Science and Technology Facilities Council

Preprint
STFC-P-2023-003

Approximating sparse Hessian matrices using large-scale linear least squares

JM Fowkes, NIM Gould, J Scott

August 2023

Submitted for publication in Numerical Algorithms

Enquiries concerning this report should be addressed to:

RAL Library
STFC Rutherford Appleton Laboratory
Harwell Oxford
Didcot
OX11 0QX

Tel: +44(0)1235 445577
email: library@stfc.ac.uk

Science and Technology Facilities Council reports are available online at:
https://epubs.stfc.ac.uk

Accessibility: a Microsoft Word version of this document (for use with assistive technology) may be available on request.

ISSN 2753-5819

STFC Author Identifiers (ORCIDs)

Author ORCIDs are provided where available.

Jaroslav M Fowkes	0000-0002-8048-4572
Nicholas IM Gould	0000-0002-1031-1588
Jennifer Scott	0000-0003-2130-1091

APPROXIMATING SPARSE HESSIAN MATRICES USING LARGE-SCALE LINEAR LEAST SQUARES

JAROSLAV M. FOWKES*, NICHOLAS I. M. GOULD*, AND JENNIFER SCOTT*†

Abstract. Large-scale optimization algorithms frequently require sparse Hessian matrices that are not readily available. Existing methods for approximating large sparse Hessian matrices have limitations. To try and overcome these, we propose a novel approach that reformulates the problem as the solution of a large linear least squares problem. The least squares problem is sparse but can include a number of rows that contain significantly more entries than other rows and are regarded as dense. We exploit recent work on solving such problems using either the normal equations or an augmented system to derive a robust approach for computing approximate sparse Hessian matrices. Example sparse Hessians from the CUTEst test problem collection for optimization illustrate the effectiveness and robustness of the new method.

Key words. Sparse nonlinear systems, sparse Hessian matrices, sparse linear least squares, sparse direct solvers.

1. Introduction.

Consider the large sparse optimization problem

$$\min_x f(x),$$

where $f(x)$ is a sufficiently smooth function of n variables. Whilst the gradient $g(x) := \nabla f(x)$ is often readily available, the Hessian matrix $H(x) := \nabla^2 f(x)$ is frequently difficult to provide. For example, the backward mode of automatic differentiation enables the gradient of a nonlinear function to be computed at a cost that is a small multiple of the that of evaluating $f(x)$, but the cost of evaluating $H(x)$ using differencing techniques is $\mathcal{O}(n)$ times that of $f(x)$. This is unfortunate because there are important theoretical and practical benefits in having access to the Hessian matrix. The explosion of interest in machine learning and data science algorithms that involve optimizing a function has further emphasised the need for good approximations to Hessian matrices.

Interest in methods for building approximations to $H(x)$ dates back to the 1960s. The focus at that time was on problems involving a small number of variables and consequently on small dense Hessian matrices. Extensions to the sparse case were not successful because either the formulae used generated dense matrices that were impractical for large problems, or imposing sparsity led to potential numerical instability in the approximation algorithms [7, 28, 29, 30]. Attention subsequently turned to limited-memory strategies [20, Chapter 7]. These did not seek to reproduce the Hessian matrix but to incorporate the curvature observed at a number of previous iterates. No attempt was made to impose sparsity.

Our interest is in large-scale problems for which it is essential that sparsity is exploited. The proposed new method formulates the problem as a large-scale linear least squares (LS) problem. In general, this LS problem is sparse but, if the Hessian matrix contains one or more rows with a large number of entries, then the LS matrix has some rows that are regarded as dense. These dense rows make the problem more challenging. Methods for tackling sparse-dense LS problems have been considered, for example, in [3, 5, 9, 23, 24, 26, 27]. Exploiting the work of Scott and Tůma [23, 26], we propose using sparse direct linear equation solvers combined with an iterative method. Recent software from the HSL Mathematical Software Library [16] is used to perform numerical experiments.

The paper is organised as follows. In Section 2, we introduce our proposed new LS formulation. Sparse direct methods for solving this LS problem are considered in Section 3, with an emphasis on the sparse-dense case. In Section 4, we report the results of numerical experiments that illustrate the potential of the new method to be used for approximating large sparse Hessian matrices in practice. Finally, concluding comments are given in Section 5.

2. Least squares formulation.

Consider the twice differentiable function $f(x)$ of n variables x, whose gradient $g(x)$ is known. The challenge is to build approximations $B^{(k)} = \{b_{ij}^{(k)}\}$ of the Hessian

* STFC Rutherford Appleton Laboratory, Harwell Campus, Didcot, Oxfordshire, OX11 0QX, UK

† School of Mathematical, Physical and Computational Sciences, University of Reading, Reading RG6 6AQ, UK. Correspondence to: jennifer.scott@stfc.ac.uk. All authors were supported by EPSRC grant number EP/X032485/1

matrix $H(x)$ at a sequence of given iterates $x^{(k)}$. $H(x^{(k)})$ is an $n \times n$ symmetric matrix and we assume that its sparsity pattern (the locations of the nonzero entries) is known. The approach we propose is based on using the data from a sequence of $m \geq 1$ previous steps to estimate $B^{(k)}$. The idea of using recent difference pairs

$$s^{(l)} := x^{(l)} - x^{(l-1)} \quad \text{and} \quad y^{(l)} := g(x^{(l)}) - g(x^{(l-1)}), \quad l = k-m+1, \ldots, k, \tag{2.1}$$

was initially proposed by Fletcher, Grothey and Leyffer [8]. Their aim was to construct $B^{(k)}$ that best satisfies the multiple *secant conditions* given by

$$B^{(k)} s^{(l)} = y^{(l)}, \qquad l = k-m+1, \ldots, k. \tag{2.2}$$

They did this by solving, for each k, the convex quadratic programming problem

$$\min_{B^{(k)}} \sum_{l=k-m+1}^{k} \|B^{(k)} s^{(l)} - y^{(l)}\|_F^2 \quad \text{such that} \quad B^{(k)} = (B^{(k)})^T \quad \text{and} \quad \mathcal{S}(B^{(k)}) = \mathcal{S}(H(x^{(k)})). \tag{2.3}$$

Here, if W is a matrix with entries $= \{w_{ij}\}$ then $\|W\|_F^2$ denotes its squared Frobenius norm and $\mathcal{S}(W) := \{(i,j) : w_{ij} \neq 0\}$ is its sparsity pattern. Solving the so-called *Constrained Procrustes Problem* (2.3) results in an estimate of the Hessian matrix that is symmetric and whose sparsity is preserved, although positive-definiteness is not guaranteed. Consequently, this technique is useful inside a trust region method where positive-definiteness of the Hessian matrix is not a requirement. Problem (2.3) can be solved using existing well-developed optimization techniques, but for large problems they are computationally prohibitively expensive. Instead, we propose stacking the (unknown) nonzero entries in the upper triangular part of $B^{(k)}$ row-by-row above each other in a vector $z^{(k)}$ of size equal to the number of nonzero entries in the upper triangular part of $B^{(k)}$ (equivalently, the entries in the lower triangular part are stacked column-by-column). In this way, if $nz(B^{(k)})$ denotes the number of nonzero entries in the upper triangular part of $B^{(k)}$, we redefine the problem as a large sparse linear system of equations of size $mn \times nz(B^{(k)})$ given by

$$A^{(k)} z^{(k)} = c^{(k)}. \tag{2.4}$$

Here the matrix $A^{(k)}$ and the vector $c^{(k)}$ are known and depend on the secant conditions (2.2). To illustrate this formulation, consider the following two simple examples.

