University of Reading

# *Approximating large-scale Hessian matrices using secant equations*

Article

Published Version

Creative Commons: Attribution 4.0 (CC-BY)

Open Access

Fowkes, J. M., Gould, N. I. M. and Scott, J. A. ORCID: https://orcid.org/0000-0003-2130-1091 (2025) Approximating large-scale Hessian matrices using secant equations. ACM Transactions on Mathematical Software (TOMS), 51 (2). 9. ISSN 1557-7295 doi: 10.1145/3728460 Available at https://centaur.reading.ac.uk/122164/

It is advisable to refer to the publisher's version if you intend to cite from the work. See Guidance on citing.

To link to this article DOI: http://dx.doi.org/10.1145/3728460

Publisher: Association for Computing Machinery (ACM)

All outputs in CentAUR are protected by Intellectual Property Rights law, including copyright law. Copyright and IPR is retained by the creators or other copyright holders. Terms and conditions for use of this material are defined in the End User Agreement.

www.reading.ac.uk/centaur

**CentAUR**

University of Reading

Central Archive at the University of Reading

Reading's research outputs online

# Approximating Large-Scale Hessian Matrices Using Secant Equations

JAROSLAV M. FOWKES and NICHOLAS I. M. GOULD, Scientific Computing Department, STFC Rutherford Appleton Laboratory, Didcot, United Kingdom
JENNIFER A. SCOTT, Scientific Computing Department, STFC Rutherford Appleton Laboratory, Didcot, United Kingdom and Department of Mathematics and Statistics, University of Reading, Reading, United Kingdom

Large-scale optimization algorithms frequently require sparse Hessian matrices that are not readily available. Existing methods for approximating large sparse Hessian matrices either do not impose sparsity or are computationally prohibitive. To try and overcome these limitations, we propose a novel approach that seeks to satisfy as many componentwise secant equations as necessary to define each row of the Hessian matrix. A naive application of this approach is too expensive for Hessian matrices that have some relatively dense rows but, by carefully taking into account the symmetry and connectivity of the Hessian matrix, we are able devise an approximation algorithm that is fast and efficient with scope for parallelism. Example sparse Hessian matrices from the CUTEst test collection for optimization illustrate the effectiveness and robustness of our proposed method.

CCS Concepts: • **Mathematics of computing** → **Continuous optimization**; **Numerical differentiation**; *Solvers*; *Continuous functions*;

Additional Key Words and Phrases: sparse nonlinear systems, sparse Hessian matrices, secant equations

**ACM Reference format:**
Jaroslav M. Fowkes, Nicholas I. M. Gould, and Jennifer A. Scott. 2025. Approximating Large-Scale Hessian Matrices Using Secant Equations. *ACM Trans. Math. Softw.* 51, 2, Article 9 (June 2025), 21 pages.
https://doi.org/10.1145/3728460

## 1 Introduction

Let us suppose we are given a smooth objective function $f(x)$ of $n$ variables $x$, whose gradient $g(x) := \nabla_x f(x)$ is known. Our aim is to compute estimates $B^{(k)}$ of the Hessian matrix $H(x) := \nabla_{xx} f(x)$ at a sequence of given iterates $x^{(k)}$. Such a requirement lies at the heart of both Newton-like methods for minimizing $f$ and methods that try to assess the stability of its gradient. We are particularly interested in the case where $H(x) = \{h_{ij}(x)\}$ is large and sparse with a known sparsity pattern

All authors were supported by EPSRC grant number EP/X032485/1.
Authors' Contact Information: Jaroslav M. Fowkes, Scientific Computing Department, STFC Rutherford Appleton Laboratory, Didcot, United Kingdom; e-mail: jaroslav.fowkes@stfc.ac.uk; Nicholas I. M. Gould, Scientific Computing Department, STFC Rutherford Appleton Laboratory, Didcot, United Kingdom; e-mail: nick.gould@stfc.ac.uk; Jennifer A. Scott (corresponding author), Scientific Computing Department, STFC Rutherford Appleton Laboratory, Didcot, United Kingdom and Department of Mathematics and Statistics, University of Reading, Reading, United Kingdom; e-mail: jennifer.scott@reading.ac.uk.

This work is licensed under Creative Commons Attribution International 4.0.

© 2025 Copyright held by the owner/author(s).
ACM 1557-7295/2025/6-ART9
https://doi.org/10.1145/3728460

$\mathcal{S}(H(x)) := \{(i,j) : h_{ij}(x) \neq 0\}$. We say that the Hessian matrix has an *entry* in row $i$ and column $j$ if $h_{ij}(x) \neq 0$ for some $x$. The assumption that the Hessian sparsity pattern is available is realistic when the underlying optimization problem is well understood, which is often the case for optimization problems arising from modelling problems, such as from the modelling of physical systems.

If we are extremely fortunate, an analytic expression for the Hessian matrix may be available. Alternatively, it may be possible to use automatic differentiation [16]. If not, we could consider finite-difference approximations to compute one column of $B^{(k)}$ at a time, for instance using forward differences:

$$B^{(k)}e_i \approx \frac{g(x^{(k)} + \Delta e_i) - g(x^{(k)})}{\Delta},$$

where $e_i$ is the $i$th unit vector and $\Delta$ is an appropriate small scalar [10]; more expensive but more accurate central differences can also be used [6, Section 5.6]. Note that if the Hessian matrix is dense then $n+1$ gradient evaluations are needed to find $B^{(k)}$. If, however, it is sparse, it may be possible to partition $\mathcal{N} = \{1, \ldots, n\}$ into a small number $q \ll n$ of disjoint subsets $\mathcal{N} = \bigcup_{j=1}^{q} \mathcal{I}_j$ with $\mathcal{I}_i \cap \mathcal{I}_j = \emptyset$ $(1 \leq i < j \leq q)$, such that the rows indexed in each $\mathcal{I}_j$ are orthogonal. In this case:

$$\sum_{i \in \mathcal{I}_j} B^{(k)}e_i \approx \frac{g(x^{(k)} + \Delta \sum_{i \in \mathcal{I}_j} e_i) - g(x^{(k)})}{\Delta},$$

and only $q+1$ gradients are required. Exploiting symmetry reduces the count further [2, 21].

Another possible approach is to require that $B^{(k)}$ satisfies the secant equation:

$$B^{(k)}(x^{(k)} - x^{(k-1)}) = g^{(k)} - g^{(k-1)},$$

where $g^{(k)} := g(x^{(k)})$. Traditionally, $B^{(k)}$ is obtained from the previous estimate $B^{(k-1)}$ by imposing the secant equation and requiring $B^{(k)} - B^{(k-1)}$ is of low rank [6, 20]—usually rank 1 or 2—rather than sparse. Thus, it is highly unlikely that $B^{(k)}$ will be sparse, even if its predecessor was, and the computational gains from utilising sparse linear algebra are lost. Although there have been attempts to derive sparse updates [7, 24, 25], their stability is a problem [23].

Secant methods start from an initial estimate $B^{(0)}$ (often $B^{(0)} = I$) and build up the approximation as new points are added. Thus, after $k$ steps, a rank $k$ (for the SR1 method) or $2k$ (for methods like BFGS or DFP) update will have been applied to $B^{(0)}$. A related limited-memory approach is to use dense low-rank updates, but instead of applying them all to $B^{(0)}$, apply the last $m$ updates as if they had been applied to a re-initialized $B^{(k-m)}$. In practice, the sequence of $m$ previous differences $s^{(l)} := x^{(l)} - x^{(l-1)}$ and $y^{(l)} := g^{(l)} - g^{(l-1)}$ $(l = k-1, \ldots, k-m)$ are recorded and then the effect (product with, solve with) of using the relevant $B^{(k)}$ is computed as it is required. Efficient algorithms exist for this [17, 19] but, as before, without attempting to impose the sparsity structure of $H^{(k)} := H(x^k)$ on $B^{(k)}$.

If the function $f(x)$ is partially separable, that is:

$$f(x) = \sum_{i=1}^{p} f_i(x),$$

where each 'element' $f_i(x)$ has a large invariant subspace [15], then an approximation:

$$B^{(k)} = \sum_{i=1}^{p} B_i^{(k)},$$

can be computed in which each element Hessian estimate $B_i^{(k)}$ satisfies its own secant equation:

$$B_i^{(k)}(x^{(k)} - x^{(k-1)}) = g_i(x^{(k)}) - g_i(x^{(k-1)}),$$

where $g_i(x) := \nabla_x f_i(x)$. The invariant subspace assumption implies $g_i$ and $B_i^{(k)}$ are structured. In particular, any differentiable $f(x)$ for which the Hessian matrix is sparse is partially separable [14], and in this case, each element secant equation only involves a few variables, leading to an excellent sparse approximation. This and its generalization to group-partial separability [3] forms the basis of the approximations used in LANCELOT [4]. In particular, group-partial separability means that in LANCELOT each element secant equation only requires the solution of a small dense linear system. However, this approximation has not been widely adopted. Unfortunately, it appears users either find it too difficult to provide the necessary separability structure or they are unwilling to do so.

From the user's perspective, a more appealing approach, and the one we advocate in this article, is to use the past $m$ accumulated data pairs $\{s^{(l)}, y^{(l)}\}_{l=k-m+1}^{k}$ and the sparsity structure of $H^{(k)}$ to estimate $B^{(k)}$ directly. The only developments we are aware of in this direction are that of Fletcher et al. [8] and our recent sparse linear least-squares approach [9]. The original idea of Fletcher et al. was to build an approximation $B^{(k)}$ that satisfies as well as possible the multiple secant conditions:

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

together with the symmetry condition:

$$B^{(k)} = (B^{(k)})^T,$$

and the sparsity condition:

$$\mathcal{S}(B^{(k)}) = \mathcal{S}(H^{(k)}).$$

Because (1) will usually be inconsistent, a reasonable compromise is to solve instead the convex quadratic program:

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

where $||B||_F^2$ denotes the squared Frobenius norm of the matrix $B$. The solution $B^{(k)}$ may be found by solving a linear system of order $n_e$, the number of entries in the upper triangle of $H^{(k)}$. As $n_e$ can be large, estimates of the solution $B^{(k)}$ may better be found using an iterative scheme such as conjugate gradients [8], but even this can be prohibitively expensive to compute, especially if such an approximation is to be used within an optimization code.

