations (20) and (23). As before, MINRES could be used in place of GMRES.

If A_s is rank deficient, then we can derive a multi-precision version of Algorithm 3 by choosing a regularization parameter α and replacing Steps 1 and 2 of Algorithm 4 as follows:

1: Apply Algorithm 2 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 $\widetilde{R}_s$ 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.

6 NUMERICAL EXPERIMENTS

In this section, we look at the performance of the different approaches that we have presented when applied to practical applications.

6.1 A_s Full Rank

We first present results for the problems in the top half of Table 3 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 hybrid QR direct-iterative method of Section 4.

Table 4. A Performance Comparison of QR-based Approaches When A_s Is Full Rank

Identifier	Approach	$\widetilde{m}$	$\widetilde{n}$	$nnz(R_s)$	$flops$	$iters$	$ratio$
lp_fit2p	Updating	13,500	3,000	3.000×10^3	4.050×10^4		5.570×10^{-11}
	Stretching	50,284	39,759	7.478×10^6	2.782×10^{11}		3.821×10^{-9}
	QR+LSMR	13,500	3,000	3.000×10^3	4.050×10^4	44	9.734×10^{-8}
sctap1-2b	Updating	33,826	15,390	1.111×10^5	4.063×10^6		3.348×10^{-12}
	Stretching	45,172	26,704	2.522×10^6	6.467×10^9		8.210×10^{-14}
	QR+LSMR	33,826	15,390	1.111×10^5	4.063×10^6	43	8.420×10^{-7}
sctap1-2r	Updating	63,392	28,830	2.084×10^5	7.618×10^6		1.427×10^{-11}
	Stretching	84,596	50,000	5.571×10^6	1.943×10^{10}		1.677×10^{-13}
	QR+LSMR	63,392	28,830	2.084×10^5	7.618×10^6	40	8.459×10^{-7}
south31	Updating	36,317	18,425	1.985×10^6	9.400×10^8		7.095×10^{-15}
	Stretching	36,810	18,914	5.905×10^6	1.888×10^{10}		4.675×10^{-15}
	QR+LSMR	36,317	18,425	1.985×10^6	9.400×10^8	6	9.704×10^{-15}
PDE1	Updating	271,791	270,595	1.392×10^7	7.813×10^9		1.337×10^{-11}
	Stretching	362,388	361,191	3.349×10^7	3.336×10^{10}		1.054×10^{-10}
	QR+LSMR	271,791	270,595	1.392×10^7	7.813×10^9	2	2.122×10^{-9}

$\widetilde{m}$ and $\widetilde{n}$ are the row and column dimensions of the matrix that are 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. $iters$ denotes the number of LSMR iterations performed (QR + LSMR only). $ratio$ is given by Equation (5).

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 4, we report statistics for the QR factorization (and for QR+LSMR, the number of iterations required to achieve convergence is given); timings are presented in Table 5. 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, 3, and 6 of Algorithm 1 plus the time for solving (10). For QR+LSMR, Solve is the time to run preconditioned LSMR and, for Stretching, it is the time to taken to solve Equation (4) with the R factor for the stretched system. Each method successfully computed the solution x and residual r with norms identical to those reported in Table 3 (for problem PDE, each computed $\|x\|_2 = 4.282 \times 10^2$ and $\|r\|_2 = 3.030 \times 10^2$).

As expected, Updating is faster than QR+LSMR, with the additional cost dependent on the number of LSMR iterations (which increases with m_d). 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 dense row (which is 67% full). 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. 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. In general, QR after 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. Overall, our experiments suggest that

Table 5. A Comparison Timings (in Seconds) for QR-based Approaches When A_s Is Full Rank

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

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.

if A_s is full rank, then using Algorithm 1 (Updating) is the best of the approaches considered and it offers very significant savings compared to the results of Table 3 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 s to less than 0.1 s. Furthermore, we observe that it is not necessary for the densest rows to be close to being full for it to be advantageous to treat them separately. This is illustrated by examples sctap1-2b and sctap1-2r for which the densest row is only 5% full.

6.2 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 $\widetilde{A}_s$ 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 $\widetilde{R}_s$ is its R factor.