Example 1 Let $n = 3$ and consider the approximate Hessian matrix with $nz(B^{(k)}) = 4$

$$B^{(k)} = \begin{pmatrix} b_{11}^{(k)} & b_{12}^{(k)} & 0 \\ b_{21}^{(k)} & 0 & b_{23}^{(k)} \\ 0 & b_{32}^{(k)} & b_{33}^{(k)} \end{pmatrix}, \quad \text{with} \quad b_{12}^{(k)} = b_{21}^{(k)} \text{ and } b_{23}^{(k)} = b_{32}^{(k)}.$$

For $m = 2$, the 6×4 linear system (2.4) is

$$\underbrace{\begin{pmatrix} s_1^{(k)} & s_2^{(k)} & 0 & 0 \\ s_1^{(k-1)} & s_2^{(k-1)} & 0 & 0 \\ 0 & s_1^{(k)} & s_3^{(k)} & 0 \\ 0 & s_1^{(k-1)} & s_3^{(k-1)} & 0 \\ 0 & 0 & s_2^{(k)} & s_3^{(k)} \\ 0 & 0 & s_2^{(k-1)} & s_3^{(k-1)} \end{pmatrix}}_{A^{(k)}} \underbrace{\begin{pmatrix} b_{11}^{(k)} \\ b_{12}^{(k)} \\ b_{23}^{(k)} \\ b_{33}^{(k)} \end{pmatrix}}_{z^{(k)}} = \underbrace{\begin{pmatrix} y_1^{(k)} \\ y_1^{(k-1)} \\ y_2^{(k)} \\ y_2^{(k-1)} \\ y_3^{(k)} \\ y_3^{(k-1)} \end{pmatrix}}_{c^{(k)}}.$$

Example 2 Let $n = 4$ and consider the approximate Hessian matrix with $nz(B^{(k)}) = 6$

$$B^{(k)} = \begin{pmatrix} b_{11}^{(k)} & 0 & 0 & b_{14}^{(k)} \\ 0 & 0 & b_{23}^{(k)} & b_{24}^{(k)} \\ 0 & b_{32}^{(k)} & 0 & b_{34}^{(k)} \\ b_{41}^{(k)} & b_{42}^{(k)} & b_{43}^{(k)} & b_{44}^{(k)} \end{pmatrix}, \quad \text{with } b_{14}^{(k)} = b_{41}^{(k)}, b_{23}^{(k)} = b_{32}^{(k)}, b_{24}^{(k)} = b_{42}^{(k)} \text{ and } b_{34}^{(k)} = b_{43}^{(k)}.$$

For $m = 2$, the 8×6 linear system (2.4) is

$$\underbrace{\begin{pmatrix} s_1^{(k)} & s_4^{(k)} & 0 & 0 & 0 & 0 \\ s_1^{(k-1)} & s_4^{(k-1)} & 0 & 0 & 0 & 0 \\ 0 & 0 & s_3^{(k)} & s_4^{(k)} & 0 & 0 \\ 0 & 0 & s_3^{(k-1)} & s_4^{(k-1)} & 0 & 0 \\ 0 & 0 & s_2^{(k)} & 0 & s_4^{(k)} & 0 \\ 0 & 0 & s_2^{(k-1)} & 0 & s_4^{(k-1)} & 0 \\ 0 & s_1^{(k)} & 0 & s_2^{(k)} & s_3^{(k)} & s_4^{(k)} \\ 0 & s_1^{(k-1)} & 0 & s_2^{(k-1)} & s_3^{(k-1)} & s_4^{(k-1)} \end{pmatrix}}_{A^{(k)}} \underbrace{\begin{pmatrix} b_{11}^{(k)} \\ b_{14}^{(k)} \\ b_{23}^{(k)} \\ b_{24}^{(k)} \\ b_{34}^{(k)} \\ b_{44}^{(k)} \end{pmatrix}}_{z^{(k)}} = \underbrace{\begin{pmatrix} y_1^{(k)} \\ y_1^{(k-1)} \\ y_2^{(k)} \\ y_2^{(k-1)} \\ y_3^{(k)} \\ y_3^{(k-1)} \\ y_4^{(k)} \\ y_4^{(k-1)} \end{pmatrix}}_{c^{(k)}}.$$

The matrix $A^{(k)}$ is rectangular with its row dimension dependent on m (the number of secant directions) while the column dimension and the number of entries in each row depend on $\mathcal{S}(H(x^{(k)}))$. An important and attractive feature of this formulation is that it naturally imposes symmetry on $B^{(k)}$.

There may be null rows present in the system (2.4); these result from linear terms in the objective and/or constraints of the optimization problem. All null rows are removed prior to solving the system. Whether the resulting LS system is over- or under-determined depends on m and the density of $\mathcal{S}(H(x^{(k)}))$. If it is over-determined then the equations will be inconsistent in general. In this case, we compute the least squares solution, that is, $z^{(k)}$ that minimizes

$$\|A^{(k)} z^{(k)} - c^{(k)}\|_2^2. \tag{2.5}$$

If the system is under-determined then there are infinitely many solutions or no solutions because the equations are inconsistent. In this case, the $z^{(k)}$ that minimizes the regularized problem

$$\|A^{(k)} z^{(k)} - c^{(k)}\|_2^2 + \sigma \|z^{(k)}\|_2^2 \quad \text{for some parameter} \quad \sigma > 0, \tag{2.6}$$

is computed. In this study, we focus on choosing m so that the system is over-determined (strictly speaking it is sufficient for the system to be well-determined).

Although most rows of $A^{(k)}$ are sparse, some can be significantly denser than the others. This occurs if the Hessian matrix has one or more rows with many entries, which can be the case in some nonlinear optimization problems where the objective and/or constraints involve all (or many) of the variables. In particular, if $B^{(k)}$ has a row with $n_1 \leq n$ entries then $A^{(k)}$ has m rows with n_1 entries (see Example 2 with $n_1 = n = 4$). If n_1 is large (compared to the number of entries in the other rows of $A^{(k)}$), we refer to the problem as a *sparse-dense* LS problem.

For simplicity of exposition, in the remainder of this paper, we omit the superscript (k). When we wish to emphasise the dependence on the secant parameter, we use the notation $A(m)$. We also denote the number of entries in the upper triangular part of $B^{(k)}$ by N and set $mn = M$.

3. Solving large-scale least squares problems. Solving large-scale linear least squares problems is well-known to be significantly harder than solving large square linear systems of equations; sparse-dense LS problems are particularly challenging. In 2017, a review by Gould and Scott [12] reported on the performance of different software packages when employed to solve an extensive set of large LS problems arising from a range of practical applications. Direct methods for solving such systems are characterized by computing a matrix factorization in such a way that the problem is transformed into one that involves solving systems of equations with factor matrices that are easy and inexpensive. Direct methods obtain the solution in a finite and fixed number of steps that is independent of A and c. Due to rounding errors the computed solution is generally not equal to the exact one, but if a direct method is well implemented, the resulting software is extremely robust and can be used as a "black box solver", with the user not needing any detailed knowledge or understanding of what is going on within the box. By contrast, an iterative method generally involves an unknown number of steps and its performance is highly problem

dependent. A major advantage of iterative methods is that they require much less memory than direct methods, for which the memory requirements generally increase rapidly with problem size. Thus for very large problems, iterative methods are needed. For these to be effective, preconditioning is required. Gould and Scott highlighted some of the weaknesses of existing preconditioners for LS problems and demonstrated the specific need for new approaches together with software designed for solving sparse-dense LS problems. This led us to look at developing new ideas for preconditioners [2] and to work on algorithms that can handle sparse-dense problems [23, 24, 25, 26, 27]. These include direct solvers and LS preconditioners and, importantly, combining direct and iterative techniques.

In this paper, the sizes of the systems we are interested in allows us to focus on sparse direct methods and, for sparse-dense problems, we use them within an iterative method. We consider using both the normal equations and the larger but sparser augmented system formulation.

3.1. Direct methods for sparse LS problems. Solving (2.5) is mathematically equivalent to solving the $N \times N$ *normal equations*

$$Cz = A^T c, \qquad C = A^T A, \tag{3.1}$$

where, if A has full column rank, the *normal matrix* C is symmetric and positive definite. Thus, standard methods for solving such systems can be employed. In particular, a Cholesky factorization $C = LL^T$, where the factor L is a lower triangular matrix, can be computed. The 2-norm condition number of the normal matrix is

$$\kappa(C) = \frac{\lambda_1(C)}{\lambda_N(C)},$$

where $\lambda_1(C)$ and $\lambda_N(C)$ are its largest and smallest eigenvalues, respectively. As the condition number of C is the square of that of A, 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 can cause similar numerical problems in finite precision arithmetic.

Observe that if P is any permutation matrix, then

$$C = A^T A = (PA)^T PA,$$

so that the normal matrix C is independent of the ordering of the rows of A. Hence for our Hessian approximations, C does not depend on the ordering of the secant conditions. However, the ordering of the rows and columns of C influences the sparsity of its factors. Many direct solvers offer an initial ordering phase that chooses an appropriate permutation to limit fill-in of the factors; otherwise, an ordering package such as METIS [17] (nested dissection ordering) or the HSL routine `HSL_MC69` (which offers minimum degree and approximate minimum degree orderings) can be employed to preorder C [16].

