Mathematical Programming
https://doi.org/10.1007/s10107-021-01621-6

FULL LENGTH PAPER

Series A

Limited-memory BFGS with displacement aggregation

Albert S. Berahas[1] · **Frank E. Curtis**[1] · **Baoyu Zhou**[1]

Received: 13 January 2020 / Accepted: 4 January 2021

Abstract

A displacement aggregation strategy is proposed for the curvature pairs stored in a limited-memory BFGS (a.k.a. L-BFGS) method such that the resulting (inverse) Hessian approximations are equal to those that would be derived from a full-memory BFGS method. This means that, if a sufficiently large number of pairs are stored, then an optimization algorithm employing the limited-memory method can achieve the same theoretical convergence properties as when full-memory (inverse) Hessian approximations are stored and employed, such as a local superlinear rate of convergence under assumptions that are common for attaining such guarantees. To the best of our knowledge, this is the first work in which a local superlinear convergence rate guarantee is offered by a quasi-Newton scheme that does not either store all curvature pairs throughout the entire run of the optimization algorithm or store an explicit (inverse) Hessian approximation. Numerical results are presented to show that displacement aggregation within an adaptive L-BFGS scheme can lead to better performance than standard L-BFGS.

Keywords Nonlinear optimization · Quasi-Newton algorithms · Broyden–Fletcher–Goldfarb–Shanno (BFGS) · Limited-memory BFGS · Superlinear convergence

Mathematics Subject Classification 49M37 · 65K05 · 65K10 · 90C30 · 90C53

This material is based upon work supported by the National Science Foundation under grant numbers CCF–1618717 and CCF–1740796.

✉ Frank E. Curtis
frank.e.curtis@lehigh.edu

Albert S. Berahas
albertberahas@gmail.com

Baoyu Zhou
baoyu.zhou@lehigh.edu

[1] Department of Industrial and Systems Engineering, Lehigh University, 200 W. Packer Ave., Bethlehem, PA, USA

1 Introduction

Quasi-Newton methods–in which one computes search directions using (inverse) Hessian approximations that are set using iterate and gradient displacement information from one iteration to the next–represent some of the most effective algorithms for minimizing nonlinear objective functions.[1] The main advantages of such methods can be understood by contrasting their computational costs, storage requirements, and convergence behavior with those of steepest descent and Newton methods. Steepest descent methods require only first-order derivative (i.e., gradient) information, but only achieve a local linear rate of convergence. Newton's method can achieve a faster (namely, quadratic) rate of local convergence, but at the added cost of forming and factoring second-order derivative (i.e., Hessian) matrices, or at least at the cost of numerous Hessian-vector products per iteration. Quasi-Newton methods lie between the aforementioned methods; they only require gradient information, yet by updating and employing Hessian approximations they are able to achieve a local superlinear rate of convergence. For many applications, the balance between cost/storage requirements and convergence rate offered by quasi-Newton methods makes them the most effective.

Davidon is credited as being the inventor of quasi-Newton methods [14]. For further information on their theoretical properties, typically when coupled with line searches to ensure convergence, see, e.g., [7,9,15,16,31–35]. Within the class of quasi-Newton methods, algorithms that employ Broyden–Fletcher–Goldfarb–Shanno (BFGS) approximations of Hessian matrices have enjoyed particular success; see [5, 18,20,38]. This is true when minimizing smooth objectives, as is the focus in the aforementioned references, but also when minimizing nonsmooth [4,11,12,23,26,27,39] or stochastic [1,2,6,10,22,24,28,37,40] functions.

For solving large-scale problems, i.e., minimizing objective functions involving thousands or millions of variables, *limited-memory* variants of quasi-Newton methods, such as the limited-memory variant of BFGS [30] (known as L-BFGS), have been successful for various applications. The main benefit of a limited-memory scheme is that one need not store nor compute matrix-vector products with the often-dense Hessian approximation; rather, one need only store a few pairs of vectors, known as *curvature pairs*, with dimension equal to the number of variables, and one can compute matrix-vector products with the corresponding Hessian approximation with relatively low computational cost. That said, one disadvantage of contemporary limited-memory methods is that they do not enjoy the local superlinear convergence rate properties achieved by full-memory schemes. Moreover, one does not know *a priori* what number of pairs should be maintained to attain good performance when solving a particular problem. Recent work [3] has attempted to develop an adaptive limited-memory BFGS method in which the number of stored curvature pairs is updated dynamically within the algorithm. However, while this approach has led to some improved empirical performance, it has not been coupled with any strengthened theoretical guarantees.

[1] Quasi-Newton methods offer the ability to update Hessian and/or inverse Hessian approximations, which is why we state *inverse* parenthetically here. For ease of exposition throughout the remainder of the paper, we often drop mention of the inverse, although in many cases it is the approximation of the inverse, not the Hessian approximation, that is used in practice.

We are motivated by the following question: Is it possible to design an *adaptive* limited-memory BFGS scheme that stores and employs only a *moderate number* of curvature pairs, yet in certain situations can achieve the same theoretical convergence rate guarantees as a full-memory BFGS scheme (such as a local superlinear rate of convergence)? We have not yet answered this question to the fullest extent possible. That said, in this paper, we do answer the following question, which we believe is a significant first step: Is it possible to develop a *limited-memory-type*[2] BFGS scheme that behaves *exactly* as a full-memory BFGS scheme–in the sense that the sequence of Hessian approximations from the full-memory scheme is represented through stored curvature pairs–while storing only a number of curvature pairs *not exceeding the dimension of the problem*? We answer this question in the affirmative by showing how one can use *displacement aggregation* such that curvature pairs computed by the algorithm may be modified adaptively to capture the information that would be stored in a full-memory BFGS approximation.

A straightforward application of the strategy proposed in this paper might not always offer practical advantages over previously proposed full-memory or limited-memory schemes in all settings. Indeed, to attain full theoretical benefits in some settings, one might have to store a number of curvature pairs up to the number of variables in the minimization problem, in which case the storage requirements and computational costs of our strategy might exceed those of a full-memory BFGS method. That said, our displacement aggregation strategy and corresponding theoretical results establish a solid foundation upon which one may design practically efficient approaches. We propose one such approach, leading to an adaptive L-BFGS scheme that we show to outperform a standard L-BFGS approach on a set of test problems. For this and other such adaptive schemes, we argue that displacement aggregation allows one to maintain *more history* in the same number of curvature pairs than in previously proposed limited-memory methods.

We conclude the paper with a discussion of how our approach can be extended to Davidon–Fletcher–Powell (DFP) quasi-Newton updating [14], the challenges of extending it to the entire Broyden class of updates [16], and further ways that displacement aggregation can be employed in practically efficient algorithms.

1.1 Contributions

We propose and analyze a *displacement aggregation* strategy for modifying the displacement pairs stored in an L-BFGS scheme such that the resulting method behaves equivalently to full-memory BFGS. In particular, we show that if a stored iterate displacement vector lies in the span of the other stored iterate displacement vectors, then the gradient displacement vectors can be modified in such a manner that the same Hessian approximation can be generated using the modified pairs with one pair (i.e, the one corresponding to the iterate displacement vector that is in the span of the others) being ignored. In this manner, one can iteratively *throw out* stored pairs and maintain at most a number of pairs equal to the dimension of the problem while generating

[2] By *limited-memory-type* BFGS algorithm, we mean one that stores and employs a finite set of curvature pairs rather than an explicit Hessian approximation.

the same sequence of Hessian approximations that would have been generated in a full-memory BFGS scheme. Employed within a minimization algorithm, this leads to an L-BFGS scheme that has the same convergence behavior as full-memory BFGS; e.g., it can achieve a local superlinear rate of convergence. To the best of our knowledge, this is the first work in which a local superlinear convergence rate guarantee is provided for a limited-memory-type quasi-Newton algorithm. We also show how our techniques can be employed within an adaptive L-BFGS algorithm–storing and employing only a small number of pairs relative to the dimension of the problem–that can outperform standard L-BFGS. We refer to our proposed "aggregated BFGS" scheme as `Agg-BFGS`.