In Table 6, we report results for partial stretching combined with Updating (Algorithm 1 is applied to the partially stretched problem). Here n_{ds} is the number of rows that are stretched, which is equal to the number of null columns in 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 reported $\|x\|_2$ and $\|r\|_2$ are for the original system; they agree with those in Table 3. The only problem for which partial stretching with updating as currently implemented is 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 $\widetilde{A}_s$ has rank deficiency 1 and it is necessary to stretch additional rows to make $\widetilde{A}_s$ full rank.

Table 6. Results for Partial Stretching Combined with Updating When A_s Is Rank Deficient

Identifier	n_{ds}	$\widetilde{m}$	$\widetilde{n}$	$nnz(\widetilde{R}_s)$	$flops$	$\|x\|_2$	$\|r\|_2$	$ratio$
aircraft	4	10,517	6,754	4.719×10^4	9.333×10^5	2.000	8.660×10^1	2.879×10^{-14}
sc205-2r	1	64,023	36,813	3.175×10^5	8.253×10^6	3.478×10^2	2.033×10^2	6.728×10^{-11}
scagr7-2b	1	15,127	11,023	1.186×10^5	5.029×10^6	1.387×10^2	6.068×10^1	8.845×10^{-12}
scagr7-2br	1	50,999	37,167	4.673×10^5	1.881×10^7	5.693×10^2	1.132×10^2	4.343×10^{-11}
scrs8-2r	7	32,820	19,493	4.242×10^5	4.655×10^7	6.435×10^3	1.354×10^2	2.457×10^{-12}

n_{ds} denotes the number of rows that are stretched; $\widetilde{m}$ and $\widetilde{n}$ are the row and column dimensions of the matrix that are factorized using SuiteSparseQR. $nnz(\widetilde{R}_s)$ is the number of entries in $\widetilde{R}_s$ and $flops$ is the number of floating-point operations needed to compute it. $ratio$ is given by Equation (5).

Table 7. Results for Regularization Combined with Updating for a Range of Values of the Regularization Parameter α

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

$Ratio$ is given by Equation (5).

Regularization requires the selection of an appropriate regularization parameter α. Table 7 illustrates that, if regularization is combined with Updating (that is, Algorithm 1 applied to Equation (11)), 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 $\widetilde{R}_s$ 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 Figures 1 and 2, for $10^{-7} \leq \alpha \leq 1$ we plot the number of LSMR iterations needed for this approach and the values of *ratio* given by Equation (5), respectively. As we expect, the iteration count 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 and *ratio* also becomes large for such values. 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.

We next consider combining LSMR with partial stretching and with regularization. Results are given in Tables 8 and 9. For the approach denoted by PStretching, we apply partial stretching, compute the QR factorization of $\widetilde{A}_s$, use the R factor $\widetilde{R}_s$ as a preconditioner for LSMR applied to

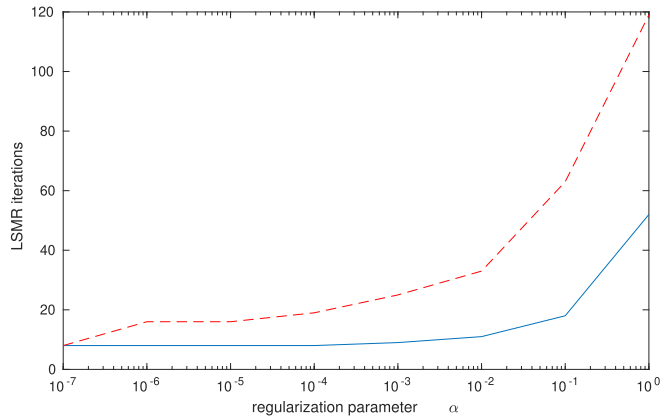

Fig. 1. LSMR iteration counts for a range of values of the regularization parameter α for problems scagr7_2b (blue solid line) and scrs8-2r (red dashed line).