The idea behind our recent sparse linear least-squares approach [9] is to instead stack the nonzero entries in the upper triangular part of $B^{(k)}$ row-by-row above each other in a vector $z^{(k)}$ (equivalently, the entries in the lower triangular part are stacked column-by-column). In this way, we redefine the problem as a large sparse linear system of equations given by:

$$A^{(k)}z^{(k)} = c^{(k)}, \tag{2}$$

where the matrix $A^{(k)}$ and the vector $c^{(k)}$ are known and depend on the secant conditions (1) (see [9] for details). This enables us to find $B^{(k)}$ by computing the least-squares solution to (2), that is, the $z^{(k)}$ that minimizes:

$$||A^{(k)}z^{(k)} - c^{(k)}||_2^2,$$

using existing sparse linear algebra solvers. However, as $z^{(k)}$ by construction contains all the nonzero entries in the upper (equiv. lower) triangular part of the Hessian, the resulting linear system can be huge for large Hessian matrices and thus prohibitively expensive to solve, because

of the large number of flops required, even by state-of-the-art sparse direct linear solvers. This unfortunately hinders incorporating this approach into existing optimization solvers and thereby its applicability in practice.

The objective of this article is to develop a more computationally efficient approach than either [8] or [9] for estimating a sparse Hessian matrix given $m$ past iterates and gradient values. We assume throughout that most of the rows of the Hessian we seek to estimate are very sparse but there may be a small number of rows that are relatively dense. Note that the sparse Hessian approximation does not always need to be highly accurate, as modern optimization algorithms are capable of dealing with inaccurate Hessian approximations, e.g., trust-region algorithms merely require a bounded Hessian approximation to converge. Of course, an accurate approximation will likely yield second-order (as opposed to first-order) convergence of the optimization algorithm, which is desirable. Moreover, our proposed approach can be used for both direct optimization methods and those that require sparse matrix-vector products. If the actual Hessian is definite and reproduced sufficiently accurately, then it will retain its definiteness. This will not affect the convergence of a trust-region method to a critical point, but may be an issue if confirmation of a point that satisfies second-order necessary optimality conditions is required; the same issue will occur for any finite-precision approximation.

The article is organised as follows. In Section 2, we lay the groundwork for our sparse Hessian approximation technique and introduce our proposed new algorithm. In Section 3, we report results of numerical experiments that illustrate the potential of the new algorithm for approximating large sparse Hessian matrices in practice. Finally, concluding remarks and suggestions for future work are given in Section 4.

## 2 Sparse Hessian Approximation

### 2.1 Hessian Matrices with All Rows Sparse

We start by assuming that each row of the Hessian matrix has only a small number of entries; in the next section, we allow for the case of practical importance in which some rows have a larger number of entries. As in [8, 9], our proposed approach uses accumulated data. But rather than imposing the full secant conditions (1) for the previous $m$ steps, for each row $i$ in the approximate Hessian $B^{(k)}$ we seek to satisfy as many *componentwise* equations:

$$e_i^T B^{(k)} s^{(l)} = e_i^T y^{(l)}, \quad l = k, k-1, \ldots, k - nz_i + 1, \tag{3}$$

as are necessary to define the row. Here $nz_i$ denotes the number of entries in row $i$. We present two algorithms for solving the componentwise equations.

*Row-Wise Independent Algorithm.* Equation (3) can be rewritten as:

$$\sum_{j \in \mathcal{S}_i^{(k)}} b_{ij}^{(k)} s_j^{(l)} = y_i^{(l)}, \quad l = k, k-1, \ldots, k - nz_i + 1, \tag{4}$$

where

$$\mathcal{S}_i^{(k)} := \{ j : h_{ij}^{(k)} \neq 0 \}$$

is the set of $nz_i$ column indices of the unknown entries in row $i$ of the Hessian. Naively (and neglecting any inconsistencies or redundancies), to compute row $i$, we need as many Equations (4) as there are entries in the row. Let $z_i^{(k)}$ denote the vector of entries in row $i$. Then Equations (4) that must be satisfied by $z_i^{(k)}$ can be rewritten as an $nz_i \times nz_i$ dense linear system:

$$A_i^{(k)} z_i^{(k)} = c_i^{(k)}, \tag{5}$$

**Algorithm 1:** Sparse Hessian Approximation (Row-Wise Independent)

1: **for** $i = 1, \ldots, n$ **do**
2: Compute the $nz_i$ entries in row $i$ by constructing and solving linear system (5).
3: **end for**
4: Symmetrise $B^{(k)} := (B^{(k)} + (B^{(k)})^T)/2$.

where $A_i^{(k)}$ is the matrix whose rows $l = k, k-1, \ldots, k-nz_i+1$ consist of the entries $s_j^{(l)}$ indexed by $j \in \mathcal{S}_i^{(k)}$ and $c_i^{(k)}$ is the vector with entries $y_i^{(l)}$, $l = k, k-1, \ldots, k-nz_i+1$. The following simple example illustrates this matrix formulation.

*Example 1.* Consider the $3 \times 3$ approximate Hessian matrix

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

For row $i = 2$, $\mathcal{S}_2^{(k)} = \{1, 3\}$, $nz_2 = 2$ and the linear system (5) is given by

$$\underbrace{\begin{pmatrix} s_1^{(k)} & s_3^{(k)} \\ s_1^{(k-1)} & s_3^{(k-1)} \end{pmatrix}}_{A_2^{(k)}} \underbrace{\begin{pmatrix} b_{21}^{(k)} \\ b_{23}^{(k)} \end{pmatrix}}_{z_2^{(k)}} = \underbrace{\begin{pmatrix} y_2^{(k)} \\ y_2^{(k-1)} \end{pmatrix}}_{c_2^{(k)}}.$$

Similarly, for row $i = 3$, $\mathcal{S}_3^{(k)} = \{2, 3\}$, $nz_3 = 2$ and (5) is given by

$$\underbrace{\begin{pmatrix} s_2^{(k)} & s_3^{(k)} \\ s_2^{(k-1)} & s_3^{(k-1)} \end{pmatrix}}_{A_3^{(k)}} \underbrace{\begin{pmatrix} b_{32}^{(k)} \\ b_{33}^{(k)} \end{pmatrix}}_{z_3^{(k)}} = \underbrace{\begin{pmatrix} y_3^{(k)} \\ y_3^{(k-1)} \end{pmatrix}}_{c_3^{(k)}}.$$

We may have insufficient past data pairs $\{s^{(l)}, y^{(l)}\}_{l=k-m+1}^{k}$ for the matrices $A_i^{(k)}$ to be non-singular. There are a number of approaches we can pursue to remedy this. For clarity of exposition, we defer discussing this to Section 2.4. Observe that the small dense systems (5) are square when we have sufficient data, because we do not use all $m$ available past data pairs but only as many as are necessary, i.e., $nz_i \leq m$.

Using (5), we can compute the sparse Hessian approximation $B^{(k)}$ as outlined in Algorithm 1. Both the off-diagonal entries $b_{ij}^{(k)}$ and $b_{ji}^{(k)}$ are computed, and then, in the final step, the average is taken to obtain a symmetric approximate Hessian matrix. If the upper and lower triangular parts of $B^{(k)}$ are stored, then the rows can be computed in parallel in any order. However, it is often the case that only the upper triangular (or equivalently lower triangular) part of $B^{(k)}$ is held and it may not be desirable to form both triangles, in which case the rows cannot be computed in parallel but must be computed in sequence.

*Row-Wise Dependent Algorithm.* Taking an average of the two off-diagonal entries (or accepting one of the computed values) does not truly account for symmetry, in the sense that we are not enforcing the off-diagonal entries to be symmetric but instead hoping that taking the average (or one) of the computed values produces a sufficiently accurate symmetric approximation. Instead, we can move the already determined entries of the Hessian over to the right-hand side of (4), that

is, we can rewrite (4) as

$$\sum_{j\in\mathcal{U}_i^{(k)}} b_{ij}^{(k)} s_j^{(l)} = y_i^{(l)} - \sum_{j\in\mathcal{K}_i^{(k)}} b_{ij}^{(k)} s_j^{(l)}, \quad l = k, k-1, \ldots, k-nu_i+1, \tag{6}$$

where

$$\mathcal{K}_i^{(k)} := \{j : h_{ij}^{(k)} \neq 0 \text{ and } b_{ij}^{(k)} \text{ is already known}\} \text{ and } \mathcal{U}_i^{(k)} := \{j : h_{ij}^{(k)} \neq 0\} \setminus \mathcal{K}_i^{(k)}$$

are the column indices of the known and unknown entries, respectively, in row $i$ of the Hessian, and $nu_i := |\mathcal{U}_i^{(k)}|$ denotes the number of remaining unknown entries in row $i$. The $i$th componentwise Equation (6) can then be rewritten as the $nu_i \times nu_i$ dense linear system:

$$U_i^{(k)} z_i^{(k)} = c_i^{(k)} - K_i^{(k)} w_i^{(k)}, \tag{7}$$

where $w_i^{(k)}$ holds the entries in row $i$ of $B^{(k)}$ that are already known, $z_i^{(k)}$ holds the remaining entries in the row, and $K_i^{(k)}$ and $U_i^{(k)}$ are the corresponding sub-matrices of $A_i^{(k)}$ (note that $K_i^{(k)}$ will in general not be square even if sufficient data is available). This formulation is illustrated in the following simple example.