Our proposal of displacement aggregation should be contrasted with the idea of *merging* information proposed in [25]. In this work, the authors prove–when minimizing convex quadratics only–that one can maintain an algorithm that converges finitely if previous pairs are replaced by linear combinations of pairs. This fact motivates a scheme proposed by the authors–when employing an L-BFGS-type scheme in general–of replacing two previous displacement pairs by a single "sum" displacement pair. In their experiments, the authors' idea often leads to reduced linear algebra time, since fewer pairs are stored, but admittedly worse performance compared to standard L-BFGS in terms of function evaluations. By contrast, our approach does not use predetermined weights to merge two pairs into one. Our scheme aggregates information stored in any number of pairs using a strategy to ensure that no information is lost, even if the underlying function is not a convex quadratic. Our experiments show that our approach can often result in reduced iteration and function evaluations compared to standard L-BFGS.

1.2 Notation

Let $\mathbb{R}$ denote the set of real numbers (i.e., scalars), let $\mathbb{R}_{\geq 0}$ (resp., $\mathbb{R}_{>0}$) denote the set of nonnegative (resp., positive) real numbers, let $\mathbb{R}^n$ denote the set of n-dimensional real vectors, and let $\mathbb{R}^{m\times n}$ denote the set of m-by-n-dimensional real matrices. Let $\mathbb{S}^n$ denote the symmetric elements of $\mathbb{R}^{n\times n}$. Let $\mathbb{N} := \{0, 1, 2, \dots\}$ denote the set of natural numbers. Let $\|\cdot\| := \|\cdot\|_2$.

We motivate our proposed scheme in the context of solving

$$\min_{x\in\mathbb{R}^n} \; f(x), \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable. Corresponding to derivatives of f, we define the gradient function $g : \mathbb{R}^n \to \mathbb{R}^n$ and Hessian function $H : \mathbb{R}^n \to \mathbb{S}^n$. In terms of an algorithm for solving (1.1), we append a natural number as a subscript for a quantity to denote its value during an iteration of the algorithm; e.g., for each iteration number $k \in \mathbb{N}$, we denote $f_k := f(x_k)$. That said, when discussing BFGS updating for the Hessian approximations employed within an algorithm, we often simplify notation by referring to a generic set of curvature pairs that may or may not come from consecutive iterations within the optimization algorithm. In such settings, we clarify our notation at the start of each discussion.

1.3 Organization

In Sect. 2, we provide background on BFGS updating. In Sect. 3, we motivate, present, and analyze our displacement aggregation approach. The results of numerical experiments with `Agg-BFGS` are provided in Sects. 4–5. Concluding remarks are given in Sect. 6.

2 Background on BFGS

The main idea of BFGS updating can be described as follows. Let the kth iterate generated by an optimization algorithm be denoted as x_k. After an iterate displacement (or step) s_k has been computed, one sets the subsequent iterate as $x_{k+1} \leftarrow x_k + s_k$. Then, in order to determine the subsequent step s_{k+1}, one uses the minimizer of the quadratic model $m_{k+1} : \mathbb{R}^n \to \mathbb{R}$ given by

$$d_{k+1} \leftarrow \arg\min_{d \in \mathbb{R}^n} m_{k+1}(d), \quad \text{where} \quad m_{k+1}(d) = f_{k+1} + g_{k+1}^T d + \tfrac{1}{2} d^T M_{k+1} d$$

and $M_{k+1} \in \mathbb{R}^{n \times n}$ is a Hessian approximation. In a line search approach, for example, one computes d_{k+1} by minimizing m_{k+1}, then computes $s_{k+1} \leftarrow \alpha_{k+1} d_{k+1} = -\alpha_{k+1} M_{k+1}^{-1} g_{k+1}$ for some $\alpha_{k+1} > 0$. With f_{k+1} and g_{k+1} determined by x_{k+1}, all that remains toward specifying the model m_{k+1} is to choose M_{k+1}. In a quasi-Newton method [14], one chooses M_{k+1} such that it is symmetric–i.e., like the exact Hessian H_{k+1}, it is an element of $\mathbb{S}^n$–and satisfies the *secant equation*

$$M_{k+1} s_k = g_{k+1} - g_k =: y_k. \tag{2.1}$$

Specifically in BFGS, one computes $W_{k+1} := M_{k+1}^{-1}$ to solve

$$\min_{W \in \mathbb{S}^n} \|W - W_k\|_{\mathcal{M}} \quad \text{s.t.} \quad W = W^T \quad \text{and} \quad W y_k = s_k,$$

where $\|\cdot\|_{\mathcal{M}}$ is a weighted Frobenius norm with weights defined by any matrix $\mathcal{M}$ satisfying the secant equation (or, for concreteness, one can imagine the weight matrix being the *average Hessian* between x_k and x_{k+1}). If one chooses $M_0 \succ 0$ and ensures that $s_k^T y_k > 0$ for all $k \in \mathbb{N}$, as is often done by employing a (weak) Wolfe line search, then it follows that $M_k \succ 0$ for all $k \in \mathbb{N}$; see [31].

Henceforth, let us focus on the sequence of inverse Hessian approximations $\{W_k\}$. After all, this sequence, not $\{M_k\}$, is the one that is often computed in practice since, for all $k \in \mathbb{N}$, the minimizer of m_k can be computed as $d_k \leftarrow -W_k g_k$. Moreover, all of our discussions about the inverse Hessian approximations $\{W_k\}$ have corresponding meanings in terms of $\{M_k\}$ since $W_k \equiv M_k^{-1}$ for all $k \in \mathbb{N}$.

2.1 Iterative and compact forms

As is well known, there are multiple ways to construct or merely compute a matrix-vector product with a BFGS inverse Hessian approximation [31]. For our purposes, it will be convenient to refer to two ways of constructing such approximations: an iterative form and a compact form [8]. For convenience, let us temporarily drop from our notation the dependence on the iteration number of the optimization algorithm and instead talk generically about constructing an inverse Hessian approximation $\overline{W} \succ 0$. Regardless of whether one employs all displacements pairs since the start of the run of the optimization algorithm (leading to a *full-memory* approximation) or only a few recent pairs (leading to a *limited-memory* approximation), the approximation can be thought of as being constructed from some initial approximation $W \succ 0$ and a set of iterate and gradient displacements such that all iterate/gradient displacement inner products are positive, i.e.,

$$S = \begin{bmatrix} s_1 & \cdots & s_m \end{bmatrix} \in \mathbb{R}^{n \times m} \tag{2.2a}$$

$$\text{and } Y = \begin{bmatrix} y_1 & \cdots & y_m \end{bmatrix} \in \mathbb{R}^{n \times m} \tag{2.2b}$$

where

$$\rho = \begin{bmatrix} \frac{1}{s_1^T y_1} & \cdots & \frac{1}{s_m^T y_m} \end{bmatrix}^T \in \mathbb{R}^m_{>0}. \tag{2.3}$$

In the iterative form of BFGS updating, one computes a Hessian approximation $\overline{W}$ from an initial matrix $W \succ 0$ and the displacement pairs in (2.2) by initializing $\overline{W} \leftarrow W$, then iteratively setting, for all $j \in \{1, \ldots, m\}$,

$$U_j \leftarrow I - \rho_j y_j s_j^T, \quad V_j \leftarrow \rho_j s_j s_j^T, \quad \text{and} \quad \overline{W} \leftarrow U_j^T \overline{W} U_j + V_j. \tag{2.4}$$

In the context of an optimization algorithm using BFGS updating, the matrix $\overline{W}$ would be set with a single update in each iteration rather than re-generated from scratch in every iteration using historical iterate/gradient displacements. As a function of the inputs, we denote the output as $\overline{W} = \text{BFGS}(W, S, Y)$.