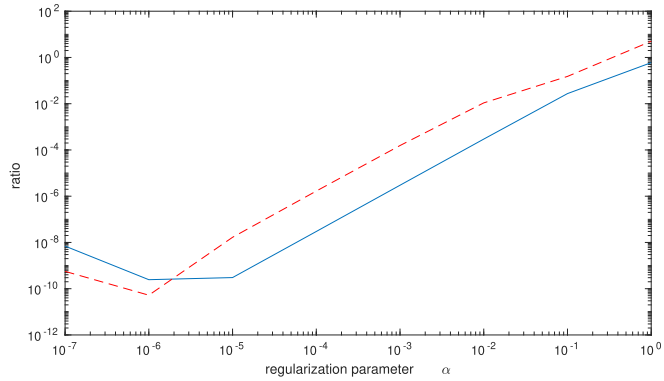

Fig. 2. Values of *ratio* for a range of values of the regularization parameter α for problems scagr7_2b (blue solid line) and scrs8-2r (red dashed line).

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. We report *ratio* for the original problem. Both approaches are successful (the computed $\|x\|_2$ and $\|r\|_2$ are consistent with Table 3 and so are not explicitly reported) 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. Observe that *ratio* for partial stretching combined with Updating (Table 6) is generally smaller than for PStretching.

Next, we illustrate how the number of dense rows affects the performance of preconditioned LSMR. We have already seen in Table 5 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

Table 8. A Performance Comparison of Employing $\widetilde{R}_s$ Computed Using Partial Stretching and Regularization as a Preconditioner for LSMR When A_s Is Rank Deficient

Identifier	Approach	$\widetilde{m}$	$\widetilde{n}$	$nnz(\widetilde{R}_s)$	$flops$	$iters$	$ratio$
aircraft	PStretching	10,517	6,754	4.719×10^4	9.333×10^5	9	4.947×10^{-12}
	Regularization	11,271	3,754	3.754×10^3	3.376×10^4	7	5.476×10^{-7}
sc205-2r	PStretching	64,023	36,813	3.175×10^5	8.253×10^6	6	9.983×10^{-9}
	Regularization	97,636	35,213	2.704×10^5	1.209×10^7	7	1.002×10^{-8}
scagr7-2b	PStretching	15,127	11,023	1.186×10^5	5.029×10^6	7	2.129×10^{-12}
	Regularization	23,590	9,743	6.027×10^4	3.667×10^6	8	3.979×10^{-9}
scagr7-2br	PStretching	50,999	37,167	4.673×10^5	1.881×10^7	7	1.046×10^{-10}
	Regularization	79,526	32,847	2.273×10^5	1.275×10^7	8	3.470×10^{-8}
scrs8-2r	PStretching	32,820	19,493	4.242×10^5	4.655×10^7	7	3.311×10^{-12}
	Regularization	42,055	14,364	8.200×10^4	2.835×10^6	16	3.532×10^{-7}

$\widetilde{m}$ and $\widetilde{n}$ are the row and column dimensions of the matrix that is factorized using SuiteSparseQR. $nnz(\widetilde{R}_s)$ is the number of entries in the factor $\widetilde{R}_s$, and $flops$ is the number of floating-point operations needed to compute it. $iters$ denotes the number of LSMR iterations performed. $ratio$ is given by Equation (5).

Table 9. A Comparison of Timings (Seconds) of Employing $\widetilde{R}_s$ Computed Using Partial Stretching and Regularization as a Preconditioner for LSMR When A_s Is Rank Deficient

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
scagr7-2br	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

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.

entries) and A_s is rank deficient. 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 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 Figure 3, 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.

6.3 Numerical Results for the Augmented Approach

We now turn our attention to the augmented approach of Section 5. In Table 10, 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 (19) with initial solution $x^{(0)} = 0$ and for Algorithm 3 (with GMRES and MINRES as the iterative refinement solver). For the latter,

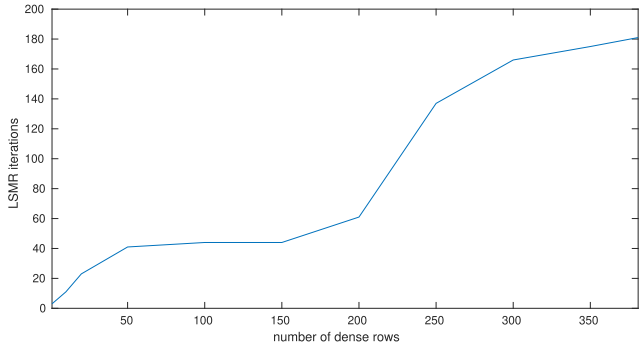

Fig. 3. The LSMR iteration count for the (modified) problem south31 as the number m_d of dense rows increases.