*Example 2.* Consider the $5 \times 5$ approximate Hessian matrix:

$$B^{(k)} = \begin{pmatrix} b_{11}^{(k)} & 0 & 0 & b_{14}^{(k)} & 0 \\ 0 & b_{22}^{(k)} & 0 & b_{24}^{(k)} & 0 \\ 0 & 0 & 0 & b_{34}^{(k)} & 0 \\ b_{41}^{(k)} & b_{42}^{(k)} & b_{43}^{(k)} & b_{44}^{(k)} & b_{45}^{(k)} \\ 0 & 0 & 0 & b_{54}^{(k)} & b_{55}^{(k)} \end{pmatrix} \text{ with } b_{14}^{(k)} = b_{41}^{(k)}, b_{24}^{(k)} = b_{42}^{(k)}, b_{34}^{(k)} = b_{43}^{(k)} \text{ and } b_{45}^{(k)} = b_{54}^{(k)}.$$

Assume the rows are computed in the natural order $1, 2, 3, 4, 5$. Row 4 has five entries and by symmetry $b_{41}^{(k)}, b_{42}^{(k)}, b_{43}^{(k)}$ are already known. The linear system (7) for row 4 is therefore:

$$\underbrace{\begin{pmatrix} s_4^{(k)} & s_5^{(k)} \\ s_4^{(k-1)} & s_5^{(k-1)} \end{pmatrix}}_{U_4^{(k)}} \underbrace{\begin{pmatrix} b_{44}^{(k)} \\ b_{45}^{(k)} \end{pmatrix}}_{z_4^{(k)}} = \underbrace{\begin{pmatrix} y_4^{(k)} \\ y_4^{(k-1)} \end{pmatrix}}_{c_4^{(k)}} - \underbrace{\begin{pmatrix} s_1^{(k)} & s_2^{(k)} & s_3^{(k)} \\ s_1^{(k-1)} & s_2^{(k-1)} & s_3^{(k-1)} \end{pmatrix}}_{K_4^{(k)}} \underbrace{\begin{pmatrix} b_{14}^{(k)} \\ b_{24}^{(k)} \\ b_{34}^{(k)} \end{pmatrix}}_{w_4^{(k)}}.$$

For this approach, the order in which the rows are computed is crucial. This can be seen from the following two Hessian sparsity patterns with arrow-head structures that are symmetric permutations of each other.

$$\begin{pmatrix} * & & & & * \\ & * & & & * \\ & & . & & \vdots \\ & & & * & * \\ * & * & \cdots & * & * \end{pmatrix} \text{ and } \begin{pmatrix} * & * & \cdots & * & * \\ * & * & & & \\ \vdots & & . & & \\ * & & & * & \\ * & & & & * \end{pmatrix}.$$

Assume the rows are processed in the order $1, 2, \ldots n$. For the matrix on the left, for each of the first $n-1$ rows, two entries—one on the diagonal and one in column $n$—must be computed. The last row has $n$ entries but, by symmetry, all the off-diagonal entries have already been computed. Thus, only the diagonal entry is unknown, and hence all the entries can be computed using two data pairs $\{s^{(k)}, y^{(k)}\}$ and $\{s^{(k-1)}, y^{(k-1)}\}$. This is in contrast with the second matrix, where the first row contains $n$ entries. Computing these requires $n$ data pairs $\{s^{(l)}, y^{(l)}\}_{l=k-n+1}^{k}$. Whilst this

**Algorithm 2:** Sparse Hessian Approximation (Row-Wise Dependent)

1: Compute the adjacency graph $\mathcal{G}(B^{(k)})$ and the (current) degree of each vertex.
2: **for** $i = 1, \ldots, n$ **do**
3: Select vertex $v$ of current minimum degree; assign the corresponding row as row $i$.
4: Compute the $nu_i$ unknown entries in row $i$ by solving the linear system (7).
5: Remove $v$; decrement the current degree of each of its remaining neighbours by 1.
6: **end for**

also requires $n$ computations, the first matrix ordering is clearly much better since it only requires two past data pairs, i.e., past iterates, to be available.

Using (7), we can compute the sparse Hessian approximation $B^{(k)}$ as described in Algorithm 2, which employs the adjacency (or connectivity) graph $\mathcal{G}(B^{(k)})$. Recall that this is an undirected graph with $n$ vertices and an edge $(i, j)$ if and only if $b_{ij}^{(k)} \neq 0, i \neq j$. If there is an edge $(i, j)$ then $i$ and $j$ are said to be neighbours. The degree of vertex $i$ is the number of neighbours it has, which is equal to the number of off-diagonal entries in row $i$ of $B^{(k)}$. Algorithm 2 aims to limit the number of data pairs required at each step. It starts by selecting a vertex $v$ of minimum degree (ties are broken arbitrarily). This can be done using a counting [22, Section 2.4.6] or bucket [5] sort in $O(n) + O(n_e)$ operations and storage locations ($n_e$ is the number of edges). Once chosen, $v$ is removed from the vertex set and the degree of each neighbour of $v$ is decreased by one (corresponding to removing all edges involving $v$ from the edge set). The process is then repeated for the reduced graph (but a complete reordering of the remaining vertices should be avoided). We define the *current degree* of a vertex to be its degree in the reduced graph (initially, the current degree is equal to the degree).

While Algorithm 2 generally requires fewer floating-point operations than Algorithm 1, it has two related disadvantages. The first is that the steps in Algorithm 1 may be performed in parallel (if the upper and lower triangular parts of $B^{(k)}$ are stored) while Algorithm 2 is largely sequential—in practice, vertices $i$ and $j$ of minimum current degree with non-overlapping sparsity patterns ($\mathcal{I}_i^+ \cap \mathcal{I}_j^+ = \emptyset$) can be processed in parallel. The second more serious defect is that inaccurate estimates from earlier steps in Algorithm 2 can be magnified when solving (7), leading to error growth even for constant Hessian matrices ($H^{(k)} = H$ for all $k$). The observed error growth is usually gradual but relentless; the more times an inaccurate early value occurs in later rows, the worse the effect, and this is potentially particularly pernicious for matrices in which some of the rows have a large number of entries. This sparse-dense case is not a problem for Algorithm 1 because each row is computed independently.

### 2.2 Sparse-Dense Hessian Matrices

We now explore the case where some rows of the Hessian matrix have appreciably more entries than others. We refer to such rows as dense (even though the number of entries may be significantly less than $n$). We assume the approximate Hessian matrix $B^{(k)}$ has been symmetrically permuted to:

$$B^{(k)} = \begin{pmatrix} B_{11}^{(k)} & B_{12}^{(k)} \\ B_{21}^{(k)} & B_{22}^{(k)} \end{pmatrix}, \quad \text{with} \quad B_{21}^{(k)} = (B_{12}^{(k)})^T, \tag{8}$$

where the blocks $B_{11}^{(k)}$ and $B_{22}^{(k)}$ are square symmetric matrices, the $n_1 < n$ rows of $(B_{11}^{(k)} \; B_{12}^{(k)})$ are sparse and the remaining $n_2 = n - n_1 \ll n_1$ rows of $(B_{21}^{(k)} \; B_{22}^{(k)})$ are classified as dense. Note that we do not explicitly perform this symmetric permutation, but define it here for ease of presentation of the subsequent algorithms. In practice, we simply need to determine the sets of sparse and dense

row indices based on the row densities of $B^{(k)}$ (these are known provided the sparsity pattern of $B^{(k)}$ is known).

*Combined and Thresholding Algorithms.* An obvious approach is to compute the $n_1$ sparse rows of $B^{(k)}$ using (5) from Algorithm 1; set $B_{21}^{(k)}$ by symmetry and then use (7) from Algorithm 2 to compute the $n_2$ dense rows. This is outlined in the report [11], but on some examples it suffers from error growth (as discussed earlier for Algorithm 2).

An alternative strategy generalises Algorithm 2 by using (7) to compute the unknown entries in row $i$ only if the number of all entries $nz_i$ in row $i$ is deemed too large to use (5). This is also outlined in [11], but again this can suffer from error growth on some examples as documented in the report.

*Block Parallel Algorithm.* A better approach that prevents potential growth is the following, which is almost the same as the naive combined algorithm above. The $n_1$ sparse rows $(B_{11}^{(k)} \;\; B_{12}^{(k)})$ are computed using (5) from Algorithm 1; $B_{21}^{(k)}$ is set by symmetry and then crucially the small $n_2 \times n_2$ block $B_{22}^{(k)}$ is computed using a variant of (7). That is, consider rewriting (4) as

$$\sum_{j \in \mathcal{V}_i^{(k)}} b_{ij}^{(k)} s_j^{(l)} = y_i^{(l)} - \sum_{j \in \mathcal{L}_i^{(k)}} b_{ij}^{(k)} s_j^{(l)}, \quad l = k, k-1, \ldots, k - nv_i + 1 \tag{9}$$

where

$$\mathcal{L}_i^{(k)} := \{j : h_{ij}^{(k)} \neq 0 \text{ and } b_{ij}^{(k)} \in B_{21}^{(k)}\} \;\text{ and }\; \mathcal{V}_i^{(k)} := \{j : h_{ij}^{(k)} \neq 0\} \setminus \mathcal{L}_i^{(k)}$$

are the column indices of the known entries in $B_{21}^{(k)}$ and unknown entries, respectively, in row $i$ of the Hessian, and $nv_i := |\mathcal{V}_i^{(k)}|$ denotes the remaining number of unknown entries in row $i$. The $i$th componentwise Equation (9) can then be rewritten as the dense linear system:

$$V_i^{(k)} z_i^{(k)} = c_i^{(k)} - L_i^{(k)} w_i^{(k)}, \tag{10}$$

where $w_i^{(k)}$ holds the entries in row $i$ of $B^{(k)}$ that are already known, i.e., in $B_{21}^{(k)}$, $z_i^{(k)}$ holds the remaining entries in the row, and $L_i^{(k)}$ and $V_i^{(k)}$ are the corresponding sub-matrices of $A_i^{(k)}$. This formulation is illustrated in the following simple example (note that, as before, $L_i^{(k)}$ will in general not be square even if sufficient data is available).