The updates performed in (2.4) correspond to a set of projections and corresponding corrections. In particular, for each $j \in \{1, \ldots, m\}$, the update *projects out* curvature information along the step/direction represented by s_j, then applies a subsequent *correction* based on the gradient displacement represented by y_j; see [13, Appendix B]. In this manner, one can understand the well-known fact that each update in a BFGS scheme involves a rank-two change of the matrix.

Rather than apply the updates iteratively, it has been shown that one can instead construct the BFGS approximation from the initial approximation by combining all low-rank changes directly. The scheme in Algorithm 1, which shows the compact form of the updates, generates the same output as (2.4); see [8].

We note in passing that for computing a matrix-vector product with a BFGS (or L-BFGS) approximation without constructing the approximation matrix itself, it is well-known that one can use the so-called two-loop recursion [30].

Algorithm 1 : BFGS Matrix Construction, Compact Form

Require: $W \succ 0$ and (S, Y) as in (2.2), with ρ as in (2.3).

1: Set $(R, D) \in \mathbb{R}^{m\times m} \times \mathbb{R}^{m\times m}$ with $R_{i,j} \leftarrow s_i^T y_j$ for all (i, j) such that $1 \leq i \leq j \leq m$ and $D_{i,i} \leftarrow s_i^T y_i$ for all $i \in \{1, \ldots, m\}$ (with all other elements being zero), i.e.,

$$R \leftarrow \begin{bmatrix} s_1^T y_1 & \cdots & s_1^T y_m \\ & \ddots & \vdots \\ & & s_m^T y_m \end{bmatrix} \quad \text{and} \quad D \leftarrow \begin{bmatrix} s_1^T y_1 & & \\ & \ddots & \\ & & s_m^T y_m \end{bmatrix}. \tag{2.5}$$

2: Set

$$\overline{W} \leftarrow W + \begin{bmatrix} S & WY \end{bmatrix} \begin{bmatrix} R^{-T}(D + Y^T W Y)R^{-1} & -R^{-T} \\ -R^{-1} & 0 \end{bmatrix} \begin{bmatrix} S^T \\ Y^T W \end{bmatrix}. \tag{2.6}$$

3: **return** $\overline{W} \equiv \text{BFGS}(W, S, Y)$

2.2 Convergence and local rate guarantees

An optimization algorithm using a BFGS scheme for updating Hessian approximations can be shown to converge and achieve a local superlinear rate of convergence under assumptions that are common for attaining such guarantees. This theory for BFGS relies heavily on so-called *bounded deterioration* of the updates or the *self-correcting* behavior of the updates, and, for attaining a fast local rate of convergence, on the ability of the updating scheme to satisfy the well-known *Dennis-Moré condition* for superlinear convergence. See [15,16] for further information.

We show in the next section that our `Agg-BFGS` approach generates the same sequence of matrices as full-memory BFGS. Hence, an optimization algorithm that employs our updating scheme maintains *all* of the convergence and convergence rate properties of full-memory BFGS. For concreteness, we state one such result as the following theorem, which follows from Theorems 6.5 and 6.6 in [31]. We cite this result later in the paper to support our claim that our `Agg-BFGS` scheme is a limited-memory-type quasi-Newton approach that can be used to achieve local superlinear convergence for an optimization algorithm.

Theorem 2.1 *Suppose that f is twice continuously differentiable and that one employs an algorithm that generates a sequence of iterates $\{x_k\}$ according to $d_k \leftarrow -W_k g_k$ and $x_{k+1} \leftarrow x_k + \alpha_k d_k$ where $\{W_k\}$ is generated using the BFGS updating scheme and, for all $k \in \mathbb{N}$, the stepsize $\alpha_k \in \mathbb{R}_{>0}$ is computed from a line search (initialized with a unit stepsize) to satisfy the Armijo-Wolfe conditions; see equation (3.6) in [31]. In addition, suppose that the algorithm converges to a point $x_* \in \mathbb{R}^n$ at which the Hessian is positive definite and Lipschitz continuous. Then, $\{x_k\}$ converges to x_* at a superlinear rate.*

Such a result cannot be proved for L-BFGS [30]. Common theoretical results for L-BFGS merely show that if the pairs employed have $s_k^T y_k$ sufficiently positive and bounded above for all $k \in \mathbb{N}$, then the Hessian approximations are sufficiently positive

definite and bounded and one can achieve a local linear rate of convergence, i.e., no better than the rate offered by a steepest descent method.

3 Displacement aggregation

We begin this section by proving a simple, yet noteworthy result about a consequence that occurs when one makes consecutive BFGS updates with iterate displacements that are linearly dependent. We use this result and other empirical observations to motivate our proposed approach, which is stated in this section. We close this section by proving that our approach is well defined, and we discuss how to implement it in an efficient manner.

3.1 Motivation: parallel consecutive iterate displacements

The following theorem shows that if one finds in BFGS that an iterate displacement is a multiple of the previous one, then one can skip the update corresponding to the prior displacement and obtain the same inverse Hessian approximation.

Theorem 3.1 *Let (S, Y) be defined as in (2.2), with ρ as in (2.3), and suppose that $s_j = \tau s_{j+1}$ for some $j \in \{1, \ldots, m-1\}$ and some nonzero $\tau \in \mathbb{R}$. Then, with*

$$\tilde{S} := \begin{bmatrix} s_1 & \cdots & s_{j-1} & s_{j+1} & \cdots & s_m \end{bmatrix}$$
$$\text{and } \tilde{Y} := \begin{bmatrix} y_1 & \cdots & y_{j-1} & y_{j+1} & \cdots & y_m \end{bmatrix},$$

Algorithm 1 yields $\text{BFGS}(W, S, Y) = \text{BFGS}(W, \tilde{S}, \tilde{Y})$ *for any* $W \succ 0$.

Proof Let $W \succ 0$ be chosen arbitrarily. For any $j \in \{1, \ldots, m-1\}$, let $W_{1:j} := \text{BFGS}(W, S_{1:j}, Y_{1:j})$ where $S_{1:j} := \begin{bmatrix} s_1 & \cdots & s_j \end{bmatrix}$ and $Y_{1:j} := \begin{bmatrix} y_1 & \cdots & y_j \end{bmatrix}$. By (2.4),

$$W_{1:j+1} = U_{j+1}^T U_j^T W_{1:j-1} U_j U_{j+1} + U_{j+1}^T V_j U_{j+1} + V_{j+1}. \tag{3.1}$$

Since $s_j = \tau s_{j+1}$, it follows that

$$\begin{aligned} U_j U_{j+1} &= (I - \rho_j y_j s_j^T)(I - \rho_{j+1} y_{j+1} s_{j+1}^T) \\ &= \left(I - \left(\frac{1}{\tau s_{j+1}^T y_j}\right) \tau y_j s_{j+1}^T\right)(I - \rho_{j+1} y_{j+1} s_{j+1}^T) \\ &= I - \left(\frac{1}{s_{j+1}^T y_j}\right) y_j s_{j+1}^T - \rho_{j+1} y_{j+1} s_{j+1}^T \\ &\quad + \left(\frac{1}{s_{j+1}^T y_j}\right) \rho_{j+1} y_j (s_{j+1}^T y_{j+1}) s_{j+1}^T \\ &= I - \rho_{j+1} y_{j+1} s_{j+1}^T = U_{j+1} \end{aligned}$$

and that

$$\begin{aligned} V_j U_{j+1} &= (\rho_j s_j s_j^T)(I - \rho_{j+1} y_{j+1} s_{j+1}^T) \\ &= \left(\frac{1}{\tau s_{j+1}^T y_j}\right) \tau^2 s_{j+1} s_{j+1}^T (I - \rho_{j+1} y_{j+1} s_{j+1}^T) \\ &= \left(\frac{1}{s_{j+1}^T y_j}\right) \tau \left(s_{j+1} s_{j+1}^T - \rho_{j+1} s_{j+1} (s_{j+1}^T y_{j+1}) s_{j+1}^T\right) = 0. \end{aligned}$$