Table 10. A Performance Comparison of Split-preconditioned GMRES and MINRES Used Directly to Solve the Augmented System (19) and for Refinement (Algorithm 3 Using GMRES and MINRES with $it_{max} = 1$)

Identifier	Approach	*iters*	$ratio_{init}$	*ratio*
aircraft	GMRES/MINRES	18/24		$1.058 \times 10^{-14}/4.682 \times 10^{-9}$
	Algorithm 3+GMRES/MINRES	18/22	1.274×10^{-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 3+GMRES/MINRES	20/30	2.240×10^{-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 3+GMRES/MINRES	22/40	2.949×10^{-4}	$8.961 \times 10^{-13}/1.391 \times 10^{-13}$
scagr7-2br	GMRES/MINRES	22/40		$3.151 \times 10^{-10}/1.824 \times 10^{-9}$
	Algorithm 3+GMRES/MINRES	22/40	3.181×10^{-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 3+GMRES/MINRES	42/98	1.246×10^{-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 3+GMRES/MINRES	40/88	1.070×10^{-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 3+GMRES/MINRES	20/24	1.298×10^{-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 3+GMRES/MINRES	16/30	4.514×10^{-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 3+GMRES/MINRES	18/38	2.985×10^{-10}	$1.282 \times 10^{-12}/1.282 \times 10^{-12}$
scagr7-2br	GMRES/MINRES	16/28		$1.943 \times 10^{-8}/5.031 \times 10^{-8}$
	Algorithm 3+GMRES/MINRES	18/46	3.594×10^{-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 3+GMRES/MINRES	30/48	2.429×10^{-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 3+GMRES/MINRES	36/100	2.685×10^{-6}	$1.793 \times 10^{-14}/8.231 \times 10^{-14}$

In each case, the sparse block A_s is rank deficient and the QR factorization of the regularized sparse matrix $\widetilde{A}_s$ is computed using SuiteSparseQR. Results in the upper (respectively, lower) half of the table are for the regularization parameter $\alpha = 10^{-2}$ (respectively, $\alpha = 10^{-5}$). *iters* denotes the number of GMRES or MINRES iterations performed. *ratio* is given by Equation (5) and $ratio_{init}$ by Equation (24).

Table 11. Timings (in Seconds) for Split-preconditioned GMRES and MINRES Used Directly to Solve the Augmented System (19) and for Refinement (Algorithm 3 Using GMRES and MINRES with $it_{max} = 1$)

Identifier	Approach	Symbolic	Numeric	Augmented	Iterative	Total
aircraft	GMRES/MINRES	0.001	0.001		0.006/0.006	0.008/0.008
	Algorithm 3+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 3+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 3+GMRES/MINRES	0.004	0.003	0.008	0.012/0.023	0.027/0.040
scagr7-2br	GMRES/MINRES	0.014	0.012		0.045/0.082	0.071/0.107
	Algorithm 3+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 3+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 3+GMRES/MINRES	0.008	0.007	0.009	0.072/0.199	0.097/0.223

Symbolic and Numeric are the times for the symbolic analysis and numerical factorization phases of SuiteSparseQR, respectively. Augmented is the time within Algorithm 3 after the QR factorization has been computed to solve the augmented system (19) and Iterative is the time for GMRES or MINRES. Total denotes the total solution time. The regularization parameter is $\alpha = 10^{-5}$.

we give the ratio (5) before refinement, that is,

$$ratio_{init} = \frac{\|A^T r^{(0)}\|_2 / \|r^{(0)}\|_2}{\|A^T b\|_2 / \|b\|_2}, \tag{24}$$

and after a single refinement step ($it_{max} = 1$). Timings are given in Table 11. Here "Augmented" denotes the time within Algorithm 3 for solving the augmented system (19) using the computed QR factorization of the regularized problem (that is, it is the time for Steps 2 to 8 of Algorithm 2). Each approach is successful, 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 10), the stopping criterion that we employed 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. A comparison of the total timings reported in Table 11 with those for regularization plus LSMR given in Table 9 indicates that, in general, the latter is more efficient, because the iterative solver time is less.