*Example 3.* Let $n = 4$ and consider the approximate symmetric Hessian matrix:

$$B^{(k)} = \begin{pmatrix} B_{11}^{(k)} & B_{12}^{(k)} \\ B_{21}^{(k)} & B_{22}^{(k)} \end{pmatrix} = \left(\begin{array}{cc|cc} b_{11}^{(k)} & 0 & b_{13}^{(k)} & b_{14}^{(k)} \\ 0 & 0 & b_{23}^{(k)} & b_{24}^{(k)} \\ \hline b_{31}^{(k)} & b_{32}^{(k)} & b_{33}^{(k)} & b_{34}^{(k)} \\ b_{41}^{(k)} & b_{42}^{(k)} & b_{43}^{(k)} & b_{44}^{(k)} \end{array}\right),$$

where the first two rows are considered sparse and the last two rows dense. The $2 \times 2$ linear system (7) for the dense row 3 is

$$\underbrace{\begin{pmatrix} s_3^{(k)} & s_4^{(k)} \\ s_3^{(k-1)} & s_4^{(k-1)} \end{pmatrix}}_{V_3^{(k)}} \underbrace{\begin{pmatrix} b_{33}^{(k)} \\ b_{34}^{(k)} \end{pmatrix}}_{z_3^{(k)}} = \underbrace{\begin{pmatrix} y_3^{(k)} \\ y_3^{(k-1)} \end{pmatrix}}_{c_3^{(k)}} - \underbrace{\begin{pmatrix} s_1^{(k)} & s_2^{(k)} \\ s_1^{(k-1)} & s_2^{(k-1)} \end{pmatrix}}_{L_3^{(k)}} \underbrace{\begin{pmatrix} b_{13}^{(k)} \\ b_{23}^{(k)} \end{pmatrix}}_{w_3^{(k)}},$$

**Algorithm 3:** Sparse-Dense Hessian Approximation (Block Parallel)

1: **parallel for** $i = 1, \ldots, n_1$ **do**
2: Compute all the entries in row $i$ of $(B_{11}^{(k)} \; B_{12}^{(k)})$ by solving the linear system (5).
3: **end parallel for**
4: Set $B_{21}^{(k)} := (B_{12}^{(k)})^T$.
5: **parallel for** $i = 1, \ldots, n_2$ **do**
6: Compute all the entries in row $i$ of $B_{22}^{(k)}$ by solving the linear system (10).
7: **end parallel for**
8: Symmetrise $B^{(k)} := (B^{(k)} + (B^{(k)})^T)/2$.

and for the dense row 4 the $2 \times 2$ system (7) is

$$\underbrace{\begin{pmatrix} s_3^{(k)} & s_4^{(k)} \\ s_3^{(k-1)} & s_4^{(k-1)} \end{pmatrix}}_{V_3^{(k)}} \underbrace{\begin{pmatrix} b_{43}^{(k)} \\ b_{44}^{(k)} \end{pmatrix}}_{z_3^{(k)}} = \underbrace{\begin{pmatrix} y_4^{(k)} \\ y_4^{(k-1)} \end{pmatrix}}_{c_3^{(k)}} - \underbrace{\begin{pmatrix} s_1^{(k)} & s_2^{(k)} \\ s_1^{(k-1)} & s_2^{(k-1)} \end{pmatrix}}_{L_3^{(k)}} \underbrace{\begin{pmatrix} b_{14}^{(k)} \\ b_{24}^{(k)} \end{pmatrix}}_{w_3^{(k)}}.$$

The final value of entries (3,4) and (4,3) of $B^{(k)}$ is $(b_{34}^{(k)} + b_{43}^{(k)})/2$ (symmetrisation).

The complete algorithm is described in Algorithm 3. Here not only the sparse rows, but also the dense rows can be handled in parallel if the lower and upper triangles of $B^{(k)}$ are stored. In fact, a dense row can be computed as soon as all the entries coming from the sparse part are known (and thus it is not necessary to wait until all the sparse rows have been computed before starting the dense rows, although we have not implemented this).

### 2.3 Hessian Matrices with Variable Row Densities

Finally, we present an algorithm that can handle general sparse Hessian matrices with variable row densities. For clarity of exposition, we omit the superscript $(k)$ and assume that the approximate Hessian matrix $B$ at the $k$th iteration has been symmetrically permuted to

$$B = \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1N} \\ B_{21} & B_{22} & \cdots & B_{2N} \\ \vdots & \vdots & \vdots & \vdots \\ B_{N1} & B_{N2} & \cdots & B_{NN} \end{pmatrix}, \quad B_{12}^T = B_{21}, \text{etc.}, \tag{11}$$

for some $2 < N < n$, where the blocks $B_{ij}$ are all square symmetric matrices with the row blocks ordered according to increasing row density, i.e., the rows in $(B_{11} \; B_{12} \; \cdots \; B_{1N})$ are the sparsest, followed by those in $(B_{21} \; B_{22} \; \cdots \; B_{2N})$, etc., with finally the rows in $(B_{N1} \; B_{N2} \; \cdots \; B_{NN})$ being the densest. We do not explicitly form this symmetric permutation, but we simply determine the sets of row indices belonging to each row block.

*Recursive Block Parallel Algorithm.* Algorithm 3 can be extended to (11) by applying it recursively: after the first sparse row block is computed using (5) and the corresponding symmetric entries are populated, the dense row block becomes the next sparse row block and is computed using (10), where only the entries in the previously computed symmetric blocks are assumed known. The process is repeated until there are no more sparse rows. We illustrate this for two levels of recursion in the diagram below, where the superscript denotes recursion depth:

$$B = \begin{pmatrix} B_{11}^{[0]} & B_{12}^{[0]} \\ B_{21}^{[0]} & B_{22}^{[0]} \end{pmatrix} \implies B_{22}^{[0]} = \begin{pmatrix} B_{11}^{[1]} & B_{12}^{[1]} \\ B_{21}^{[1]} & B_{22}^{[1]} \end{pmatrix} \implies B_{22}^{[1]} = \begin{pmatrix} B_{11}^{[2]} & B_{12}^{[2]} \\ B_{21}^{[2]} & B_{22}^{[2]} \end{pmatrix}$$

**Algorithm 4:** Sparse-Dense Hessian Approximation (Recursive Block Parallel)

**Require:** $r_{\max}$ maximum recursion depth, $n_{\min}$ minimum entries per row to recurse on.

Find rows $i$ with $nz_i \leq m$ and place them in $(B_{11}^{[0]}\ B_{12}^{[0]})$.
**parallel for** $i = 1, \ldots, n_1^{[0]}$ **do**
    Compute all the entries in row $i$ of $(B_{11}^{[0]}\ B_{12}^{[0]})$ by solving the linear system (5).
**end parallel for**
Set $B_{21}^{[0]} := (B_{12}^{[0]})^T$.
Initialise recursion counter $r := 0$.
**while** $r < r_{\max}$ and there exist rows $i$ with $n_{\min} \leq nv_i \leq m$ **do**
    Increment $r := r + 1$
    Find rows $i$ with $n_{\min} \leq nv_i \leq m$ and place them in $(B_{11}^{[r]}\ B_{12}^{[r]})$.
    **parallel for** $i = 1, \ldots, n_1^{[r]}$ **do**
        Compute all entries in row $i$ of $(B_{11}^{[r]}\ B_{12}^{[r]})$ by solving the system (10) where
        only the entries in previously computed symmetric blocks are assumed known.
    **end parallel for**
    Set $B_{21}^{[r]} := (B_{12}^{[r]})^T$.
**end while**
**parallel for** $i = 1, \ldots, n_2^{[r]}$ **do**
    Compute all entries in row $i$ of the remaining $B_{22}^{[r]}$ by solving the system (10)
    where only entries in previously computed symmetric blocks are assumed known.
**end parallel for**
Symmetrise $B := (B + B^T)/2$.

where $B_{11}^{[0]} = B_{11}$, $B_{11}^{[1]} = B_{22}$ and $B_{11}^{[2]} = B_{33}$ in (11). As before, we assume at each level of recursion $r$ that $n_1^{[r]}$ rows of $(B_{11}^{[r]}\ B_{12}^{[r]})$ are classified as sparse and the remaining $n_2^{[r]}$ rows of $(B_{21}^{[r]}\ B_{22}^{[r]})$ are classified as dense. This approach of course requires a threshold for the sparse/dense split in terms of the unknown entries per row $nv_i$ on which to recurse, and a natural choice is $m$ the number of past iterates available. Since it is inefficient to solve very small dense linear systems, we also include a minimum unknown entries per row threshold $n_{\min}$. Finally, to prevent the algorithm from recursing excessively (which could hamper parallelism), we also require a maximum recursion depth $r_{\max}$. As before, all the row blocks can be handled in parallel if the upper and lower triangles are stored. The complete algorithm is described in Algorithm 4 (here we omit the iteration superscript $(k)$ for clarity). Note that if there is no recursion ($r = 0$) then Algorithm 4 is equivalent to Algorithm 3 above.