An alternative approach is to use the much larger but sparser $(M+N) \times (M+N)$ *augmented system*

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} r \\ z \end{pmatrix} = \begin{pmatrix} c \\ 0 \end{pmatrix}, \tag{3.2}$$

where $r = c - Az$ is the residual vector. This is a symmetric indefinite system (commonly called a saddle point system) and therefore, if there is sufficient memory available, a sparse direct solver that incorporates numerical pivoting for stability can be used. Well-known and widely-available codes that compute an LDL^T factorization in which L is unit lower triangular and D is block diagonal with blocks of size 1 and 2 include `MA57` [6] and `HSL_MA97` [14, 15], MUMPS [18] and WSMP [32]. Again, preordering of the rows of the augmented system is key to limiting the density of the L factor and hence the memory requirements and the operation counts. Numerical results for direct solvers applied to both the normal equations and augmented system approaches are given in [22]. The reported experiments indicated that neither approach is consistently the best in terms of speed and/or the size of the computed factors.

Prescaling A can also be important for the success of the solver. In general, in place of (3.1) we solve

$$C(S)\hat{z} = (AS)^T c, \qquad C(S) = (AS)^T(AS), \qquad z = S\hat{z},$$

where S is a diagonal scaling matrix. For example, S could be chosen so that the 2-norm of each column of the scaled matrix AS is equal to unity. Similarly, in place of (3.2), we solve

$$\begin{pmatrix} I & AS \\ (AS)^T & 0 \end{pmatrix} \begin{pmatrix} r \\ \hat{z} \end{pmatrix} = \begin{pmatrix} c \\ 0 \end{pmatrix}, \qquad z = S\hat{z}.$$

To simplify notation, we omit S from the following discussion (but it is used in all numerical experiments).

Methods based on the QR factorization of A are also possible. These can be more stable for ill-conditioned problems but they can also be prohibitively expensive for large-scale problems. A recent computational study of QR methods for solving sparse least squares problems is given in [27].

3.2. Influence of m on the normal matrix. Assume that $A(m)$ is sparse with full column rank. The rows of $A(m)$ can be permuted so that

$$PA(m) = \begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_m \end{pmatrix},$$

where each A_j is of order $n \times N$ and $\mathcal{S}(A_j) = \mathcal{S}(A_{j+1})$, $1 \le j < m$. In Example 1, A_1 comprises rows 1, 3 and 5 and A_2 rows 2, 4 and 6. It follows that the $N \times N$ normal matrix is

$$C(m) = A(m)^T A(m) = \sum_{j=1}^{m} A_j^T A_j = \sum_{j=1}^{m} C_j,$$

where the C_j are independent and each has the same sparsity pattern. The C_j can be computed in parallel and then summed to obtain $C(m)$. Thus increasing m has a limited effect on the work required to form $C(m)$.

Writing $C(m+1) = C(m) + A_{m+1}^T A_{m+1}$, it follows from the Courant-Fisher theorem that if the eigenvalues $\{\lambda_i(C(m))\}$ and $\{\lambda_i(C(m+1))\}$ $(1 \le i \le N)$ are in decreasing order then the extreme eigenvalues satisfy

$$\lambda_1(C(m)) \le \lambda_1(C(m+1)), \quad \lambda_N(C(m)) \le \lambda_N(C(m+1)).$$

That is, as m increases the eigenvalues of the corresponding normal matrix move away from zero. There is, however, no guarantee that the conditioning of the normal matrix improves.

3.3. Solving sparse-dense LS problems. Observe that if one or more rows of A contain a significant number of entries, then the normal matrix C is effectively dense and factorizing it is impractical for large problems. Indeed, a direct solver will fail because of insufficient memory and if an incomplete factorization of C is employed as a preconditioner for an iterative method, the error in the factorization can be so large as to prohibit its effectiveness as a preconditioner. Dense rows do not prevent the use of a general-purpose sparse indefinite direct solver to solve the augmented system (3.2), but this fails to take advantage of the block structure and the need for pivoting for numerical stability inhibits the exploitation of parallelism. Obtaining robust preconditioners for such systems has been the subject of substantial research (see, for instance, [4, 21, 31] and the references therein), but this remains a challenge.

Assume the M rows of the (permuted) LS system matrix A are split into two parts with a conformal splitting of the right-hand side vector c as follows

$$A = \begin{pmatrix} A_s \\ A_d \end{pmatrix}, \ A_s \in R^{m_s \times N}, \ A_d \in R^{m_d \times N}, \ c = \begin{pmatrix} c_s \\ c_d \end{pmatrix}, \ c_s \in R^{m_s}, \ c_d \in R^{m_d}, \tag{3.3}$$

with $M = m_s + m_d$, $m_s \geq N$ and $m_s \gg m_d$. Problem (2.5) becomes

$$\min_z \left\| \begin{pmatrix} A_s \\ A_d \end{pmatrix} z - \begin{pmatrix} c_s \\ c_d \end{pmatrix} \right\|_2^2. \tag{3.4}$$

Splitting can be used to tackle sparse-dense problems in which A contains $m_d \geq 1$ rows that have many more entries than the other rows (in our Hessian matrix approximations, m_d is a multiple of m). These "dense" rows comprise A_d. In Example 2, the last $m_d = m = 2$ rows of $A^{(k)}$ arise from the last row of $B^{(k)}$, which is dense and so these rows are dense (with n entries). Another possible motivation for splitting the rows is to accommodate appending a set of additional rows, which are not necessarily dense, to A. For example, if the number of secant conditions is increased to $m + m_1$ then A_d corresponds to the extra $m_d = m_1 n$ rows and we are then interested in approaches that avoid recomputing everything from scratch.

Using (3.3), the normal equations are given by

$$Cz = (C_s + A_d^T A_d)z = d, \qquad C_s = A_s^T A_s, \qquad d = A_s^T c_s + A_d^T c_d.$$

These can be solved using the equivalent $(n + m_d) \times (n + m_d)$ blocked linear system

$$\begin{pmatrix} C_s & A_d^T \\ A_d & -I \end{pmatrix} \begin{pmatrix} z \\ A_d z \end{pmatrix} = \begin{pmatrix} d \\ 0 \end{pmatrix}. \tag{3.5}$$

If A_s has full column rank and all its rows are sparse, then the reduced normal matrix C_s is symmetric positive definite and sparse. Let $C_s = L_s L_s^T$ be its Cholesky factorization. We then have the signed Cholesky factorization

$$\begin{pmatrix} C_s & A_d^T \\ A_d & -I \end{pmatrix} = \begin{pmatrix} L_s & 0 \\ B_d & L_d \end{pmatrix} \begin{pmatrix} I & 0 \\ 0 & -I \end{pmatrix} \begin{pmatrix} L_s^T & B_d^T \\ 0 & L_d^T \end{pmatrix}, \tag{3.6}$$

where B_d is the solution of the triangular system

$$L_s B_d^T = A_d^T$$

and L_d is a Cholesky factor of the $m_d \times m_d$ (negative) Schur complement

$$I + B_d B_d^T = L_d L_d^T.$$

Assuming m_d is small, L_d can be computed using dense linear algebra and most of the work is in computing the factorization of C_s. Thus, if the rows in A_d change (whether or not they are dense), this approach provides an inexpensive updating strategy.

In practice, A_s can contain null columns. This is illustrated by Example 2, in which A_s comprises the first 6 rows; column 6 of A_s is null. In this case, A_s is rank-deficient and C_s is positive semidefinite and a Cholesky factorization breaks down. Even if C_s has no null columns, it can be singular or highly ill conditioned. There are a number of ways to overcome this, including removing the null columns explicitly [24] or employing matrix stretching [26]. A more straightforward approach is to use regularization in which the Cholesky factorization of the shifted matrix $C_s(\alpha) = A_s^T A_s + \alpha I$ is computed. Clearly, it is always possible to find a shift $\alpha > 0$ so that breakdown is avoided. When using a shift, the computed value of the least-squares objective may differ from the optimum for the original problem. However, we can seek to recover the required solution by using the factors within a preconditioner for an iterative method.