6.4 Multi-precision Results

Finally, to illustrate the use of multi-precision arithmetic, we set u_f to single precision and $u = u_r$ to double precision. Results are presented in Table 12 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}. As already noted, these results are computed using the HSL QR package MA49, because it offers single and double versions. The results show that the multi-precision approach is feasible and offers another option for solving sparse-dense LS problems. We do not report timings here, because, in our experiments with MA49, we did not find that the QR factorization time of A_s using the single precision version was significantly

Table 12. Multi Precision Results (Algorithm 4) Obtained Using MA49

Identifier	$iters$	$\|x\|$	$\|r\|$	$ratio_{init}$	$ratio$
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}

The problems in the upper part do not use regularization while those in the lower part use regularization with $\alpha = 10^{-3}$. $iters$ denotes the number of GMRES iterations performed. $it_{max} = 1$. $ratio$ is given by Equation (5) and $ratio_{init}$ by Equation (24).

less than for the double precision version. Furthermore, the GMRES times dominate the total run time, suggesting that to reduce the time the efficiency of the matrix-vector products and the application of the preconditioner require improvement. This and exploring why, in our tests, the single precision version of MA49 does not offer good speedups, is outside the scope of our current study. Our findings with the current software suggests that a key advantage of the single precision approach is that it is possible in some cases to perform the QR factorization in single precision when there is insufficient memory to perform it in double precision, potentially extending the class of problems that can be successfully solved.

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. We have brought together a number of different approaches, with particular emphasis on problems in which the sparse part A_s of the system matrix A is rank deficient. In this case, we have shown that combining basic approaches for sparse-dense LS problems with either partial matrix stretching or regularization is effective. For the former, the solution of the original system can be obtained directly using the sparse QR solver combined with updating but the initial stretching step currently adds a significant overhead. Regularization is much more straightforward to implement and less expensive, even though it may be necessary to use a hybrid 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 the potential for multi-precision solvers, which extends the class of problems for which a QR factorization of the sparse row block can be computed. While all the proposed approaches have been shown to be effective, no one method is the best choice for all problems.

In this article, we have assumed that a QR factorization of A_s (possibly after stretching or regularization) can be performed. However, memory limitations may mean that for very large problems this is not possible and an iterative approach that avoids a complete QR factorization is needed. A number of approaches to computing a LS preconditioner based on incomplete orthogonal factorizations have been proposed (including References [7–9, 30, 31, 34, 35, 46]); no comparisons have been made between them. This is probably because, with the exception of the MIQR package of Li

and Saad [31] there is (as far as we are aware) no software available (and the implementations are nontrivial). In their study of preconditioners for LS problems, Gould and Scott [24, 25] 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 as the HSL software package HSL_MI35 [38, 40]), and thus we have not experimented further with it here. Designing, developing and implementing algorithms (and underlying theory) for efficient and reliable incomplete orthogonal factorization preconditioners remains an open challenge.

Other possible future directions include designing and developing more efficient implementations of the algorithm that performs partial stretching to make the approach really viable. When there are several rows that require stretching, our plan is 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 costs. More work is also needed to develop efficient multi-precision solvers. We have shown that using multi-precision is feasible but its (optional) use needs to be built into the solver package. Furthermore, a QR-based solver that efficiently handles dense rows for the user is needed. We have shown that it is possible to do this by writing code around an existing solver but it would be more efficient and much more user-friendly to design the software in the first place to allow for dense rows. Importantly, a row does not need to be very dense for it to be worthwhile to handle it as dense.

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.

ACKNOWLEDGMENTS

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. We thank the referees for their constructive feedback and insightful comments that have led to significant improvements to this article.