### 2.4 Implementation Details

In each of the four algorithms described above, we need to solve a dense linear system of equations for each row $i$ of the approximate Hessian at each iteration. This dense system is either of the form (5), i.e.:

$$A_i z_i = c_i,$$

or of the form (7), i.e.:

$$U_i z_i = c_i - K_i w_i,$$

or of the form (10), i.e.:

$$V_i z_i = c_i - L_i w_i.$$

Thus generically, we can write each of these dense linear system in the form:

$$Az = c, \tag{12}$$

for $A \in \mathbb{R}^{m_s \times n_s}$ where we solve for the unknowns $z \in \mathbb{R}^{n_s}$ in a particular Hessian row.

*Dealing with Insufficient Data.* Ideally, we would have sufficient past data available so that $m \geq m_s$ and the matrix $A$ in (12), made up from the $m_s$ most recent $\{s^{(l)}\}_{l=k-m_s+1}^{k}$, would be non-singular. But this may not be the case. Firstly, in the early stages of the optimization algorithm, there may not be enough data; this will certainly be the case if $m < m_s$. Secondly, $A$ formed as above may be singular (or close to singular) and in this case either there will be insufficient data to determine $z$ uniquely or, if the objective function $f$ is not quadratic, the gradient data $c$, made up from the $m_s$ most recent $\{y^{(l)}\}_{l=k-m_s+1}^{k}$, may itself be inconsistent. In such cases, one possible remedy is to assign certain components of $z$ to zero, and solve for the remainder (for example, entries far from the diagonal could be dropped). However, this is relatively arbitrary and a better strategy is to find the smallest $z$ consistent with the data by solving the constrained least-squares problem:

$$\min_{z \in \mathbb{R}^{n_s}} ||z||_2 \quad \text{subject to} \quad Az = c,$$

using, for example, the singular-value decomposition of $A$. When the latest data is inconsistent, rather than trying to find earlier data to exchange, we can add earlier data into $A$ and $c$ and then solve a weighted least-squares problem:

$$\min_{z \in \mathbb{R}^{n_s}} ||W(Az - c)||_2,$$

where the diagonal weighting matrix $W$ favours the latest data. Once again, a singular-value decomposition of $WA$ is suitable. The simplest case is $W = I$. This is used in the implementations of our proposed algorithms that we employ for our numerical experiments in Section 3.

*Solving the Linear System.* In both the under- and over-determined cases, we compute the compact singular-value decomposition $A = U\Sigma V^T \in \mathbb{R}^{m_s \times n_s}$, where the columns of $U \in \mathbb{R}^{m_s \times r_s}$ and $V \in \mathbb{R}^{n_s \times r_s}$ are orthogonal, $\Sigma \in \mathbb{R}^{r_s \times r_s}$ is non-singular and diagonal, and $r_s$ is the rank of $A$. We then find the required solution $z = V\Sigma^{-1}U^T c$ using `gelsd`, the SVD divide-and-conquer algorithm from LAPACK [1]. It is also possible to use a faster but potentially less stable variant based on a QR factorization of $A$ with interchanges using LAPACK `gelsy`, as well as a faster-still LU-based approach when $A$ is square and non-singular using LAPACK `gesv`, but we have found that the resulting dense linear systems are small enough that these offer little advantage (the largest linear system solved by Algorithm 3 or 4 in the tests is of size $95 \times 94$). For numerical stability it may be desirable to add extra data (if available) when solving the linear system and we include such an option in our implementations.

## 3 Numerical Experiments

We now examine how the proposed algorithms perform in practice using the subset of Hessian matrices from the CUTEst [13] collection that was used in [9] and is listed in Table 1. This subset contains matrices that are known to be challenging to approximate and often require a large number of past iterates $m$ to obtain a good approximation. Across all CUTEst test problems the median $m$ required is 4, which is far smaller than the values needed by the chosen matrices. Our experiments are performed on either a single processor core or 28 processor cores of a dedicated single node on the STFC SCARF cluster, comprising 32 AMD Epyc 7502 CPUs clocked at 2.5 GHz with 256 GB of RAM. The algorithms from Section 2 have been implemented in the Fortran 2018 package `SHA` as part of the GALAHAD library [12]. All codes are compiled in double precision using the GNU Fortran 13.2 compiler with O2 optimization, znver2 processor architecture, and OpenMP parallelism.

Table 1. CUTEst Test Problems

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

The problems in the top (respectively, bottom) half of the table are unconstrained (respectively, constrained). 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 $H$, and the largest number $nnz(row)$ of entries in a row of $H$.

LAPACK routines are provided by OpenBLAS 0.3.24, compiled for the cluster node as provided by SCARF (and limited by configuration to 28 OpenMP threads).

In our numerical experiments, we seek to investigate the accuracy attained by the different algorithms under ideal circumstances. We therefore test whether they compute good approximations in the simple case in which the Hessian matrix is fixed, that is, $H(x^{(k)}) = H$ for all $k$. The method used to generate the test Hessian matrices is described in Appendix A. Having generated $H$, we randomly generate $s^{(l)} \in (-1, 1)$ and then compute $y^{(l)} = Hs^{(l)}$ for $l = 1, \ldots, m$. We vary the number of past iterates $m = 1, \ldots, 100$ to explore how well the algorithms cope in both the under- and over-determined cases. For numerical stability, we use one extra past iterate if available when solving the linear system. To verify the accuracy of the computed $B = \{b_{ij}\}$, we assume $H$ is known and compute both the maximum relative componentwise error:

$$\text{max\_rel\_err} = \max_{(i,j)\in\mathcal{S}(H)} |b_{ij} - h_{ij}|/\max(1, |h_{ij}|), \tag{13}$$

as well as the median relative componentwise error:

$$\text{med\_rel\_err} = \operatorname*{med}_{(i,j)\in\mathcal{S}(H)} |b_{ij} - h_{ij}|/\max(1, |h_{ij}|). \tag{14}$$

### 3.1 Results When Using Sparse Finite-Differences

We start by approximating the Hessians using the sparse finite-difference algorithm proposed by Powell and Toint [21]. We use the implementation FDH from GALAHAD with the recommended relative finite difference step size of $10^{-7}$ (see Appendix A for details). In Table 2, we report the maximum and median relative errors, as well as the runtime. We can immediately see that the results are

Table 2. The Maximum and Median Relative Error and Runtime (in Seconds) When Applying the GALAHAD Package FDH with a Relative Finite-Difference Step Size of $10^{-7}$ to the Test Problems

| Identifier | FDH | | |
|---|---|---|---|
| | max_rel_err | med_rel_err | Runtime |
| BQPGAUSS | 8.35E-03 | 2.80E-09 | 6.00E-03 |
| CURLY30 | 3.41E-01 | 6.16E-07 | 3.10E-02 |
| DRCAV1LQ | [a] | 3.50E-08 | 3.70E-02 |
| JIMACK | 4.73E+02 | 6.03E-07 | 6.91 |
| NCVXBQP1 | 2.68E-04 | 5.30E-09 | 3.00E-02 |
| SINQUAD | 9.58E-06 | 1.47E-08 | 2.00E-03 |
| SPARSINE | 2.13E-04 | 1.10E-07 | 4.60E-02 |
| SPARSQUR | 1.70E-04 | 5.06E-08 | 7.10E-02 |
| WALL100 | 1.28E+01 | 8.99E-09 | 9.71E-01 |
| CAR2 | 2.59 | 0 | 2.50E-02 |
| GASOIL | 2.61E-07 | 0 | 4.00E-03 |
| LUKVLE12 | 9.04E-01 | 3.75E-08 | 4.00E-03 |
| MSQRTA | 1.31E-03 | 3.02E-09 | 2.49E-01 |
| ORTHREGE | 2.51E-02 | 1.61E-09 | 8.00E-03 |
| TWIRIMD1 | 5.34E-06 | 4.97E-09 | 6.80E-02 |
| YATP1SQ | 1.22E-05 | 8.62E-09 | 2.71E-01 |

[a]indicates the error exceeds $10^{15}$.

generally poor, with only three test problems achieving a maximum relative error of less than $10^{-5}$. For a number of examples (including DRCAV1LQ), the errors are very large, illustrating that sparse finite-difference approximations to the Hessian matrix are potentially unstable. Moreover, they require $q + 1$ gradient evaluations at each optimization iteration $k$ as there is no re-use of prior data. Finding the minimum $q$ is NP-hard [18], but a good heuristic (employed by FDH) is related to our Algorithms 2 for which $q = \max_i nu_i$. We will therefore not consider sparse finite-difference algorithms further.

### 3.2 Results When Treating All Rows as Sparse

Next, we illustrate the potential shortcomings of the row-wise independent and row-wise dependent algorithms. In Table 3, we report the relative errors when applying the serial Algorithms 1 and 2 to the test examples with $m = 100$ past iterates. For Algorithm 1, we see that while the median relative error is often close to machine precision, the maximum relative error can be large for problems containing dense rows. This is to be expected as Algorithm 1 treats each row $i$ independently and therefore requires as many past iterates $m$ as there are entries $nz_i$ in the row. We can see from Table 1 that a number of test problems have at least 1 row with more than 100 entries and it is precisely on these problems that Algorithm 1 struggles. The situation is much worse for Algorithm 2, which requires $\max_i nu_i$ past iterates, as there is relentless error growth for a significant number of problems. Again, this is expected, because inaccurate estimates from the early rows are magnified when substituted into later rows.