It is straightforward to verify that

$$C_s(\alpha) + A_d^T A_d = \begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} C_s(\alpha) & A_d^T \\ A_d & -I \end{pmatrix} \begin{pmatrix} I \\ A_d \end{pmatrix}$$

and

$$(C_s(\alpha) + A_d^T A_d)^{-1} = \begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} C_s(\alpha) & A_d^T \\ A_d & -I \end{pmatrix}^{-1} \begin{pmatrix} I \\ 0 \end{pmatrix}. \tag{3.7}$$

If we factorize $C_s(\alpha)$, then combined with (3.7), the factorization (3.6) (with C_s replaced by $C_s(\alpha)$) can be used to obtain a symmetric positive definite preconditioner for use with an iterative solver such as CGLS, LSQR or LSMR (see [26]).

An alternative approach is to use the splitting (3.3) with the augmented system (3.2) to obtain

$$K \begin{pmatrix} r_s \\ r_d \\ z \end{pmatrix} = \begin{pmatrix} I & 0 & A_s \\ 0 & I & A_d \\ A_s^T & A_d^T & 0 \end{pmatrix} \begin{pmatrix} r_s \\ r_d \\ z \end{pmatrix} = \begin{pmatrix} c_s \\ c_d \\ 0 \end{pmatrix}, \tag{3.8}$$

where

$$r = \begin{pmatrix} r_s \\ r_d \end{pmatrix} = \begin{pmatrix} c_s \\ c_d \end{pmatrix} - \begin{pmatrix} A_s \\ A_d \end{pmatrix} z.$$

Eliminating r_s reduces the problem to a 2-block system of order $(N + m_d) \times (N + m_d)$ of the form

$$K_r \begin{pmatrix} z \\ r_d \end{pmatrix} = \begin{pmatrix} -A_s^T c_s \\ c_d \end{pmatrix}, \qquad K_r = \begin{pmatrix} -C_s & A_d^T \\ A_d & I \end{pmatrix}. \tag{3.9}$$

Either K or K_r can be factorized using a sparse symmetric indefinite solver. The former has the advantage of not requiring the explicit computation of the reduced normal matrix C_s while the latter is a smaller system that corresponds to choosing the first m_s pivots in the factorization of K in the natural order.

4. Numerical experiments. The problems used in our experiments all come from the CUTEst test collection[1] [11]; they are listed in Table 4.1. The table includes the minimum number m_{min} of secant equations for the corresponding least squares matrix $A^{(k)}$ in equation (2.4) to be overdetermined (excluding null rows). In practice, there may be situations, particularly during the earlier iterations of an optimization algorithm, where there are insufficient past iterations to enable m to be as large as in our experiments. In the current study, we do not consider this initialisation phase but assume throughout that we can use any $m \geq m_{min}$.

TABLE 4.1

CUTEst test problems. The problems in the top (respectively, bottom) part of the table are constrained (respectively, unconstrained). The columns report the CUTEst identifier, the dimension n of H, the number $nnz(H)$ of nonzeros in the lower triangular part of H, the number n_{null} of null rows in H, the largest number $nnz(row)$ of entries in a row of H, and the number n_d of dense rows in H. m_{min} is the minimum number of secant equations for the corresponding least squares matrix $A^{(k)}$ in equation (2.4) to be overdetermined.

identifier	n	$nnz(H)$	n_{null}	$nnz(row)$	n_d	m_{min}
BQPGAUSS	2,003	9,298	0	552	1	5
CURLY30	10,000	309,535	0	61	0	31
DRCAV1LQ	4,489	87,635	12	41	0	20
JIMACK	3,549	118,824	0	81	0	34
NCVXBQP1	50,000	199,984	0	9	0	4
SINQUAD	5,000	9,999	0	5,000	1	2
SPARSINE	5,000	79,554	0	56	0	16
SPARSQUR	10,000	159,494	0	56	0	16
WALL100	149,624	1,446,475	0	42	0	10
CAR2	5,999	50,964	0	5,999	1	9
GASOIL	10,403	8,606	6,998	1,602	3	3
LUKVLE12	9,997	22,492	0	2,502	1	3
MSQRTA	1,024	33,264	0	64	0	33
ORTHREGE	7,506	17,509	2	2,504	4	3
TWIRIMD1	1,247	42,197	0	660	0	34
YATP1SQ	123,200	368,550	0	352	0	3

The characteristics of the machine used to perform the experiments are given in Table 4.2. Eight processor cores are used for our reported results and timings are elapsed times in seconds. All experiments

[1] `https://github.com/ralna/CUTEst`

TABLE 4.2
Test machine characteristics

CPU	Two Intel Xeon E5-2687W octa-core processors
Memory	64 GB
Compiler	gfortran version 9.4.0 with options -O3 -fopenmp
BLAS	MKL BLAS

(with the exception of the conditioning results given in Table 4.3) are performed using the Fortran linear least squares solver `HSL_MA85` from the HSL Mathematical Software Library [16]. This package is designed for large-scale problems that may contain some dense rows. It solves the system (3.5) or (3.8)/(3.9) using the sparse direct linear equation solver `HSL_MA87` or `HSL_MA97` respectively [13, 14, 15], combined with the iterative solvers CGLS and GMRES, respectively. `HSL_MA87` uses a DAG-based algorithm to compute the Cholesky factorization of sparse symmetric positive definite matrices, while `HSL_MA97` is a multifrontal code that incorporates numerical pivoting within an LDLT factorization. Both `HSL_MA87` and `HSL_MA97` employ OpenMP for parallelism and exploit high level BLAS routines. `HSL_MA85` includes options for scaling the least squares problem and for ordering the linear systems to limit the number of entries in the factors and the operations needed to perform the factorizations. We use equilibration scaling and nested dissection ordering. In our tests, Algorithm 1 of [26] with the density parameter set to 0.05 is used to identify rows of the least squares matrix that we treat as dense.

For the purposes of verifying the results obtained using our least squares approach, we assume the Hessian matrix $H = \{h_{ij}\}$ is known and report the relative componentwise error

$$rel\_err = \max_{(i,j)\in\mathcal{S}(H)} |b_{ij} - h_{ij}|/\max(1, |h_{ij}|), \tag{4.1}$$

where $B = \{b_{ij}\}$ is the computed approximation of H. We also report the norm of the least squares residual $\|r\|_2 = \|Az - c\|_2$.

4.1. Fixed Hessian matrix, general steps. While our ultimate goal is to provide useful, evolving Hessian matrix approximations for nonlinear functions, we start by testing whether the proposed new LS methods can compute good approximations in the simple case in which the Hessian matrix is fixed. That is, $H(x^{(k)}) = H$ for all k. To do this, we consider the (unconstrained) quadratic programming problem

$$\min_x f(x) = \frac{1}{2}x^T Hx + g^T x + c, \tag{4.2}$$

involving a scalar c, vector g and symmetric matrix H (note that here $H = \nabla^2 f(x)$ for all x). This problem underlies much of unconstrained optimization, with f and g often representing function and gradient values of a Taylor approximation to a nonlinear function $f(x)$ evaluated at suitable x, and H being an approximation to its Hessian matrix. This H is the matrix we seek to approximate.

We also want to test problems with constraints; these can involve dense rows. Thus, we consider the more general problem

$$\begin{aligned} \min_x \quad & f(x) = \frac{1}{2}x^T Hx + g^T x + c \\ \text{such that} \quad & \frac{1}{2}x^T H_q x + g_q^T x + c_q \le 0, \quad q = 1, \ldots, n_c. \end{aligned} \tag{4.3}$$

Here c_q, g_q and H_q are the values, gradients and approximations to Hessian matrices of a given set of n_c nonlinear constraints. In this case, if μ_q are Lagrange multipliers, the quadratic Lagrangian function

$$\mathcal{L}(x, \mu) = \frac{1}{2}x^T Hx + g^T x + \sum_{q=1}^{n_c} \mu_q \left(\frac{1}{2}x^T H_q x + g_q^T x\right),$$

for which the Hessian matrix

$$H_L = \nabla_x^2 \mathcal{L}(x, \mu) = H + \sum_{q=1}^{n_c} \mu_q H_q, \text{ for all } x \text{ and fixed } \mu_q,$$

is fundamental to many constrained optimization algorithms. We want to approximate the matrix H_L.