REFERENCES

[1] M. Adlers. 2000. *Topics in Sparse Least Squares Problems.* Technical Report. Department of Mathematics, Linköping University, Linköping, Sweden.
[2] M. Adlers and Å. Björck. 2000. Matrix stretching for sparse least squares problems. *Numer. Lin. Algebr. Appl.* 7, 2 (2000), 51–65.
[3] F. L. Alvarado. 1997. Matrix enlarging methods and their application. *BIT Numer. Math.* 37, 3 (1997), 473–505.
[4] P. R. Amestoy, I. S Duff, and C. Puglisi. 1996. Multifrontal QR factorization in a multiprocessor environment. *Numer. Linear Algebra Appl.* 3, 4 (1996), 275–300.
[5] H. Avron, E. Ng, and S. Toledo. 2009. Using perturbed *QR* factorizations to solve linear least-squares problems. *SIAM J. Matrix Anal. Appl.* 31, 2 (2009), 674–693.
[6] C. Aykanat, A. Pinar, and Ü. V. Çatalyürek. 2002. *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.
[7] Z.-Z. Bai, I. S. Duff, and A. J. Wathen. 2001. A class of incomplete orthogonal factorization methods. I: Methods and theories. *BIT Numer. Math.* 41, 1 (2001), 53–70.
[8] Z.-Z. Bai, I. S. Duff, and J.-F. Yin. 2009. Numerical study on incomplete orthogonal factorization preconditioners. *J. Comput. Appl. Math.* 226, 1 (2009), 22–41.
[9] Z.-Z. Bai and J.-F. Yin. 2009. Modified incomplete orthogonal factorization methods using givens rotations. *Computing* 86, 1 (2009), 53–69.
[10] Å. Björck. 1967. Iterative refinement of linear least squares solutions I. *BIT Numer. Math.* 7, 4 (1967), 257–278.
[11] Å. Björck. 1996. *Numerical Methods for Least Squares Problems.* SIAM, Philadelphia, PA.