Table 3. The Maximum and Median Relative Error When Applying Algorithms 1, 2, 3, and 4 to the Test Problems with $m = 100$ past Iterates

| Identifier | Algorithm 1 | | Algorithm 2 | | Algorithms 3 and 4 | |
|---|---|---|---|---|---|---|
| | max_rel_err | med_rel_err | max_rel_err | med_rel_err | max_rel_err | med_rel_err |
| BQPGAUSS | 7.31E+02 | 2.84E-15 | 4.80E+12 | 3.26E-15 | 1.13E-11 | 1.28E-15 |
| CURLY30 | 6.32E-12 | 4.60E-15 | 1.74E-07 | 9.42E-12 | 6.32E-12 | 4.60E-15 |
| DRCAV1LQ | 9.25E-10 | 5.68E-15 | [a] | [a] | 9.25E-10 | 5.68E-15 |
| JIMACK | 5.27E-10 | 1.40E-14 | [a] | [a] | 5.27E-10 | 1.40E-14 |
| NCVXBQP1 | 2.14E-11 | 8.66E-16 | [a] | 7.31E+02 | 2.14E-11 | 8.66E-16 |
| SINQUAD | 5.81 | 6.79E-04 | 1.04E-10 | 2.40E-16 | 5.28E-11 | 2.13E-16 |
| SPARSINE | 1.65E-10 | 3.68E-14 | [a] | [a] | 1.65E-10 | 3.68E-14 |
| SPARSQUR | 2.44E-10 | 1.04E-14 | [a] | [a] | 2.44E-10 | 1.04E-14 |
| WALL100 | 2.07E-10 | 1.03E-14 | [a] | 2.21E-09 | 2.07E-10 | 1.03E-14 |
| CAR2 | 9.71E-15 | 0 | [a] | [a] | 1.14E-14 | 0 |
| GASOIL | 8.98E-01 | 4.73E-02 | 1.06E-12 | 1.67E-16 | 7.45E-14 | 1.38E-16 |
| LUKVLE12 | 9.49E-01 | 6.06E-16 | 6.45E-10 | 1.60E-15 | 1.40E-12 | 5.77E-16 |
| MSQRTA | 1.95E-13 | 2.28E-15 | [a] | [a] | 1.95E-13 | 2.28E-15 |
| ORTHREGE | 4.87E+01 | 1.36E-01 | 1.14E-11 | 4.44E-16 | 4.55E-13 | 4.44E-16 |
| TWIRIMD1 | 7.05E-01 | 1.40E-14 | [a] | 3.85E-14 | 4.54E-12 | 2.33E-15 |
| YATP1SQ | 2.06 | 1.49E-01 | 4.24E-08 | 9.71E-16 | 1.77E-10 | 6.86E-16 |

[a]indicates error exceeds $10^{15}$.

### 3.3 Results When Allowing for Rows of Different Densities

To illustrate that the block parallel and recursive block parallel algorithms are able to remedy the shortcomings of the row-wise independent and row-wise dependent algorithms we start by assuming sufficient data are available. In Table 3, we report the relative errors when applying Algorithms 3 and 4 in parallel on 28 processor cores with $m = 100$ past iterates. We set the maximum recursion depth to $r_{\max} = 25$; this prevents excessive recursion from hampering parallelism within Algorithm 4. We also set the minimum entries per row to recurse on to $n_{\min} = 10$; this means the dense linear systems are of size at least $10 \times 10$. Whilst the algorithm is sensitive to $r_{\max}$, $n_{\min}$ is almost never triggered for sufficiently small values. The results in Table 3 show that for both algorithms the median relative error is close to machine precision and the maximum relative error is acceptably small, even for problems containing dense rows. In these tests with sufficient past iterates, the errors for both Algorithms 3 and 4 are identical, but this is not always the case; this is illustrated below for the insufficient data regime.

We now turn our attention to runtimes and illustrate the benefits of parallelism. In Table 4, we report the runtime in seconds using a single core and 28 cores. Runtimes are measured in seconds using the Fortran `system_clock` routine. We can that Algorithms 3 and 4 both achieve significant speedup on problems that are not already fast to solve in serial (i.e., have runtimes less than 0.1 seconds), so that in all cases we are able to approximate the Hessian in parallel in under 0.5 seconds.

### 3.4 Results in the Low Accumulated past Data Regime

In this section, we report on the case where we have a low number of past data pairs, i.e., the number of past iterates $m$ is small. We first consider the block parallel algorithm. In Figures 1 and 2, we present the maximum and median relative errors, respectively, when applying Algorithm 3

Table 4. The Runtime (in Seconds) When Applying Algorithms 3 and 4 in Serial (Single Core) and Parallel (28 Cores) with $m = 100$ past Iterates

| Identifier | Algorithm 3 | | | Algorithm 4 | | |
|---|---|---|---|---|---|---|
| | Serial | Parallel | Speedup | Serial | Parallel | Speedup |
| BQPGAUSS | 0.05 | 0.01 | 3.47 | 0.05 | 0.01 | 3.43 |
| CURLY30 | 4.64 | 0.33 | 13.89 | 4.66 | 0.34 | 13.79 |
| DRCAV1LQ | 0.85 | 0.06 | 14.66 | 0.85 | 0.06 | 15.14 |
| JIMACK | 2.01 | 0.15 | 13.77 | 2.01 | 0.15 | 13.14 |
| NCVXBQP1 | 0.53 | 0.04 | 13.00 | 0.53 | 0.05 | 11.57 |
| SINQUAD | 0.01 | 0.01 | 1.86 | 0.01 | 0.01 | 2.00 |
| SPARSINE | 0.71 | 0.04 | 17.32 | 0.71 | 0.05 | 13.58 |
| SPARSQUR | 1.44 | 0.07 | 19.73 | 1.43 | 0.09 | 15.71 |
| WALL100 | 8.30 | 0.41 | 20.24 | 8.29 | 0.43 | 19.46 |
| CAR2 | 0.19 | 0.02 | 8.77 | 0.19 | 0.02 | 8.73 |
| GASOIL | 0.01 | 0.01 | 2.00 | 0.01 | 0.01 | 1.62 |
| LUKVLE12 | 0.04 | 0.01 | 4.78 | 0.04 | 0.01 | 3.91 |
| MSQRTA | 0.52 | 0.06 | 9.34 | 0.52 | 0.05 | 10.08 |
| ORTHREGE | 0.02 | 0.01 | 3.43 | 0.02 | 0.01 | 2.88 |
| TWIRIMD1 | 0.61 | 0.05 | 11.26 | 0.61 | 0.06 | 10.31 |
| YATP1SQ | 0.49 | 0.05 | 9.48 | 0.49 | 0.05 | 10.47 |

For Algorithm 4, the maximum recursion depth is $r_{\max} = 25$ and the minimum entries per row to recurse on is $n_{\min} = 10$.

to the unconstrained (left) and constrained (right) test problems as the number of past iterates $m$ varies from 1 to 100. Corresponding runtimes are given in Figure 3. We see that when there is insufficient past data ($m$ is too small) there is little the algorithm can do to get an accurate Hessian approximation, however once enough data pairs are available ($m$ is sufficiently large), there is a sharp transition to an accurate approximation. This is most clearly seen for problem CURLY30, for which the Hessian matrix is banded with a total bandwidth of 61. As expected, there is a sharp transition in the relative errors after $m = 61$. It is also interesting to see that a large maximum relative error is not necessarily indicative of a poor Hessian approximation. For example, problem BQPGAUSS has large maximum relative error until $m = 92$ but a low median relative error from $m = 11$ onwards. This suggests that the approximation is mostly very good from $m = 11$ onwards, aside from some outlier entries of the matrix that require more data pairs. Looking at the runtime in Figure 3, we see that the times are generally reasonable (under 1 second).

Next, we consider the recursive block parallel algorithm. We focus on the test problems for which recursion shows a clear benefit (for completeness, we include the full numerical results for Algorithm 4 in Appendix B). In Figure 4, we report the maximum (left) and median (right) relative errors when applying Algorithms 3 (dashed line) and 4 (solid line) to the problems for which recursion is advantageous as $m$ varies from 1 to 100. Again, when there is insufficient past data ($m$ too small) it is not possible to achieve an accurate Hessian approximation but once enough data pairs become available ($m$ sufficiently large) Algorithm 4 outperforms Algorithm 3. For example, for problem BQPGAUSS, to achieve a maximum relative error around $10^{-11}$, Algorithm 3 requires $m = 92$ past iterates whereas Algorithm 4 only requires $m = 39$. Similarly, for problem WALL100, Algorithm 3 requires $m = 42$ achieve a maximum relative error around $10^{-8}$ whereas Algorithm 4

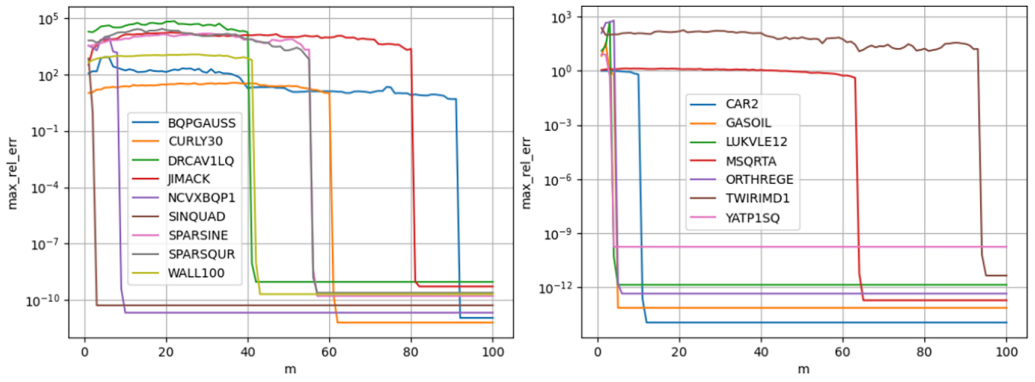


Fig. 1. Maximum relative error when applying Algorithm 3 in parallel on the unconstrained (left) and constrained (right) examples as $m$ varies from 1 to 100.