Our interest is in investigating how the proposed new LS approximation methods perform in practice. To do so, we consider idealised instances of problems (4.2) and (4.3) in which the Hessian matrices H and H_L are fixed (they are independent of the iteration k). The method we use to generate our test Hessian matrices is described in Appendix A. For unconstrained (respectively, constrained) CUTEst problems, having generated a fixed Hessian matrix H (respectively, H_L), we randomly generate $s^{(l)} \in (-1, 1)$ and then compute $y^{(l)} = Hs^{(l)}$ (respectively, $y^{(l)} = H_L s^{(l)}$) for $l = 1, \ldots, m$.

4.1.1. Varying the secant parameter. Table 4.3 presents estimates of the extremal eigenvalues for a subset of the test problems. These values were computed using the Matlab function `eigs`. They illustrate how the conditioning of the normal matrix improves as the secant parameter m increases. Choosing the minimum value m_{min} for (2.4) to be overdetermined (excluding null rows) can result in the system being close to singular (or even singular) in machine precision. Based on our experiments, we advocate choosing m to be at least $m_{min} + 5$. However, for some problems a larger value is needed. In particular, we use $m = m_{min} + 10$ for our experiments involving BQPGAUSS, and for TWIRIMD1 we use $m = 70$. Note that it is possible to construct artificial examples for which increasing m leads to growth in the condition number, but we did not encounter this behaviour in practice.

TABLE 4.3

The conditioning of the normal matrix $C(m)$ for problems from the CUTEst collection. The columns report the dimension N and number of nonzeros $nnz(C(m))$ in $C(m)$, the secant parameter m, estimates of the largest and smallest eigenvalues of $C(m)$ and its condition number. The smallest m is the minimum for (2.4) to be overdetermined (excluding null rows).

identifier	N	$nnz(C(m))$	m	$\lambda_1(C(m))$	$\lambda_N(C(m))$	$\kappa(C(m))$
BQPGAUSS	9,298	1,093,014	5	2.10E+02	2.84E-23	7.41E+24
			10	2.36E+02	3.88E-17	6.07E+18
			15	2.42E+02	1.60E-02	1.52E+04
			20	2.53E+02	1.08E-01	2.36E+03
			25	2.78E+02	2.10E-01	1.32E+03
NCVXBQP1	199,984	2,499,562	4	1.58E+01	1.27E-19	1.24E+20
			9	2.36E+01	2.27E-02	1.04E+03
			14	2.91E+01	1.96E-01	1.48E+02
			19	3.42E+01	4.36E-01	7.85E+01
WALL100	1,446,475	58,554,691	10	4.55E+01	1.34E-15	3.38E+16
			15	5.40E+01	4.41E-04	1.22E+05
			20	6.13E+01	4.27E-02	1.43E+03
			25	6.61E+01	2.02E-01	3.27E+02
CAR2	50,964	37,339,450	9	2.12E+03	1.44E-32	1.47E+35
			14	2.16E+03	1.48E-03	1.45E+06
			19	2.17E+03	8.13E-03	2.67E+05
			24	2.21E+03	1.78E-02	1.24E+05
			29	2.27E+03	2.93E-02	7.72E+04
ORTHREGE	17,509	25,127,545	3	8.73E+02	-8.50E-20	1.03E+22
			8	9.01E+02	1.86E-04	4.86E+06
			15	9.32E+02	4.39E-03	2.12E+05
			25	1.02E+03	2.62E-02	3.89E+04
			35	1.05E+03	4.13E-02	2.55E+04
TWIRIMD1	42,197	18,403,497	34	3.63E+02	-5.54E-20	6.56E+21
			60	4.23E+02	-1.33E-17	3.18E+19
			70	4.46E+02	1.58E-02	2.83E+04
			80	4.68E+02	1.82E-01	2.57E+03
			90	4.77E+02	4.47E-01	1.07E+03

For the subset of problems in Table 4.3, Table 4.4 shows the effects of varying the secant parameter m on the performance of the LS solver. Both the normal equation and the augmented system approaches are

TABLE 4.4

The effects of varying the secant parameter m on the normal equation and augmented system formulations. The columns report the secant parameter m, nnz(L) and nflops are the number of entries in the computed matrix factor and the flops required to compute it. For the augmented system approach, ndelay is the number of delayed pivots. $\|r\|_2$ is the least squares residual and rel_err is given by (4.1).

identifier	normal equation				augmented system				
m	$nnz(L)$	$nflops$	$\|r\|_2$	rel_err	$nnz(L)$	$nflops$	$ndelay$	$\|r\|_2$	rel_err
BQPGAUSS									
5	3.55E+06	2.20E+09	1.88E-07	2.74E+01	3.32E+06	2.37E+09	2.93E+02	2.74E-09	3.78E+02
10	3.55E+06	2.20E+09	5.63E-09	2.15E+02	3.09E+06	1.93E+09	1.00E+01	2.93E-09	8.94E+01
15	3.55E+06	2.20E+09	2.65E-09	8.26E-11	3.25E+06	2.24E+09	1.00E+00	3.80E-10	2.74E-10
20	3.55E+06	2.20E+09	8.50E-10	4.26E-11	3.41E+06	2.27E+09	0.00E+00	2.35E-10	2.18E-11
25	3.55E+06	2.20E+09	1.93E-08	7.91E-10	3.40E+06	2.29E+09	0.00E+00	2.77E-10	1.49E-11
NCVXBQP1									
4	2.06E+07	7.34E+09	1.80E-02	1.69E+02	2.34E+07	1.56E+10	2.77E+04	1.78E-01	1.42E+03
9	2.06E+07	7.34E+09	6.34E-09	1.28E-12	1.97E+07	6.29E+09	9.57E+02	1.47E-08	7.98E-12
14	2.06E+07	7.34E+09	7.92E-09	3.58E-12	2.08E+07	5.97E+09	0.00E+00	1.12E-07	9.26E-12
19	2.06E+07	7.34E+09	9.16E-09	1.22E-12	2.26E+07	5.84E+09	0.00E+00	9.80E-07	3.31E-12
WALL100									
10	6.32E+08	1.09E+12	1.20E-08	1.67E-02	5.98E+08	9.44E+11	2.88E+05	2.69E-04	3.88E+00
15	6.32E+08	1.09E+12	1.43E-08	5.08E-11	6.39E+08	9.46E+11	6.57E+03	9.06E-07	2.36E-09
20	6.32E+08	1.09E+12	1.49E-08	1.77E-11	6.52E+08	9.89E+11	2.48E+02	3.85E-06	3.49E-10
25	6.32E+08	1.09E+12	1.64E-08	1.02E-11	6.59E+08	9.95E+11	0.00E+00	1.01E-06	9.01E-11
CAR2									
9	3.65E+06	2.73E+08	3.03E-10	1.82E-01	8.22E+06	1.01E+10	1.99E+04	5.80E-12	1.36E+00
14	3.65E+06	2.73E+08	2.05E-10	7.25E-11	3.51E+06	2.48E+08	1.00E+00	5.32E-14	2.23E-14
19	3.65E+06	2.73E+08	1.71E-10	3.15E-11	3.76E+06	2.83E+08	0.00E+00	4.64E-14	5.35E-15
24	3.65E+06	2.73E+08	1.75E-10	1.89E-11	4.02E+06	3.23E+08	1.00E+00	4.76E-14	1.61E-15
29	3.65E+06	2.73E+08	1.83E-10	8.71E-12	4.27E+06	3.63E+08	0.00E+00	5.21E-14	1.57E-15
ORTHREGE									
3	4.25E+04	1.43E+05	1.13E-01	6.32E+00	1.28E+07	4.24E+10	5.00E+03	1.00E-08	1.37E+05
8	4.25E+04	1.43E+05	1.80E-06	2.08E-07	4.43E+05	1.54E+07	9.00E+00	2.90E-11	6.11E-12
15	4.25E+04	1.43E+05	1.72E-06	6.14E-08	7.95E+05	4.98E+07	9.00E+00	2.84E-11	3.25E-13
25	4.25E+04	1.43E+05	1.09E-06	1.56E-08	1.30E+06	1.33E+08	9.00E+00	4.90E-11	1.33E-13
35	4.25E+04	1.43E+05	5.45E-09	2.17E-10	1.80E+06	2.57E+08	9.00E+00	3.80E-11	6.22E-13
TWIRIMD1									
34	3.11E+08	3.66E+12	7.01E-09	1.05E+00	1.58E+08	9.81E+11	5.10E+04	5.51E-09	1.04E+02
50	3.11E+08	3.66E+12	3.06E-09	3.80E-01	9.41E+07	2.36E+11	8.94E+03	1.73E-09	1.81E+02
60	3.11E+08	3.66E+12	1.41E-08	3.82E-02	1.01E+08	2.43E+11	1.49E+03	7.02E-11	1.58E+01
70	3.11E+08	3.66E+12	2.79E-12	3.71E-13	1.01E+08	2.52E+11	3.10E+01	1.81E-12	1.61E-13
80	3.11E+08	3.66E+12	2.12E-12	8.27E-14	1.13E+08	3.11E+11	3.10E+01	1.55E-12	4.69E-14
90	3.11E+08	3.66E+12	1.99E-12	5.07E-14	1.26E+08	3.75E+11	2.90E+01	1.43E-12	1.92E-14