[12] A. Buttari. 2013. Fine-grained multithreading for the multifrontal QR factorization of sparse matrices. *SIAM J. Sci. Comput.* 35, 4 (2013), C323–C345.
[13] E. Carson and N. Higham. 2018. Accelerating the solution of linear systems by iterative refinement in three precisions. *SIAM J. Sci. Comput.* 40, 2 (2018), A817–A84.
[14] E. Carson, N. Higham, and S. Pranesh. 2020. Three-precision GMRES-based iterative refinement for least squares problems. *SIAM J. Sci. Comput.* 42, 6 (2020), A4063–A4083.
[15] T. A. Davis. 2011. Algorithm 915, SuiteSparseQR: Multifrontal multithreaded rank-revealing sparse QR factorization. *ACM Trans. Math. Software* 38, 1 (2011), 8:1–8:22.
[16] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng. 2004. Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm. *ACM Trans. Math. Software* 30, 3 (2004), 377–380.
[17] J. Demmel, Y. Hida, E. J. Riedy, and X. S. Li. 2009. Extra-precise iterative refinement for overdetermined least squares problems. *ACM Trans. Math. Software* 35, 4 (2009), 28:1–28:32.
[18] I. S. Duff and J. A. Scott. 2005. Stabilized bordered block diagonal forms for parallel sparse solvers. *Parallel Comput.* 31, 3–4 (2005), 275–289.
[19] M. C. Ferris and J. D. Horn. 1998. Partitioning mathematical programs for parallel solution. *Math. Programming* 80, 1 (1998), 35–62.
[20] D. C.-L. Fong and M. A. Saunders. 2011. LSMR: An iterative algorithm for sparse least-squares problems. *SIAM J. Sci. Comput.* 33, 5 (2011), 2950–2971.
[21] A. George and M. T. Heath. 1980. Solution of sparse linear least squares problems using givens rotations. *Lin. Algebr. Appl.* 34 (1980), 69–83.
[22] J. R. Gilbert, E. Ng, and B. W. Peyton. 1997. Separators and structure prediction in sparse orthogonal factorization. *Lin. Algebr. Appl.* 262 (1997), 82–97.
[23] N. I. M. Gould, D. Orban, and Ph. L. Toint. 2015. CUTEst: A constrained and unconstrained testing environment with safe threads for mathematical optimization. *Comput. Optim. Appl.* 60, 1 (2015), 545–557.
[24] N. I. M. Gould and J. A. Scott. 2015. *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.
[25] N. I. M. Gould and J. A. Scott. 2017. The state-of-the-art of preconditioners for sparse linear least squares problems. *ACM Trans. Math. Softw.* 43, 4 (2017), 36:1–35.
[26] J. F. Grcar. 1990. *Matrix Stretching for Linear Equations.* Technical Report SAND90-8723. Sandia National Laboratories.
[27] M. T. Heath. 1982. Some extensions of an algorithm for sparse linear least squares problems. *SIAM J. Sci. Stat. Comput.* 3, 2 (1982), 223–237.
[28] G. T. Herman, A. Lent, and H. Hurwitz. 1980. A storage-efficient algorithm for finding the regularized solution of a large, inconsistent system of equations. *J. Inst. Math. Appl.* 25, 4 (1980), 361–366.
[29] HSL 2018. HSL. A Collection of Fortran Codes for Large-scale Scientific Computation. Retrieved from http://www.hsl.rl.ac.uk.
[30] A. Jennings and M. A. Ajiz. 1984. Incomplete methods for solving $A^T Ax = b$. *SIAM J. Sci. Stat. Comput.* 5, 4 (1984), 978–987.
[31] N. Li and Y. Saad. 2006. MIQR: A multilevel incomplete QR preconditioner for large sparse least-squares problems. *SIAM J. Matrix Anal. Appl.* 28, 2 (2006).
[32] C. Meszaros. 2007. Detecting dense columns in interior point methods for linear programs. *Comput. Optim. Appl.* 36, 2–3 (2007), 309–320.
[33] C. C. Paige and M. A. Saunders. 1982. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Trans. Math. Softw.* 8, 1 (1982), 43–71.
[34] A. T. Papadopoulos, I. S. Duff, and A. J. Wathen. 2005. A class of incomplete orthogonal factorization methods. II: Implementation and results. *BIT Numer. Math.* 45, 1 (2005), 159–179.
[35] Y. Saad. 1988. Preconditioning techniques for nonsymmetric and indefinite linear systems. *J. Comput. Appl. Math.* 24, 1–2 (1988), 89–105.
[36] M. A. Saunders. 1995. *Cholesky-based Methods for Sparse Least Squares: The Benefits of Regularization.* Technical Report SOL 95-1. Department of Operations Research, Stanford University.
[37] M. A. Saunders. 1995. Solution of sparse rectangular systems using LSQR and Craig. *BIT Numer. Math.* 35, 4 (1995), 588–604.
[38] J. A. Scott and M. Tuma. 2014. HSL_MI28: An efficient and robust limited-memory incomplete cholesky factorization code. *ACM Trans. Math. Softw.* 40, 4 (2014), 24:1–19.
[39] J. A. Scott and M. Tuma. 2017. Solving mixed sparse-dense linear least-squares problems by preconditioned iterative methods. *SIAM J. Sci. Comput.* 39, 6 (2017), A2422–A2437.
[40] J. A. Scott and M. Tuma. 2018. A Schur complement approach to preconditioning sparse least-squares problems with some dense rows. *Numer. Algorithms* 79, 4 (2018), 1147–1168. DOI: 10.1007/s11075-018-0478-2

[41] J. A. Scott and M. Tuma. 2019. Sparse stretching for solving sparse-dense linear least-squares problems. *SIAM J. Sci. Comput.* 41, 3 (2019), A1604–A1625.
[42] J. A. Scott and M. Tuma. 2021. Strengths and limitations of stretching for least-squares problems with some dense rows. *ACM Trans. Math. Softw.* 47, 1 (2021), 1:1–25.
[43] C. Sun. 1995. *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.
[44] C. Sun. 1997. Parallel solution of sparse linear least squares problems on distributed-memory multiprocessors. *Parallel Comput.* 23, 13 (1997), 2075–2093.
[45] R. J. Vanderbei. 1991. Splitting dense columns in sparse linear systems. *Lin. Algebr. Appl.* 152 (1991), 107–117.
[46] X. Wang, K. A. Gallivan, and R. Bramley. 1997. CIMGS: An incomplete orthogonal factorization preconditioner. *SIAM J. Sci. Comput.* 18, 2 (1997), 516–536.
[47] S. N. Yeralan, T. A. Davis, and S. Ranka. 2017. Algorithm 980: Sparse QR factorization on the GPU. *ACM Trans. Math. Software* 44, 2 (2017), 17:1–17:29.

Received October 2020; revised October 2021; accepted October 2021