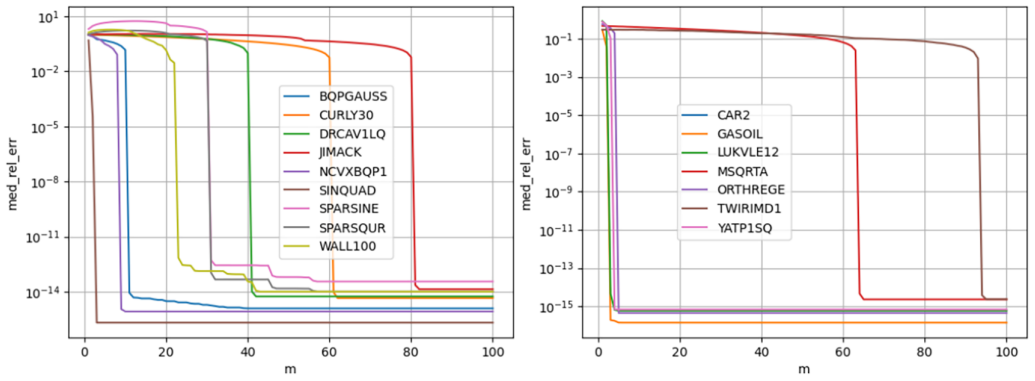


Fig. 2. Median relative error when applying Algorithm 3 in parallel on the unconstrained (left) and constrained (right) examples as $m$ varies from 1 to 100.

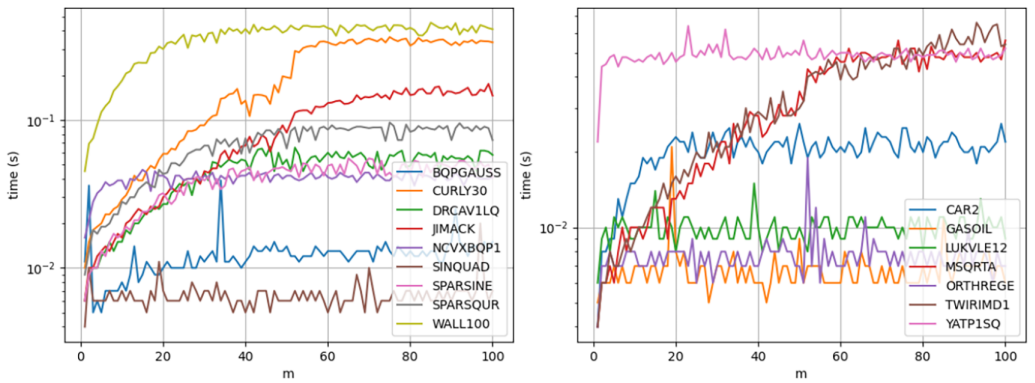


Fig. 3. Runtime (in seconds) when applying Algorithm 3 in parallel on the unconstrained (left) and constrained (right) examples as $m$ varies from 1 to 100.

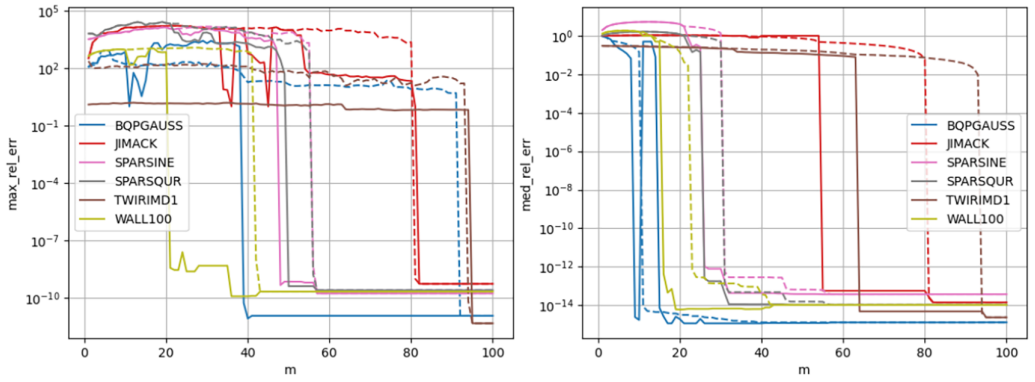


Fig. 4. Maximum relative error (left) and median relative error (right) when applying Algorithm 3 (dashed line) in parallel and Algorithm 4 (solid line) in parallel on problems for which recursion shows a clear benefit as $m$ varies from 1 to 100.

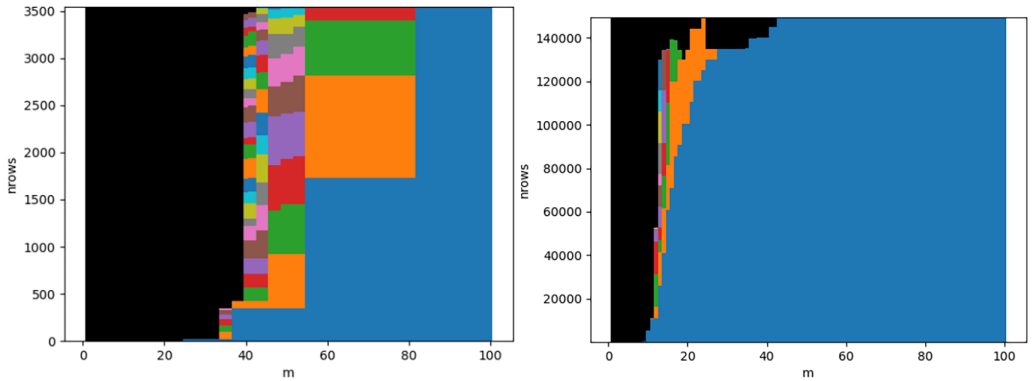


Fig. 5. Number of rows in the block for each level of recursion (each colour) when applying Algorithm 4 in parallel on problem JIMACK (left) and WAL100 (right) as $m$ varies from 1 to 100.

only requires $m = 21$. Much the same behaviour can be observed for the median relative error. For problem JIMACK, Algorithm 3 requires $m = 81$ to achieve a median relative error around $10^{-14}$ whereas Algorithm 4 only requires $m = 55$. For problem TWIRIMD1, the corresponding statistics are $m = 94$ and $m = 64$. These examples illustrate the effectiveness of the primary aim of the recursion in Algorithm 4: to reduce the number of past iterates required to achieve an accurate Hessian approximation by reducing the number of unknown Hessian entries that need to be solved for in each row of the Hessian.

Finally, for the JIMACK and WALL100 test problems, Figure 5 shows the number of rows in each level of recursion within Algorithm 4. The number in the first 'sparse' level $n_1^{[0]}$ (lines 2–4 in Algorithm 4) is depicted in blue, the number in the last 'dense' level $n_2^{[r]}$ (lines 16–19 in Algorithm 4) is depicted in black, and the numbers in the intermediate levels $n_1^{[r]}$ (lines 10–13 in Algorithm 4) are depicted using other colours. Note that for the non-recursive Algorithm 3 we only have the first and last level (blue and black). We see that when $m$ is sufficiently large (i.e., sufficient past iterates are available) no recursion is necessary and all rows are treated as 'sparse' (blue). As $m$ decreases, recursion is required to reduce the number of unknown entries needed to determine in each row of the Hessian, so that we stay within the given budget of $m$ past iterates. Eventually,

$m$ becomes so small that there are not enough past iterates available to accurately estimate the Hessian and all rows are treated as 'dense' (black).

## 4 Conclusions and Future Work

We have presented new methods for computing approximate Hessian matrices from optimization iterate and gradient differences when the sparsity structure is known in advance. The methods are promising and offer the potential to exploit reasonable parallel execution on a modest number of processors. Unlike some earlier approaches, unwarranted growth in matrix entries due to rounding seems to be avoided in practice. The methods are available in the Fortran package SHA, (a C interface is available), as part of the open source GALAHAD library [12].

The next step will be to explore how the new methods behave when embedded inside actual optimization algorithms. In particular, we would like to know if the accumulated Hessian approximations obtained only from gradients lead to good second-order (Newton-like) algorithms. Or in other words, whether the Hessian approximations capture sufficient curvature information to still enable the higher-order convergence exhibited by second-order optimization algorithms. This is the subject of our ongoing research.

Finally, although we have motivated our estimation strategies on $m$ past observed optimization steps $\{s^{(l)}\}_{l=k-m+1}^{k}$ and gradient differences $\{y^{(l)}\}_{l=k-m+1}^{k}$ generated as an optimization algorithm proceeds, they apply equally to methods that judiciously choose steps $s^{(l)}$ and compute approximate $y^{(l)} = (g(x^{(k)} + \Delta s^{(l)}) - g(x^{(k)}))/\Delta$ at a particular point $x^{(k)}$, as is typical of the sparse finite-difference schemes [2, 21] mentioned in the introduction and reported on in our numerical experiments. The application of our estimation strategies to such methods is an interesting avenue for future research.

## Acknowledgements

We are grateful to Coralia Cartis, Lukas Mackinder, Jared Tanner, and Philippe Toint for earlier stimulating discussions. In particular, in 2016, Lukas Mackinder submitted a dissertation for a Master of Science degree at the University of Oxford related to the initial ideas behind this work; the dissertation is not publicly available. We would also like to thank the two anonymous referees for their suggestions in helping to improve the article.

## References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, et al. 1999. *LAPACK Users' Guide*. SIAM.

[2] T. F. Coleman and J. J. Moré. 1984. Estimation of sparse Hessian matrices and graph coloring problems. *Mathematical Programming* 28, 3 (1984), 243–270. DOI: https://doi.org/10.1007/BF02612334

[3] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. 1990. An introduction to the structure of large scale nonlinear optimization problems and the LANCELOT project. In *Computing Methods in Applied Sciences and Engineering*. R. Glowinski and A. Lichnewsky (Eds.), SIAM, 42–51.