reported on (with the modifications of Section 3.3 used for the sparse-dense problems BQPGAUSS, CAR2 and ORTHREGE). For the normal equation formulation, the work involved in computing the Cholesky factors is independent of m but the computed solution, residual and rel_err depend on m. The size of the augmented system increases with m, but this may not mean an increase in the number and entries in the factor or the operation count. This can occur if for smaller m the problem is ill-conditioned because then the indefinite factorization involves more work to retain numerical stability. The number of delayed pivots (reported as $ndelay$) is an indication of this (for larger m, $ndelay$ is zero, or close to zero). We note that the quality of the results measured using the residual and relative error is similar for both the normal equations and augmented system approaches.

4.1.2. The importance of exploiting dense rows. For problems with one or more dense rows, Tables 4.5 and 4.6 illustrate the importance of exploiting these rows when solving the least squares problem (2.5). $m_d = 0$ means that all the rows (including those that are dense) are treated by the solver `HSL_MA85`

TABLE 4.5

Results for the normal system formulation with and without exploiting the dense rows in the least squares matrix A. m is the number of secant equations and m_d the number of rows in A classified as dense. $nnz(L)$ and $nflops$ are the number of entries in the normal matrix Cholesky factor and the flops required to compute it. $\|r\|_2$ is the least squares residual and rel_err is given by (4.1). The elapsed times (in seconds) for the factor and solve phases of the least squares solver `HSL_MA85` *are given by T(factor) and T(solve).*

identifier	m	m_d	$nnz(L)$	$nflops$	$\|r\|_2$	rel_err	$T(factor)$	$T(solve)$
BQPGAUSS	15	15	3.55E+06	2.20E+09	3.39E-09	6.38E-11	0.004	0.033
		0	4.39E+06	2.95E+09	3.99E-10	1.70E-10	0.366	0.009
SINQUAD	7	7	1.50E+04	2.50E+04	2.45E-09	4.25E-11	0.001	0.058
		0	1.26E+07	4.21E+10	2.40E-10	4.22E-11	3.279	0.035
CAR2	14	14	3.65E+06	2.73E+08	2.05E-10	7.25E-11	0.016	0.017
		0	2.61E+07	9.19E+10	5.85E-14	3.36E-14	6.115	0.034
GASOIL	8	24	1.92E+04	8.00E+04	3.96E-09	1.40E-09	0.001	0.038
		0	7.59E+06	1.09E+10	9.60E-13	4.63E-12	0.983	0.014
LUKVLE12	8	8	5.80E+05	2.01E+07	8.73E-08	1.80E-09	0.007	0.106
		0	1.09E+07	2.17E+10	2.53E-12	1.33E-13	1.368	0.033
ORTHREGE	8	32	4.25E+04	1.42E+05	4.06E-09	3.33E-10	0.002	0.120
		0	4.26E+07	1.73E+11	1.90E-10	3.54E-09	4.539	0.048

TABLE 4.6

Results for the augmented system formulation with and without exploiting the dense rows in the least squares matrix A. m is the number of secant equations and m_d the number of rows in A classified as dense. $nnz(L)$ and $nflops$ are the number of entries in the augmented system factor and the flops required to compute it. $\|r\|_2$ is the least squares residual and rel_err is given by (4.1). The elapsed times (in seconds) for the factor and solve phases of the least squares solver `HSL_MA85` *are given by T(factor) and T(solve).*

identifier	m	m_d	$nnz(L)$	$nflops$	$\|r\|_2$	rel_err	$T(factor)$	$T(solve)$
BQPGAUSS	15	15	3.28E+06	2.24E+09	3.80E-10	2.74E-10	0.327	0.009
		0	2.29E+06	9.94E+08	1.19E-10	9.14E-12	0.329	0.010
SINQUAD	7	7	8.54E+04	7.25E+05	2.11E-10	4.23E-11	0.045	0.002
		0	5.40E+05	6.78E+06	2.11E-10	4.21E-11	0.185	0.005
CAR2	14	14	3.51E+06	2.48E+08	5.32E-14	2.23E-14	0.476	0.022
		0	5.30E+06	2.86E+08	4.84E-14	2.06E-14	1.270	0.037
GASOIL	8	24	1.21E+05	1.89E+06	1.53E-13	2.22E-14	0.028	0.002
		0	2.53E+05	2.36E+06	1.51E-13	6.89E-14	0.124	0.004
LUKVLE12	8	8	3.22E+05	5.27E+06	2.40E-12	1.12E-13	0.051	0.015
		0	9.03E+05	1.02E+07	2.09E-12	8.23E-14	0.474	0.016
ORTHREGE	8	32	4.43E+05	1.54E+07	2.90E-11	6.11E-12	0.089	0.006
		0	7.21E+05	1.46E+07	1.54E-11	7.56E-13	0.372	0.006

as sparse. As expected, this leads to much denser factors that are more expensive to compute. For the normal equation formulation the increases are particularly large. For example, for problem ORTHREGE, if dense rows are exploited the normal matrix formulation requires 1.42E+05 flops and the solution time is 0.122 seconds but if all the rows are treated as sparse, the flops needed are 1.73E+11 and the time increases to 4.587 seconds.

When dense rows are exploited, the normal equations can be significantly faster than using the augmented system (for example, for problems BQPGAUSS and CAR2). This is because the Cholesky factorization is faster than an LDL^T factorization that has the overhead of pivoting for numerical stability.

In the remainder of the paper, all experiments on problems containing dense rows exploit those rows.

4.1.3. Results for problems with no dense rows. Table 4.7 reports results for the problems that have no dense rows. Again, both the normal equation and augmented system formulations are successful and generally of comparable quality.

TABLE 4.7

Results for the normal equation and augmented system formulations for problems with no dense rows. The columns report the secant parameter m, $nnz(L)$ and $nflops$ are the number of entries in the computed matrix factor and the flops required to compute it. $\|r\|_2$ is the least squares residual and rel_err is given by (4.1).

		normal equation				augmented system			
identifier	m	$nnz(L)$	$nflops$	$\|r\|_2$	rel_err	$nnz(L)$	$nflops$	$\|r\|_2$	rel_err
CURLY30	36	3.12E+08	3.32E+11	5.27E-08	2.87E-13	3.18E+08	3.15E+11	4.85E-06	1.06E-10
DRCAV1LQ	25	8.08E+07	1.16E+11	1.53E-08	9.75E-11	8.46E+07	1.18E+11	1.84E-07	1.46E-09
JIMACK	39	4.20E+08	2.45E+12	4.73E-09	1.26E-09	4.26E+08	2.39E+12	5.55E-09	1.50E-09
NCVXBQP1	9	2.06E+07	7.34E+09	6.35E-09	2.90E-12	1.97E+07	6.29E+09	1.47E-08	7.98E-12
SPARSINE	21	4.16E+08	4.09E+12	3.73E-09	2.06E-11	4.72E+08	5.15E+12	5.12E-08	1.79E-09
SPARSQUR	21	1.39E+09	2.59E+13	1.30E-08	1.87E-11	1.31E+09	2.14E+13	1.74E-07	5.70E-10
WALL100	15	6.32E+08	1.09E+12	1.43E-08	4.51E-11	6.39E+08	9.46E+11	9.29E-07	1.43E-08
MSQRTA	38	1.64E+08	1.28E+12	1.20E-12	8.77E-14	1.89E+08	1.66E+12	2.52E-12	3.29E-13
TWIRIMD1	70	3.11E+08	3.66E+12	2.76E-12	3.37E-13	1.01E+08	2.52E+11	1.81E-12	1.61E-13
YATP1SQ	8	1.08E+08	3.72E+10	1.12E-11	4.47E-11	1.81E+07	2.62E+08	4.51E-12	4.57E-13

4.2. Fixed Hessian matrix, nearly-dependent steps. Having confirmed that under idealized circumstances we can recover good approximations to Hessian matrices using our least squares approaches, we now consider two more realistic scenarios. In the first, we recognise that algorithms may produce steps that lie close to low-dimensional subspaces rather than uniformly in R^n. For example, it is well known that the iterates generated by the steepest-descent method tend to lie predominantly in a subspace spanned by the eigenvectors corresponding to the two largest eigenvalues of the Hessian [1, 19]. Our aim is thus to assess the ability to approximate a Hessian matrix when the step directions $s^{(l)}$ are not well distributed.