Combining these facts with (3.1), one finds that

$$W_{1:j+1} = U_{j+1}^T W_{1:j-1} U_{j+1} + V_{j+1},$$

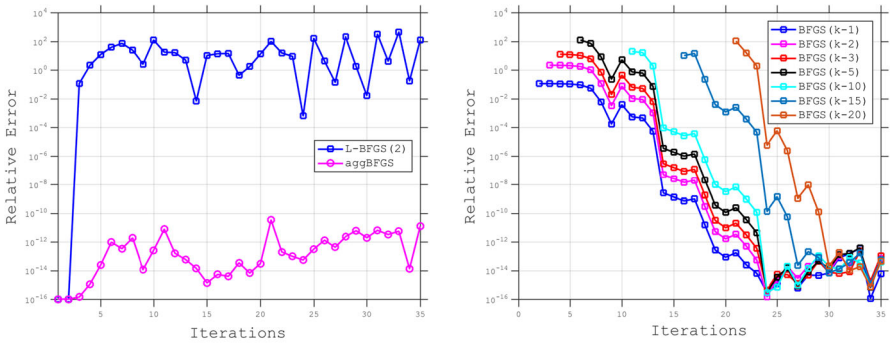
meaning that one obtains the same matrix by applying the updates corresponding to (s_j, y_j) and (s_{j+1}, y_{j+1}) as when one skips the update for (s_j, y_j) and only applies the one for (s_{j+1}, y_{j+1}). The result now follows by the fact that, if one starts with the same initial matrix $W_{1:j+1}$, then applying the updates defined by (2.4) with the same pairs $\{(s_{j+2}, y_{j+2}), \ldots, (s_m, y_m)\}$ yields the same result. □

Theorem 3.1 is a consequence of the *over-writing* process that signifies BFGS. That is, with the update associated with each curvature pair, the BFGS update over-writes curvature information along the direction corresponding to each iterate displacement. The theorem shows that if two consecutive iterate displacement vectors are linearly dependent, then the latter update completely over-writes the former–regardless of the gradient displacement vectors–and as a result the same matrix is derived if the former update is skipped.

What if a previous iterate displacement is not parallel with a subsequent displacement, but is in the span of multiple subsequent displacements? One might suspect that the information in the previous displacement might get over-written and can be ignored. *It is not so simple*. We illustrate this in two ways.

– Suppose that one accumulates curvature pairs $\{(s_j, y_j)\}_{j=0}^k$ and compares the corresponding BFGS approximations with those generated by an L-BFGS scheme with a history length of n. (Hereafter, we refer to the latter scheme as L-BFGS(n).) Suppose also that for $k > n$ one finds that the latest n iterate displacements span $\mathbb{R}^n$. If this fact meant that the updates corresponding to these pairs would over-write all curvature information from previous pairs, then one would find no difference between BFGS and L-BFGS(n) approximations for $k > n$. However, one does not find this to be the case. In Fig. 1a, we plot the maximum absolute difference between corresponding matrix entries of the BFGS and L-BFGS(2) inverse Hessian approximations divided by the largest absolute value of an element of the BFGS inverse Hessian approximation when an algorithm is employed to minimize the Rosenbrock function [36]

$$f(x_{(1)}, x_{(2)}) = 100(x_{(2)} - x_{(1)}^2)^2 + (1 - x_{(1)})^2. \tag{3.2}$$



(a) L-BFGS(2) and `Agg-BFGS` vs. BFGS **(b)** BFGS$(k-j)$ vs. BFGS

Fig. 1 Illustrations that the over-writing of curvature information in BFGS inverse Hessian approximation updating is not absolute, even when previous iterate displacements lie in the span of subsequent iterate displacements. Relative error is the maximum absolute difference between corresponding matrix entries divided by the largest absolute value of an element of the BFGS inverse Hessian approximation

The pairs are generated by running an optimization algorithm using BFGS updating with an Armijo-Wolfe line search; the different approximations were computed as side computations. It was verified that for $k > 2$ the iterate displacements used to generate the L-BFGS(2) matrices were linearly independent. One finds that the differences between the approximations is large for $k > n$. For the purposes of comparison, we also plot the differences between the BFGS inverse Hessian approximations and those generated by our `Agg-BFGS` scheme described later; these are close to machine precision.

– Another manner in which one can see that historical information contained in a BFGS approximation is not always completely over-written once subsequent iterate displacements span $\mathbb{R}^n$ is to compare BFGS approximations with those generated by a BFGS scheme that starts j *iterations late*. Let us refer to such a scheme as BFGS$(k-j)$. For example, for $j = 1$, BFGS$(k-1)$ approximations employ all pairs since the beginning of the run of the algorithm *except* the first one. By observing the magnitude of the differences in the approximations as k increases, one can see how long the information from the first pair *lingers* in the BFGS approximation. In Fig. 1b, we plot the maximum absolute difference between corresponding matrix entries of the BFGS and BFGS$(k-j)$ inverse Hessian approximations divided by the largest absolute value of an element of the BFGS inverse Hessian approximation for various values of j using the same pairs generated by the algorithm from the previous bullet. We only plot once the corresponding matrices start to deviate, i.e., for the BFGS and BFGS$(k-j)$ approximations the matrices only start to differ at the $j+1$st iteration. One finds, e.g., that the information from the first pair lingers–in the sense that it influences the BFGS approximation–for iteration numbers beyond n, which in this example is equal to 2.

On the other hand, one might expect that the information stored in a full-memory BFGS matrix *could be* contained in a set of at most n curvature pairs, although not necessarily a subset of the pairs that are generated by the optimization algorithm. For

example, this might be expected by recalling the fact that if a BFGS method with an exact line search is used to minimize a strongly convex quadratic function, then the Hessian of the quadratic can be recovered *exactly* if n iterations are performed; see [16,25,31]. Letting W represent a BFGS matrix–perhaps computed from more than n pairs–this means that one could run an auxiliary BFGS method to minimize $x^T W x$ to *re-generate* W using at most n pairs. By noting the equivalence between the iterates generated by this auxiliary BFGS scheme and the conjugate gradient (CG) method, one finds that the modified iterate displacements would lie in a certain Krylov subspace determined by the initial point and the matrix W [16,31].

Practically speaking, it would not be computationally efficient to run an entire auxiliary BFGS scheme (for minimizing W) in order to capture W using a smaller number of curvature pairs. Moreover, we are interested in a scheme that can be used to reduce the number of curvature pairs that need to be stored even when an iterate displacement lies in the span of, say, only $m < n$ subsequent displacements. Our `Agg-BFGS` scheme is one efficient approach for achieving this goal.