[4] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. 1996. Numerical experiments with the LANCELOT package (release A) for large-scale nonlinear optimization. *Mathematical Programming* 73, 1 (1996), 73–110. DOI: https://doi.org/10.1016/0025-5610(95)00054-2

[5] E. Corwin and A. Logar. 2004. Sorting in linear time – variations on the bucket sort. *Journal of Computing Sciences in Colleges* 20, 1 (2004), 197–202.

[6] J. E. Dennis and R. B. Schnabel. 1983. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice Hall, Englewood Cliffs, NJ, *XIII+378 Pages*.

[7] R. Fletcher. 1995. An optimal positive definite update for sparse Hessian matrices. *SIAM Journal on Optimization* 5, 1 (1995), 192–218. DOI: https://doi.org/10.1137/0805010

[8] R. Fletcher, A. Grothey, and S. Leyffer. 1997. Computing sparse Hessian and Jacobian approximations with optimal hereditary properties. In *Large-Scale Optimization with Applications, Part II: Optimal Design and Control*. A. R. Conn,

L. T. Biegler, T. F. Coleman, and F. N. Santosa (Eds.), Springer, New York, Vol. 93, 37–52. DOI: https://doi.org/10.1007/978-1-4612-1960-6_3

[9] J. M. Fowkes, J. A. Scott, and N. I. M. Gould. 2024. Approximating sparse Hessian matrices using large-scale linear least squares. *Numerical Algorithms* 96, 4 (2024), 1675–1698.

[10] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. 1983. Computing forward-difference intervals for numerical optimization. *SIAM Journal on Scientific and Statistical Computing* 4, 2 (1983), 310–321. DOI: https://doi.org/10.1137/0904025

[11] N. I. M. Gould. 2013. *Computing Useful Sparse Hessian Approximations Satisfying Componentwise Secant Equations I: Using a Known Sparsity Pattern*. Working Note RAL 2013-1. STFC-Rutherford Appleton Laboratory, Oxfordshire, UK. Retrieved from https://www.numerical.rl.ac.uk/reports/gNAGIR20131.pdf

[12] N. I. M. Gould, D. Orban, and Ph. L. Toint. 2003. GALAHAD, a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Transactions on Mathematical Software* 29, 4 (2003), 353–372.

[13] N. I. M. Gould, D. Orban, and Ph. L. Toint. 2015. CUTEst: A constrained and unconstrained testing environment with safe threads for mathematical optimization. *Computational Optimization and Applications* 60, 3 (2015), 545–557.

[14] A. Griewank and Ph. L. Toint. 1982. On the unconstrained optimization of partially separable functions. In *Nonlinear Optimization, 1981*. M.J.D. Powell (Ed.), Academic Press, London, 301–312.

[15] A. Griewank and Ph. L. Toint.1982. Partitioned variable metric updates for large structured optimization problems. *Journal of Numerical Mathematics* 39, 1 (1982), 119–137. DOI: https://doi.org/10.1007/BF01399316

[16] A. Griewank and A. Walther. 2008. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation* (2nd. ed.). SIAM, XXII+438 pages. DOI: https://doi.org/10.1137/1.9780898717761

[17] D. C. Liu and J. Nocedal. 1989. On the limited memory BFGS method for large-scale optimization. *Mathematical Programming* 45, 1–3 (1989), 503–528. DOI: https://doi.org/10.1007/BF01589116

[18] S. T. McCormick. 1983. Optimal approximation of sparse Hessians and its equivalence to a graph coloring problem. *Mathematical Programming* 26, 2 (1983), 153–171.

[19] J. Nocedal. 1980. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation* 35, 151 (1980), 773–782. DOI: https://doi.org/10.2307/2006193

[20] J. Nocedal and S. J. Wright. 2006. *Numerical Optimization*. Springer, New York. Retrieved from https://books.google.co.uk/books?id=VbHYoSyelFcC

[21] M. J. D. Powell and Ph. L. Toint. 1979. On the estimation of sparse Hessian matrices. *SIAM Journal on Numerical Analysis* 16, 6 (1979), 1060–1074. DOI: https://doi.org/10.1137/0716078

[22] H. H. Seward. 1954. *Information Sorting in the Application of Electronic Digital Computers to Business Operations*. Technical Report. Report R-232, Digital Computer Laboratory, Massachusetts Institute of Technology, USA.

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

[24] Ph. L. Toint. 1977. On sparse and symmetric matrix updating subject to a linear equation. *Mathematics of Computation* 31, 140 (1977), 954–961. DOI: https://doi.org/10.2307/2006125

[25] Ph. L. Toint. 1978. Some numerical results using a sparse matrix updating formula in unconstrained optimization. *Mathematics of Computation* 32, 143 (1978), 839–851. DOI: https://doi.org/10.2307/2006489

## Appendices

## A Generation of Hessian Matrices Using CUTEst

Here, we describe how the fixed Hessians $H$ for unconstrained and constrained CUTEst problems used in the numerical experiments in Section 3 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 $x_i^{start}$ $(1 \le i \le n)$ is the initial value for component $i$ of $x^{start}$, with lower and upper bounds $x_i^l$ and $x_i^u$, then:

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

Here, $\rho \in (0, 1)$ is the pseudo random number returned by the call `rand(seed,.true.,rho)`, where `rand` is from the GALAHAD library [12] 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 $\mu^{rand}$ returned by `rand(seed,.false.,mu(i))`. We then set $H = H_L^{cutest}(x^{pert}, \mu^{rand})$.

For the finite-difference experiments, we use a relative step size of $10^{-7}x^{pert}$.

## B Complete Numerical Results for Algorithm 4

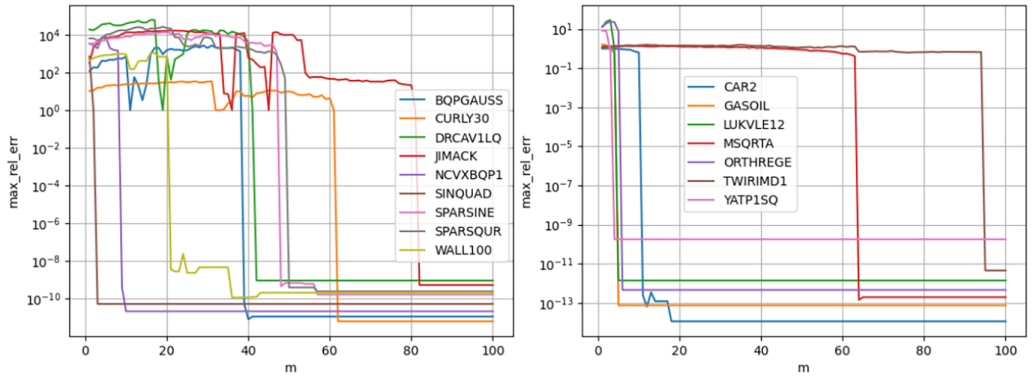


Fig. B1. Maximum relative error when applying Algorithm 4 in parallel on the unconstrained (left) and constrained (right) examples as $m$ varies from 1 to 100.

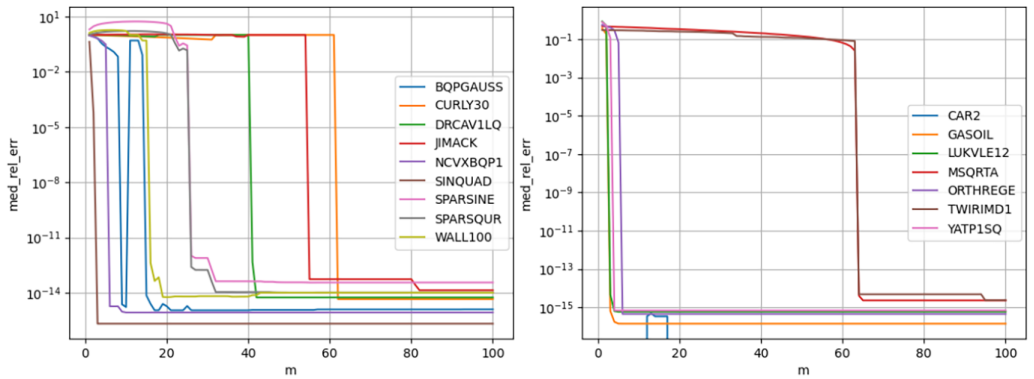


Fig. B2. Median relative error when applying Algorithm 4 in parallel on the unconstrained (left) and constrained (right) examples as $m$ varies from 1 to 100.

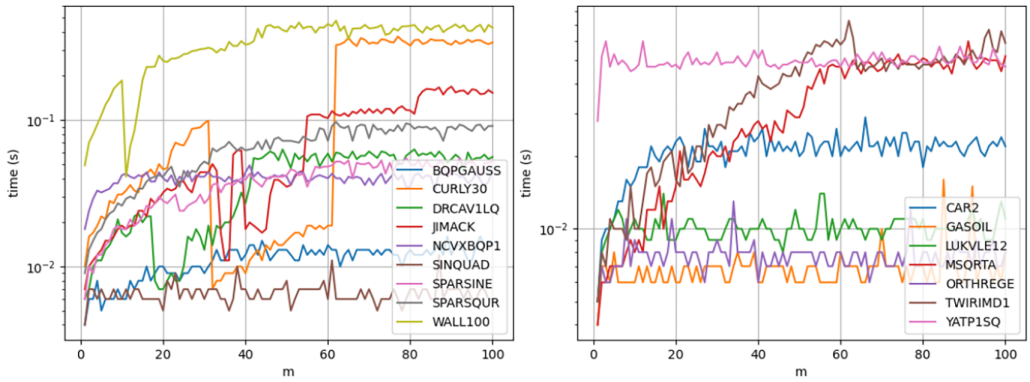


Fig. B3. Runtime (in seconds) when applying Algorithm 4 in parallel on the unconstrained (left) and constrained (right) examples as $m$ varies from 1 to 100.

Received 7 May 2024; revised 9 December 2024; accepted 28 March 2025