TABLE 4.8

Results for the normal equation and augmented system formulations for problems with nearly-dependent steps. The columns report the secant parameter m, $nnz(L)$ and $nflops$ are the number of entries in the computed matrix factor and the flops required to compute it. $\|r\|_2$ is the least squares residual and rel_err is given by (4.1).

		normal equation				augmented system			
identifier	m	$nnz(L)$	$nflops$	$\|r\|_2$	rel_err	$nnz(L)$	$nflops$	$\|r\|_2$	rel_err
BQPGAUSS	20	3.55E+06	2.20E+09	1.23E-08	7.69E-10	3.41E+06	2.27E+09	3.49E-10	6.05E-11
CURLY30	50	3.12E+08	3.32E+11	5.95E-08	1.11E-13	3.32E+08	3.24E+11	4.94E-06	7.13E-12
DRCAV1LQ	30	8.08E+07	1.16E+11	1.78E-08	1.10E-10	8.55E+07	1.19E+11	1.31E-07	3.02E-09
JIMACK	49	4.20E+08	2.45E+12	6.66E-09	2.12E-09	4.23E+08	2.36E+12	1.07E-08	4.32E-09
NCVXBQP1	9	2.06E+07	7.34E+09	6.46E-09	3.71E-12	1.97E+07	6.29E+09	3.95E-07	2.80E-11
SINQUAD	7	1.50E+04	2.50E+04	1.52E-09	4.88E-11	8.50E+04	7.25E+05	1.96E-10	4.87E-11
SPARSINE	30	4.16E+08	4.09E+12	4.19E-09	2.36E-11	4.40E+08	4.43E+12	1.34E-08	5.46E-10
SPARSQUR	25	1.39E+09	2.59E+13	1.38E-08	2.83E-11	1.33E+09	2.17E+13	2.62E-07	1.74E-09
WALL100	20	6.32E+08	1.09E+12	1.50E-08	4.38E-11	6.52E+08	9.89E+11	1.59E-06	9.99E-10
CAR2	19	3.65E+06	2.73E+08	2.22E-10	6.95E-11	3.76E+06	2.83E+08	5.84E-14	1.23E-14
GASOIL	8	1.92E+04	8.00E+04	1.51E-08	4.09E-09	1.22E+05	1.90E+06	1.80E-13	1.87E-14
LUKVLE12	8	5.80E+05	2.01E+07	4.33E-08	1.20E-09	3.22E+05	5.27E+06	2.92E-12	2.99E-13
MSQRTA	50	1.64E+08	1.28E+12	1.30E-12	6.48E-14	1.77E+08	1.39E+12	2.55E-12	1.26E-13
ORTHREGE	8	4.25E+04	1.43E+05	3.56E-09	1.66E-10	4.43E+05	1.54E+07	1.51E-11	8.39E-11
TWIRIMD1	80	3.11E+08	3.66E+12	5.29E-12	3.01E-12	1.13E+08	3.11E+11	2.99E-12	1.48E-12
YATP1SQ	8	1.08E+08	3.72E+10	2.68E-11	6.76E-10	1.81E+07	2.62E+08	6.30E-12	1.61E-11

To this end, we repeat our experiments for the fixed H and H_L except we now generate the $s^{(l)}$, $l = 1, \ldots, m$ as follows. For a chosen $d < m$, we compute $s^{(l)} \in (-1, 1)$ randomly as before for $l = 1, \ldots, d$. Then for some small $0 < \epsilon \ll 1$ and $l = d+1, \ldots, m$, we set $s^{(l)} = s^{(l-d)} + \epsilon\rho$, where $\rho \in (-1, 1)$ is a pseudo random number. This is intended to simulate optimization steps $s^{(l)}$ that lie in subspaces of effective (but not exact) dimension d. In our experiments, $\epsilon = 10^{-5}$ and $d = 0.8m$. The results are given in Table 4.8. As the conditioning gets worse with nearly-dependent steps, for some of the problems we found that to

obtain a *rel_err* of $O(10^{-9})$ or less a larger secant parameter was required; the values used are reported in column 2 of the table. For example, for problem BQPGAUSS, we used $m = 20$, compared to the previous value of 15. With appropriate m, we again see that both the normal equation and augmented system formulations are successful in obtaining high quality approximate Hessian matrices.

4.3. Varying Hessian matrix. In practice it is unlikely that the Hessian matrix is fixed, and thus exact reproduction from gradient differences is unlikely. In particular, from Taylor's theorem

$$H(x)s = y + e, \quad \text{where} \quad y = g(x+s) - g(x) \quad \text{and} \quad \|e\| = O(\|s\|^2), \tag{4.4}$$

for objective functions with gradients $g(x)$ and locally Lipschitz Hessian matrices $H(x)$. If there are m steps $s^{(l)}$, then

$$H(x)S = Y + E,$$

where $S = (s^{(1)}, \ldots, s^{(m)})$, $Y = (y^{(1)}, \ldots, y^{(m)})$, $y^{(l)} = g(x + s^{(l)}) - g(x)$ and $\|E\| = O(\|S\|^2)$. Thus if $B = YS^{-1}$ then

$$\|H(x) - B\| \leq \|ES^{-1}\|,$$

and B is a good approximation to $H(x)$ provided $\|S\|$ and $\|S^{-1}\|$ are modest.

We simulate this for $l = 1, \ldots, m$ by generating $s^{(l)}$ as in Section 4.2 and then generating a perturbed $y^{(l)} = Hs^{(l)} + \epsilon\rho$ (or $y^{(l)} = H_L s^{(l)} + \epsilon\rho$) for small $0 < \epsilon \ll 1$ and pseudo random $\rho \in (-1, 1)$. We no longer expect to reproduce H exactly (as the LS problem no longer has a zero residual), but our hope is to observe errors in $\|H - B\|$ of order approximately ϵ. In our experiments we set $\epsilon = 10^{-5}$. Because only $y^{(l)}$ is perturbed, the least-squares matrix and the factorizations of the normal matrix and the augmented system matrix are unchanged. Thus, in Table 4.9 we only report the least-squares residual $\|r\|_2$ and the relative error *rel_err* given by (4.1). We see that *rel_err* is now $O(10^{-5})$ or less and, with the default convergence tolerances for the solvers, both approaches report the same residuals and relative errors.

TABLE 4.9

Results for the normal equation and augmented system formulations for problems that simulate varying the Hessian matrix. The columns report the secant parameter m, the least squares residual $\|r\|_2$ and the relative error rel_err given by (4.1).

		normal equation		augmented system	
identifier	m	$\|r\|_2$	*rel_err*	$\|r\|_2$	*rel_err*
BQPGAUSS	15	8.29E-04	2.38E-05	8.29E-04	2.38E-05
CURLY30	36	1.30E-03	2.14E-08	1.30E-03	2.14E-08
DRCAV1LQ	25	8.97E-04	1.18E-05	8.97E-04	1.18E-05
JIMACK S	44	1.12E-03	1.61E-05	1.12E-03	1.61E-05
SINQUAD	7	9.10E-04	2.27E-05	9.10E-04	2.27E-05
SPARSINE	30	1.53E-03	4.14E-06	1.53E-03	4.14E-06
SPARSQUR	21	1.30E-03	7.99E-06	1.30E-03	7.99E-06
WALL100	15	5.16E-03	5.53E-05	5.16E-03	5.53E-05
CAR2	14	1.04E-03	7.06E-05	1.04E-03	7.06E-05
GASOIL	8	7.86E-04	1.27E-04	7.86E-04	1.27E-04
LUKVLE12	3	4.96E-04	8.55E-04	4.96E-04	8.55E-04
MSQRTA	38	4.33E-04	2.01E-05	4.33E-04	2.01E-05
NCVXBQP1	9	2.89E-03	2.84E-06	2.89E-03	2.84E-06
ORTHREGE	8	1.18E-03	1.29E-04	1.18E-03	1.29E-04
TWIRIMD1	70	1.22E-03	2.52E-05	1.22E-03	2.52E-05
YATP1SQ	8	4.54E-03	2.55E-04	4.54E-03	2.55E-04

5. Concluding remarks and future directions. In this paper, we have considered the problem of approximating sparse Hessian matrices. We have proposed a novel approach that uses the secant conditions

and then solves a large sparse linear LS problem. Solving this is challenging because the LS system matrix can contain dense rows (for example, when the underlying optimization problem involves constraints that involve many variables) and it can be poorly conditioned. In our experiments, we found that increasing the number m of secant equations improves the conditioning of the LS problem. For many of our tests, a sufficient value of m was generally not much larger than the minimum value m_{min} that ensures the LS problem is overdetermined (typically $m_{min} + 5$) but when we generated test problems in which we made the conditioning worse, larger m were needed to retain approximately the same level of accuracy in the computed Hessian matrix.