3.2 Basics of `Agg-BFGS`

The basic building block of our proposed scheme can be described as follows. Suppose that, in addition to the iterate/gradient displacement information $(S, Y) \equiv ([s_1 \ \cdots \ s_m], [y_1 \ \cdots \ y_m]) =: (S_{1:m}, Y_{1:m})$ as defined in (2.2) with inner products yielding (2.3), one also has a previous curvature pair

$$(s_0, y_0) \in \mathbb{R}^n \times \mathbb{R}^n \ \text{ with } \ \rho_0 = 1/s_0^T y_0 > 0 \tag{3.3}$$

such that

$$s_0 = S_{1:m}\tau \ \text{ for some } \ \tau \in \mathbb{R}^m. \tag{3.4}$$

(As in the Proof of Theorem 3.1, we have introduced the subscript "$1 : m$" for S and Y to indicate the dependence of these matrices on quantities indexed 1 through m. We make use of similar notation throughout the rest of the paper.) Given the linear dependence of the iterate displacement s_0 on the iterate displacements in $S_{1:m}$, our goal is to determine *aggregated* gradient displacements

$$\tilde{Y}_{1:m} := \left[\tilde{y}_1 \ \cdots \ \tilde{y}_m\right] \tag{3.5}$$

such that, for a given initial matrix $W \succ 0$, one finds

$$\text{BFGS}(W, S_{0:m}, Y_{0:m}) = \text{BFGS}(W, S_{1:m}, \tilde{Y}_{1:m}). \tag{3.6}$$

That is, our goal is to determine $\tilde{Y}_{1:m}$ such that the matrix $\text{BFGS}(W, S_{0:m}, Y_{0:m})$ is equivalently generated by ignoring (s_0, y_0) and employing $(S_{1:m}, \tilde{Y}_{1:m})$.

Remark 1 For simplicity, we are presenting the basics of `Agg-BFGS` using indices for the iterate and gradient displacement vectors from 0 to m. However, these

need not correspond exactly to the iterates of the outer optimization algorithm. In other words, our strategy and corresponding theoretical results apply equally for $(S, Y) \equiv ([s_{k_1} \ \cdots \ s_{k_m}], [y_{k_1} \ \cdots \ y_{k_m}])$ where $\{k_1, \ldots, k_m\}$ are some (not necessarily consecutive) iteration numbers in the outer optimization algorithm. We remark on this generality further when presenting the implementation of our scheme within a complete optimization algorithm; see Sect. 3.4.

A basic view of our approach is stated as Algorithm 2, wherein we invoke the compact form of BFGS updates (recall (2.6)). However, while Algorithm 2 provides an understanding of the intent of our displacement aggregation scheme, it does not specify the manner in which $\tilde{Y}_{1:m}$ yielding (3.6) can be found. It also does not address the fact that multiple such matrices might exist. Toward a specific scheme, one can expand the compact forms of BFGS$(W, S_{0:m}, Y_{0:m})$ and BFGS$(W, S_{1:m}, \tilde{Y}_{1:m})$, specify that the aggregated displacements have the form

$$\tilde{Y}_{1:m} = W^{-1} S_{1:m} \begin{bmatrix} A & 0 \end{bmatrix} + y_0 \begin{bmatrix} b \\ 0 \end{bmatrix}^T + Y_{1:m} \tag{3.7}$$

for some $A \in \mathbb{R}^{m \times (m-1)}$ and $b \in \mathbb{R}^{m-1}$, and compare like terms in order to derive three key equations that ensure that (3.6) holds. The zero blocks in the variable matrices in (3.7) are motivated by Theorem 3.1, since in the case of $m = 1$ one can set $\tilde{Y}_{1:1} = Y_{1:1}$; otherwise, A and b need to be computed to satisfy (3.6).

Algorithm 2 : Displacement Aggregation, Basic View

Require: $W \succ 0$, $(S_{1:m}, Y_{1:m}, \rho_{1:m})$ as in (2.2)–(2.3), and (s_0, y_0, ρ_0) as in (3.3)–(3.4).

1: Set $\tilde{Y}_{1:m}$ as in (3.5) such that, with

$$\tilde{R}_{1:m} := \begin{bmatrix} s_1^T \tilde{y}_1 & \cdots & s_1^T \tilde{y}_m \\ & \ddots & \vdots \\ & & s_m^T \tilde{y}_m \end{bmatrix} \quad \text{and} \quad \tilde{D}_{1:m} := \begin{bmatrix} s_1^T \tilde{y}_1 & & \\ & \ddots & \\ & & s_m^T \tilde{y}_m \end{bmatrix}, \tag{3.8}$$

one finds with

$$R_{0:m} := \begin{bmatrix} s_0^T y_0 & s_0^T Y_{1:m} \\ & R_{1:m} \end{bmatrix} \quad \text{and} \quad D_{0:m} := \begin{bmatrix} s_0^T y_0 & \\ & D_{1:m} \end{bmatrix}$$

that

$$\begin{aligned} &\text{BFGS}(W, S_{0:m}, Y_{0:m}) \\ &\equiv W + \begin{bmatrix} S_{0:m} & W Y_{0:m} \end{bmatrix} \begin{bmatrix} R_{0:m}^{-T}(D_{0:m} + Y_{0:m}^T W Y_{0:m}) R_{0:m}^{-1} & -R_{0:m}^{-T} \\ -R_{0:m}^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_{0:m}^T \\ Y_{0:m}^T W \end{bmatrix} \\ &= W + \begin{bmatrix} S_{1:m} & W \tilde{Y}_{1:m} \end{bmatrix} \begin{bmatrix} \tilde{R}_{1:m}^{-T}(\tilde{D}_{1:m} + \tilde{Y}_{1:m}^T W \tilde{Y}_{1:m}) \tilde{R}_{1:m}^{-1} & -\tilde{R}_{1:m}^{-T} \\ -\tilde{R}_{1:m}^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_{1:m}^T \\ \tilde{Y}_{1:m}^T W \end{bmatrix} \\ &\equiv \text{BFGS}(W, S_{1:m}, \tilde{Y}_{1:m}). \end{aligned}$$

2: **return** $\tilde{Y}_{1:m}$.

Following these steps, one obtains the more detailed view of our scheme that is stated in Algorithm 3, where the three key equations are stated as (3.9). Algorithm 3 does not specify a precise scheme for computing (A, b) in order to satisfy (3.9). We specify such a scheme in Sect. 3.4 after first showing in the following subsection that real values for (A, b) always exist to satisfy (3.9).

Algorithm 3 : Displacement Aggregation, Detailed View

Require: $W \succ 0$, $(S_{1:m}, Y_{1:m}, \rho_{1:m})$ as in (2.2)–(2.3), and (s_0, y_0, ρ_0) as in (3.3)–(3.4).
1: Set $\tilde{Y}_{1:m}$ as in (3.7) such that, with $\chi_0 := 1 + \rho_0 \|y_0\|_W^2$, one finds

$$\tilde{R}_{1:m} = R_{1:m}, \tag{3.9a}$$

$$\begin{bmatrix} b \\ 0 \end{bmatrix} = -\rho_0 (S_{1:m}^T Y_{1:m} - R_{1:m})^T \tau, \quad \text{and} \tag{3.9b}$$

$$\begin{aligned} (\tilde{Y}_{1:m} - Y_{1:m})^T W (\tilde{Y}_{1:m} - Y_{1:m}) = {} & \frac{\chi_0}{\rho_0} \begin{bmatrix} b \\ 0 \end{bmatrix} \begin{bmatrix} b \\ 0 \end{bmatrix}^T \\ & - \begin{bmatrix} A & 0 \end{bmatrix}^T (S_{1:m}^T Y_{1:m} - R_{1:m}) \\ & - (S_{1:m}^T Y_{1:m} - R_{1:m})^T \begin{bmatrix} A & 0 \end{bmatrix}. \end{aligned} \tag{3.9c}$$

2: **return** $\tilde{Y}_{1:m}$.

3.3 Existence of real solutions for `Agg-BFGS`

Our goal now is to prove the following theorem for our `Agg-BFGS` scheme.

Theorem 3.2 *Suppose one has $W \succ 0$ along with:*

- *$(S_{1:m}, Y_{1:m})$ as defined in (2.2) with the columns of $S_{1:m}$ being linearly independent and the vector $\rho_{1:m}$ as defined in (2.3), and*
- *(s_0, y_0, ρ_0) defined as in (3.3) such that (3.4) holds for some $\tau \in \mathbb{R}^m$.*

Then, there exist $A \in \mathbb{R}^{m \times (m-1)}$ and $b \in \mathbb{R}^{m-1}$ such that, with $\tilde{Y}_{1:m} \in \mathbb{R}^{n \times m}$ defined as in (3.7), the equations (3.9) hold. Consequently, for this $\tilde{Y}_{1:m}$, one finds that

$$s_i^T \tilde{y}_i = s_i^T y_i > 0 \ \text{ for all } \ i \in \{1, \ldots, m\} \tag{3.10}$$

and $\text{BFGS}(W, S_{0:m}, Y_{0:m}) = \text{BFGS}(W, S_{1:m}, \tilde{Y}_{1:m})$.

Proof First, observe that there are $(m+1)(m-1)$ unknowns in the formula (3.7) for $\tilde{Y}_{1:m}$; in particular, there are $m(m-1)$ unknowns in A and $m-1$ unknowns in b, yielding $m(m-1) + (m-1) = (m+1)(m-1)$ unknowns in total. To see the number of equations effectively imposed by (3.9), first notice that (3.7) imposes $\tilde{y}_m = y_m$.

Hence, by ignoring the last column of $R_{1:m}$ to define the submatrix

$$P := \begin{bmatrix} s_1^T y_1 & \cdots & s_1^T y_{m-1} \\ 0 & \ddots & \vdots \\ \vdots & \ddots & s_{m-1}^T y_{m-1} \\ 0 & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{m \times (m-1)}, \tag{3.11}$$

and similarly defining $\tilde{P}$ with size $m \times (m-1)$ as a submatrix of $\tilde{R}_{1:m}$, the key equations in (3.9) can be simplified to have the following form:

$$\tilde{P} = P, \tag{3.12a}$$

$$b = -\rho_0 (S_{1:m}^T Y_{1:m-1} - P)^T \tau, \tag{3.12b}$$

$$\text{and} \quad (\tilde{Y}_{1:m-1} - Y_{1:m-1})^T W (\tilde{Y}_{1:m-1} - Y_{1:m-1})$$
$$= \frac{\chi_0}{\rho_0} b b^T - A^T (S_{1:m}^T Y_{1:m-1} - P) - (S_{1:m}^T Y_{1:m-1} - P)^T A. \tag{3.12c}$$

Observing the number of nonzero entries in (3.12a) (recall (3.11)) and recognizing the symmetry in (3.12c), one finds that the effective number of equations are:

$$\underbrace{m(m-1)/2}_{\text{(in (3.12a))}} + \underbrace{(m-1)}_{\text{(in (3.12b))}} + \underbrace{m(m-1)/2}_{\text{(in (3.12c))}} = \underbrace{(m+1)(m-1)}_{\text{(in (3.12))}}.$$

Hence, (3.12) (and so (3.9)) is a square system of linear and quadratic equations to be solved for the unknowns in the matrix A and vector b.

Equation (3.12b) represents a formula for $b \in \mathbb{R}^{m-1}$. Henceforth, let us assume that b is equal to the right-hand side of this equation, meaning that all that remains is to determine that a real solution for A exists. Let us write

$$A = \begin{bmatrix} a_1 & \cdots & a_{m-1} \end{bmatrix} \quad \text{where } a_i \text{ has length } m \text{ for all } \; i \in \{1, \ldots, m-1\}.$$

Using this notation, one finds from (3.7) that the jth column of (3.12a) requires

$$S_{1:j}^T y_j = S_{1:j}^T \tilde{y}_j \iff S_{1:j}^T y_j = S_{1:j}^T (W^{-1} S_{1:m} a_j + b_j y_0 + y_j);$$

hence, (3.12a) reduces to the system of affine equations

$$S_{1:j}^T W^{-1} S_{1:m} a_j = -b_j S_{1:j}^T y_0 \;\; \text{for all} \;\; j \in \{1, \ldots, m-1\}. \tag{3.13}$$

For each $j \in \{1, \ldots, m-1\}$, let us write

$$a_j = Q^{-1} \begin{bmatrix} a_{j,1} \\ a_{j,2} \end{bmatrix}, \;\; \text{where } Q := S_{1:m}^T W^{-1} S_{1:m} \succ 0, \tag{3.14}$$

with $a_{j,1}$ having length j and $a_{j,2}$ having length $m-j$. (Notice that Q is positive definite under the conditions of the theorem, which requires that $S_{1:m}$ has full column rank.) Then, in order for (3.13) to be satisfied, one must have

$$a_{j,1} = -b_j S_{1:j}^T y_0 \in \mathbb{R}^j. \tag{3.15}$$

Moreover, with this value for $a_{j,1}$, it follows that (3.13), and hence (3.12a), is satisfied for any $a_{j,2}$. Going forward, our goal is to show the existence of $a_{j,2} \in \mathbb{R}^{m-j}$ for all $j \in \{1, \ldots, m-1\}$ such that (3.12c) holds, completing the proof.

Observing (3.12c), one finds with (3.7) that it may be written as

$$A^T Q A + \Omega^T A + A^T \Omega - \omega\omega^T = 0, \tag{3.16}$$

where

$$\Omega := S_{1:m}^T y_0 b^T + S_{1:m}^T Y_{1:m-1} - P \in \mathbb{R}^{m\times(m-1)}, \tag{3.17a}$$

$$\text{and } \omega := \frac{b}{\sqrt{\rho_0}} \in \mathbb{R}^{m-1}. \tag{3.17b}$$

One may rewrite equation (3.16) as

$$(QA+\Omega)^T Q^{-1}(QA+\Omega) = \omega\omega^T + \Omega^T Q^{-1}\Omega. \tag{3.18}$$

Let us now rewrite the equations in (3.18) in a particular form that will be useful for the purposes of our proof going forward. Consider the matrix $QA+\Omega$ in (3.18). By the definitions of $a_{j,1}$, $a_{j,2}$, Q, and Ω in (3.14)–(3.17), as well as of P from (3.11), the jth column of this matrix is given by

$$\begin{aligned}[QA+\Omega]_j &= \begin{bmatrix} -b_j S_{1:j}^T y_0 \\ a_{j,2} \end{bmatrix} + \begin{bmatrix} b_j S_{1:j}^T y_0 \\ b_j S_{j+1:m}^T y_0 \end{bmatrix} + \begin{bmatrix} 0_j \\ S_{j+1:m}^T y_j \end{bmatrix} \\ &= \begin{bmatrix} 0_j \\ a_{j,2} \end{bmatrix} + \begin{bmatrix} 0_j \\ S_{j+1:m}^T (b_j y_0 + y_j) \end{bmatrix},\end{aligned}$$

where 0_j is a vector of zeros of length j. Letting $L \in \mathbb{R}^{m\times m}$ be any matrix such that $L^T L = Q^{-1}$ (whose existence follows since $Q^{-1} \succ 0$), defining $Z := [z_1 \ \cdots \ z_{m-1}] \in \mathbb{R}^{(m-1)\times(m-1)}$ as any matrix such that $Z^T Z = \omega\omega^T + \Omega^T Q^{-1}\Omega$ (whose existence follows since $\omega\omega^T + \Omega^T Q^{-1}\Omega \succeq 0$), and defining, for all $j \in \{1, \ldots, m-1\}$,

$$\phi_j(a_{j,2}) := L\left(\begin{bmatrix} 0_j \\ a_{j,2} \end{bmatrix} + \begin{bmatrix} 0_j \\ S_{j+1:m}^T (b_j y_0 + y_j) \end{bmatrix}\right), \tag{3.19}$$

it follows that the $(i, j) \in \{1, \ldots, m-1\} \times \{1, \ldots, m-1\}$ element of (3.18) is

$$\phi_i(a_{i,2})^T \phi_j(a_{j,2}) = z_i^T z_j. \tag{3.20}$$

Using the notation from the preceding paragraph, let us use an inductive argument to prove the existence of a real solution of (3.18). This induction will follow the indices $\{1, \ldots, m-1\}$ *in reverse order*. As a base case, consider the index $m-1$, in which case one has the one-dimensional unknown $a_{m-1,2}$. One finds with

$$a^*_{m-1,2} := -s_m^T(b_{m-1}y_0 + y_{m-1}) \in \mathbb{R}$$

that

$$\phi_{m-1}(a^*_{m-1,2}) = L\begin{bmatrix} 0_{m-1} \\ -s_m^T(b_{m-1}y_0 + y_{m-1}) + s_m^T(b_{m-1}y_0 + y_{m-1}) \end{bmatrix} = 0_m.$$