Existing methods for solving sparse-dense LS problems can be used and if these employ sparse direct solvers, then our numerical tests found them to be robust. The main weakness of the new approach is the size of the LS system matrix, which has a row dimension of mn and a column dimension equal to the number of entries in the Hessian matrix. Thus, although we were able to solve all our CUTEst test examples with a direct solver, the LS approach can be expensive in terms of time and memory requirements for large problems and for even larger LS problems, a preconditioned iterative solver will be needed. There is currently a lack of efficient, robust preconditioners for sparse-dense LS, although recent work of Al Daas, Jolivet and Scott [2] is promising, This lack of preconditioners hinders the development of "black box" software for computing Hessian matrices using our new approach. Consequently, our future plan is to design and implement alternative strategies that again use the secant equations but seek to employ more efficient methods of solution.

REFERENCES

[1] H. Akaike. On a successive transformation of probability distribution and its application to the analysis of the optimum gradient method. *Annals of the Institute of Statistical Mathematics*, 11(1):1–16, 1959.

[2] H. Al Daas, P. Jolivet, and J. A. Scott. A robust algebraic domain decomposition preconditioner for sparse normal equations. *SIAM J. Sci. Comput.*, 44(3), 2022. doi:10.1137/21M1434891.

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

[4] M. Benzi, G.H. Golub, and J. Liesen. Numerical solution of saddle point problems. *Acta Numerica*, 14:1–137, 2005.

[5] Å. Björck. A general updating algorithm for constrained linear least squares problems. *SIAM J. Sci. Stat. Comput.*, 5(2):394–402, 1984.

[6] I. S. Duff. MA57—a code for the solution of sparse symmetric definite and indefinite systems. *ACM Trans. Math. Soft.*, 30:118–154, 2004.

[7] R. Fletcher. An optimal positive definite update for sparse Hessian matrices. *SIAM J. Optimization*, 5(1):192–217, 1995.

[8] R. Fletcher, A. Grothey, and S. Leyffer. Computing sparse Hessian and Jacobian approximations with optimal hereditary properties. In A.R. Conn L.T. Biegler, T.F. Coleman and F.N. Santosa, editors, *Large-Scale Optimization with Applications, Part II: Optimal Design and Control*, volume 93 of *IMA Volumes in Mathematics and its Applications*, pages 37–52, Berlin, 1997. Springer.

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

[10] N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD—a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Trans. Math. Soft.*, 29(4):353–372, 2003.

[11] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEst : a constrained and unconstrained testing environment with safe threads for mathematical optimization. *Optim., Meth. Soft.*, 2014.

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

[13] J. D. Hogg, J. K. Reid, and J. A. Scott. Design of a multicore sparse Cholesky factorization using DAGs. *SIAM J. Sci. Comput.*, 32(6):3627–3649, 2010.

[14] J. D. Hogg and J. A. Scott. HSL_MA97: a bit-compatible multifrontal code for sparse symmetric systems. Technical Report RAL-TR-2011-024, STFC-Rutherford Appleton Lab., 2011.

[15] J. D. Hogg and J. A. Scott. New parallel sparse direct solvers for multicore archiectures. *Algorithms*, 6:702–725, 2013.

[16] HSL. A collection of Fortran codes for large-scale scientific computation, accessed 2023. `http://www.hsl.rl.ac.uk`.

[17] METIS. A family of multilevel partitioning algorithms, accessed 2022. `https://github.com/KarypisLab`.

[18] MUMPS. A parallel sparse direct solver. Version 5.5.0, accessed 2022. `http://mumps.enseeiht.fr/`.

[19] J. Nocedal, A. Sartenaer, and C. Zhu. On the behavior of the gradient norm in the steepest descent method. *Computational Optimization and Applications*, 22:5–35, 2002.

[20] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2006.

[21] M. Rozložník. *Saddle-Point Problems and Their Iterative Solution*. Nečas Center Series. Birkhäuser/Springer, Cham, 2018.

[22] J. A. Scott. On using Cholesky-based factorizations and regularization for solving rank-deficient linear least-squares problems. *SIAM J. Sci. Comput.*, 39:C319–339, 2017.

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

[24] J. A. Scott and M. Tůma. A Schur complement approach to preconditioning sparse least-squares problems with some dense rows. *Numerical Algorithms*, 79(4):1147–1168, 2018.

[25] J. A. Scott and M. Tůma. Sparse stretching for solving sparse-dense linear least-squares problems. *SIAM J. Sci. Comput.*, 41:A1604–A1625, 2019.

[26] J. A. Scott and M. Tůma. Strengths and limitations of stretching for least-squares problems with some dense rows. *ACM Trans. Math. Soft.*, 47(1):1:1–25, 2021.

[27] J. A. Scott and M. Tůma. A computational study of using black-box QR solvers for large-scale sparse-dense linear least squares problems. *ACM Trans. Math. Soft.*, 48(1):5:1–24, 2022.

[28] D. C. Sorensen. An example concerning quasi-Newton estimates of a sparse Hessian. *SIGNUM Newsletter*, 16(2):8–10, 1981.

[29] Ph. L. Toint. On sparse and symmetric matrix updating subject to a linear equation. *Math. Comp.*, 31(140):954–961, 1977.

[30] Ph. L. Toint. Some numerical result using a sparse matrix updating formula in unconstrained optimization. *Math. Comp.*, 32(1403):839–851, 1978.

[31] A. J. Wathen. Preconditioning. *Acta Numer.*, 24:329–376, 2015.

[32] WSMP. Watson Sparse Matrix Package (Version 20.12), 2020. `http://researcher.watson.ibm.com/researcher/view_group.php?id=1426`.

Appendix A. Generation of Hessian matrices using CUTEst. Here we describe how the unconstrained and constrained fixed Hessians H and H_L used in the numerical experiments in Section 4 are generated.

For each unconstrained CUTEst test example we evaluate its Hessian matrix $H^{cutest}(x)$ at a point x^{pert} that is a random perturbation of the CUTEst starting point x^{start} and set $H = H^{cutest}(x^{pert})$. Specifically, if x_i^{start} $(1 \leq i \leq n)$ is the initial value for component i of x^{start}, with lower and upper bounds x_i^l and x_i^u, then

$$x_i^{pert} = \begin{cases} x_i^l & \text{if} \quad x_i^l = x_i^u, \\ x_i^l + \rho \min(x_i^u - x_i^l, 1) & \text{if} \quad x_i^{start} \leq x_i^l, \\ x_i^u - \rho \min(x_i^u - x_i^l, 1) & \text{if} \quad x_i^{start} \geq x_i^u, \\ x_i^{start} + \rho \min(x_i^u - x_i^{start}, 1) & \text{otherwise.} \end{cases}$$

Here $\rho \in (0, 1)$ is the pseudo random number returned by the call `rand(seed,.true.,rho)`, where `rand` is from the optimization package GALAHAD [10] and the default `seed` is used.

For the constrained examples, we evaluate the Hessian of the Lagrangian matrix $H_L^{cutest}(x, \mu)$ at a random perturbation x^{pert} of x^{start} (as above) and randomly generated Lagrange multipliers $\mu_q^{rand} \in (-1, 1)$ $(1 \leq q \leq n_c)$, with component i of μ^{rand} returned by `rand(seed,.false.,mu(i))`. We then set $H_L = H_L^{cutest}(x^{pert}, \mu^{rand})$.