Hence, letting $a_{m-1,2} = a^*_{m-1,2} + \lambda_{m-1}$, where λ_{m-1} is one-dimensional, one finds that the left-hand side of the $(i, j) = (m-1, m-1)$ equation in (3.20) is $\|\phi_{m-1}(a_{m-1,2})\|_2^2$. This is a strongly convex quadratic in the unknown λ_{m-1}. Since $\|\phi_{m-1}(a^*_{m-1,2})\|_2 = 0$ and $z_{m-1}^T z_{m-1} \geq 0$, there exists $\lambda^*_{m-1} \in \mathbb{R}$ such that $a_{m-1,2} = a^*_{m-1,2} + \lambda^*_{m-1} \in \mathbb{R}$ satisfies the $(i, j) = (m-1, m-1)$ equation in (3.20).

Now suppose that there exists real $\{a_{\ell+1,2}, \ldots, a_{m-1,2}\}$ such that (3.20) holds for all (i, j) with $i \in \{\ell+1, \ldots, m-1\}$ and $j \in \{i, \ldots, m-1\}$. (By symmetry, these values also satisfy the (j, i) elements of (3.20) for these same values of the indices i and j.) To complete the inductive argument, we need to show that this implies the existence of $a_{\ell,2} \in \mathbb{R}^{m-\ell}$ satisfying (3.20) for all (i, j) with $i \in \{j, \ldots, m-1\}$ and $j = \ell$, i.e., solving the following system for $a_{\ell,2}$:

$$\phi_{m-1}(a_{m-1,2})^T \phi_\ell(a_{\ell,2}) = z_{m-1}^T z_\ell \tag{3.21a}$$

$$\vdots$$

$$\phi_{\ell+1}(a_{\ell+1,2})^T \phi_\ell(a_{\ell,2}) = z_{\ell+1}^T z_\ell \tag{3.21b}$$

$$\phi_\ell(a_{\ell,2})^T \phi_\ell(a_{\ell,2}) = z_\ell^T z_\ell. \tag{3.21c}$$

Notice that (3.21a)–(3.21b) are affine equations in $a_{\ell,2}$, whereas (3.21c) is a quadratic equation in $a_{\ell,2}$. For all $t \in \{\ell+1, \ldots, m-1\}$, let

$$\psi_{\ell+1,t} := \begin{bmatrix} 0_{t-(\ell+1)} \\ a_{t,2} + S^T_{t+1:m}(b_t y_0 + y_t) \end{bmatrix} \in \mathbb{R}^{m-(\ell+1)},$$

so that, by (3.19), one may write

$$\phi_t(a_{t,2}) = L\begin{bmatrix} 0_{\ell+1} \\ \psi_{\ell+1,t} \end{bmatrix} \quad \text{for all} \quad t \in \{\ell+1, \ldots, m-1\}.$$

Our strategy is first to find $a^*_{\ell,2} \in \mathbb{R}^{m-\ell}$ satisfying (3.21a)–(3.21b) such that

$$a^*_{\ell,2} + S^T_{\ell+1:m}(b_\ell y_0 + y_\ell) \in \operatorname{span}\left\{\begin{bmatrix} 0 \\ \psi_{\ell+1,\ell+1} \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ \psi_{\ell+1,m-1} \end{bmatrix}\right\} \tag{3.22}$$

and (cf. (3.21c))

$$\phi_\ell(a^*_{\ell,2})^T \phi_\ell(a^*_{\ell,2}) \le z_\ell^T z_\ell. \tag{3.23}$$

Once this is done, we will argue the existence of a nonzero vector $\bar{a}_{\ell,2} \in \mathbb{R}^{m-\ell}$ such that $a^*_{\ell,2} + \lambda_\ell \bar{a}_{\ell,2}$ satisfies (3.21a)–(3.21b) for arbitrary one-dimensional λ_ℓ. From here, it will follow by the fact that the left-hand side of (3.21c) is a strongly convex quadratic in the unknown λ_ℓ and the fact that (3.23) holds that we can claim that there exists $\lambda^*_\ell \in \mathbb{R}$ such that $a_{\ell,2} = a^*_{\ell,2} + \lambda^*_\ell \bar{a}_{\ell,2} \in \mathbb{R}^{m-\ell}$ satisfies (3.21).

To achieve the goals of the previous paragraph, first let c be the column rank of $[\psi_{\ell+1,\ell+1} \ \cdots \ \psi_{\ell+1,m-1}]$ so that there exists $\{t_1, \ldots, t_c\} \subseteq \{\ell+1, \ldots, m-1\}$ with

$$\operatorname{span}\left\{\begin{bmatrix} 0 \\ \psi_{\ell+1,t_1} \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ \psi_{\ell+1,t_c} \end{bmatrix}\right\} = \operatorname{span}\left\{\begin{bmatrix} 0 \\ \psi_{\ell+1,\ell+1} \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ \psi_{\ell+1,m-1} \end{bmatrix}\right\}. \tag{3.24}$$

For completeness, let us first consider the extreme case when $c = 0$. In this case,

$$\psi_{\ell+1,t} = 0_{m-(\ell+1)} \ \text{ and } \ \phi_t(a_{t,2}) = 0_m \ \text{ for all } \ t \in \{\ell+1, \ldots, m-1\}. \tag{3.25}$$

Hence, by our induction hypothesis, it follows from (3.20) and (3.25) that

$$z_t = 0_{m-1} \ \text{ for all } \ t \in \{\ell+1, \ldots, m-1\}. \tag{3.26}$$

Consequently from (3.25) and (3.26), the affine equations (3.21a)–(3.21b) are satisfied by any $a^*_{\ell,2} \in \mathbb{R}^{m-\ell}$. In particular, one can choose

$$a^*_{\ell,2} = -S^T_{\ell+1:m}(b_\ell y_0 + y_\ell),$$

and find by (3.19) that

$$\phi_\ell(a^*_{\ell,2}) = L\left(\begin{bmatrix} 0_\ell \\ a^*_{\ell,2} + S^T_{\ell+1:m}(b_\ell y_0 + y_\ell) \end{bmatrix}\right) = 0_m, \tag{3.27}$$

which shows that this choice satisfies (3.23).

Now consider the case when $c > 0$. For $a^*_{\ell,2}$ to satisfy (3.22), it follows with (3.24) that we must have

$$a^*_{\ell,2} + S^T_{\ell+1:m}(b_\ell y_0 + y_\ell) = \begin{bmatrix} 0 & \cdots & 0 \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix} \beta_\ell, \tag{3.28}$$

where β_ℓ has length c. Choosing

$$\beta_\ell := \left(\begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix}^T L^T L \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix} \right)^{-1} \begin{bmatrix} z_{t_1}^T \\ \vdots \\ z_{t_c}^T \end{bmatrix} z_\ell \in \mathbb{R}^c, \tag{3.29}$$

it follows with (3.28) that, for any $t \in \{t_1, \ldots, t_c\}$, one finds

$$\begin{aligned} \phi_t(a_{t,2})^T \phi_\ell(a_{\ell,2}^*) &= \begin{bmatrix} 0_{\ell+1} \\ \psi_{\ell+1,t} \end{bmatrix}^T L^T L \begin{bmatrix} 0_\ell \\ a_{\ell,2}^* + S_{\ell+1:m}^T (b_\ell y_0 + y_\ell) \end{bmatrix} \\ &= \begin{bmatrix} 0_{\ell+1} \\ \psi_{\ell+1,t} \end{bmatrix}^T L^T L \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix} \beta_\ell = z_t^T z_\ell. \end{aligned} \tag{3.30}$$

We shall now prove that, for any $t \in \{\ell+1, \ldots, m-1\} \backslash \{t_1, \ldots, t_c\}$, one similarly finds that $\phi_t(a_{t,2})^T \phi_\ell(a_{\ell,2}^*) = z_t^T z_\ell$. Toward this end, first notice that for any such t it follows from (3.24) that $\psi_{\ell+1,t} = [\psi_{\ell+1,t_1} \ \cdots \ \psi_{\ell+1,t_c}] \gamma_{\ell,t}$ for some $\gamma_{\ell,t} \in \mathbb{R}^c$. Combining the relationship (3.24) along with the inductive hypothesis that, for any pair (i, j) with $i \in \{\ell+1, \ldots, m-1\}$ and $j \in \{i, \ldots, m-1\}$, one has

$$\begin{bmatrix} 0_{\ell+1} \\ \psi_{\ell+1,i} \end{bmatrix}^T L^T L \begin{bmatrix} 0_{\ell+1} \\ \psi_{\ell+1,j} \end{bmatrix} = \phi_i(a_{i,2})^T \phi_j(a_{j,2}) = z_i^T z_j, \tag{3.31}$$

it follows with the positive definiteness of $Q^{-1} = L^T L$ that

$$\begin{aligned} \operatorname{rank}\left(\begin{bmatrix} z_{\ell+1} & \cdots & z_{m-1} \end{bmatrix} \right) &= \operatorname{rank}\left(\begin{bmatrix} \phi_{\ell+1}(a_{\ell+1,2}) & \cdots & \phi_{m-1}(a_{m-1,2}) \end{bmatrix} \right) \\ &= \operatorname{rank}\left(\begin{bmatrix} \phi_{t_1}(a_{t_1,2}) & \cdots & \phi_{t_c}(a_{t_c,2}) \end{bmatrix} \right) \\ &= \operatorname{rank}\left(\begin{bmatrix} z_{t_1} & \cdots & z_{t_c} \end{bmatrix} \right) = c. \end{aligned} \tag{3.32}$$

From (3.32), it follows that for any $t \in \{\ell+1, \ldots, m-1\} \backslash \{t_1, \ldots, t_c\}$ there exists $\bar{\gamma}_{\ell,t} \in \mathbb{R}^c$ such that $z_t = [z_{t_1} \ \cdots \ z_{t_c}] \bar{\gamma}_{\ell,t}$. Combining the definitions of $\gamma_{\ell,t}$ and $\bar{\gamma}_{\ell,t}$ along with (3.31), it follows for any such t that

$$\begin{aligned} &\gamma_{\ell,t}^T \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix}^T L^T L \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix} \\ &\quad = \begin{bmatrix} 0_{\ell+1} \\ \psi_{\ell+1,t} \end{bmatrix}^T L^T L \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix} \\ &\quad = z_t^T \begin{bmatrix} z_{t_1} & \cdots & z_{t_c} \end{bmatrix} \\ &\quad = \bar{\gamma}_{\ell,t}^T \begin{bmatrix} z_{t_1} & \cdots & z_{t_c} \end{bmatrix}^T \begin{bmatrix} z_{t_1} & \cdots & z_{t_c} \end{bmatrix} \\ &\quad = \bar{\gamma}_{\ell,t}^T \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix}^T L^T L \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix}, \end{aligned}$$

from which it follows that $\gamma_{\ell,t} = \bar{\gamma}_{\ell,t}$. Thus, with (3.30) and the definitions of $\gamma_{\ell,t}$ and $\bar{\gamma}_{\ell,t}$, it follows for any $t \in \{\ell+1, \ldots, m-1\} \backslash \{t_1, \ldots, t_c\}$ that

$$\begin{aligned}
\phi_t(a_{t,2})^T \phi_\ell(a^*_{\ell,2}) &= \begin{bmatrix} 0_{\ell+1} \\ \psi_{\ell+1,t} \end{bmatrix}^T L^T L \begin{bmatrix} 0_\ell \\ a^*_{\ell,2} + S^T_{\ell+1:m}(b_\ell y_0 + y_\ell) \end{bmatrix} \\
&= \gamma_{\ell,t}^T \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix}^T L^T L \begin{bmatrix} 0_\ell \\ a^*_{\ell,2} + S^T_{\ell+1:m}(b_\ell y_0 + y_\ell) \end{bmatrix} \\
&= \gamma_{\ell,t}^T \begin{bmatrix} z_{t_1} & \cdots & z_{t_c} \end{bmatrix}^T z_\ell \\
&= \bar{\gamma}_{\ell,t}^T \begin{bmatrix} z_{t_1} & \cdots & z_{t_c} \end{bmatrix}^T z_\ell = z_t^T z_\ell,
\end{aligned}$$

Combining this with (3.30), it follows that $a^*_{\ell,2}$ from (3.28) with β_ℓ from (3.29) satisfies (3.21a)–(3.21b), as desired. Let us show now that this $a^*_{\ell,2}$ also satisfies (3.23). Indeed, by (3.28), (3.29), and (3.31), it follows that

$$\begin{aligned}
&\phi_\ell(a^*_{\ell,2})^T \phi_\ell(a^*_{\ell,2}) \\
&= \begin{bmatrix} 0_\ell \\ a^*_{\ell,2} + S^T_{\ell+1:m}(b_\ell y_0 + y_\ell) \end{bmatrix}^T L^T L \begin{bmatrix} 0_\ell \\ a^*_{\ell,2} + S^T_{\ell+1:m}(b_\ell y_0 + y_\ell) \end{bmatrix} \\
&= \beta_\ell^T \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix}^T L^T L \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix} \beta_\ell \\
&= z_\ell^T \begin{bmatrix} z_{t_1}^T \\ \vdots \\ z_{t_c}^T \end{bmatrix}^T \left(\begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix}^T L^T L \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix} \right)^{-1} \begin{bmatrix} z_{t_1}^T \\ \vdots \\ z_{t_c}^T \end{bmatrix} z_\ell \\
&= z_\ell^T \begin{bmatrix} z_{t_1}^T \\ \vdots \\ z_{t_c}^T \end{bmatrix}^T \left(\begin{bmatrix} z_{t_1}^T \\ \vdots \\ z_{t_c}^T \end{bmatrix} \begin{bmatrix} z_{t_1} & \cdots & z_{t_c} \end{bmatrix} \right)^{-1} \begin{bmatrix} z_{t_1}^T \\ \vdots \\ z_{t_c}^T \end{bmatrix} z_\ell \leq z_\ell^T z_\ell,
\end{aligned}$$

where the last inequality comes from the fact that the eigenvalues of

$$\begin{bmatrix} z_{t_1} & \cdots & z_{t_c} \end{bmatrix} \left(\begin{bmatrix} z_{t_1}^T \\ \vdots \\ z_{t_c}^T \end{bmatrix} \begin{bmatrix} z_{t_1} & \cdots & z_{t_c} \end{bmatrix} \right)^{-1} \begin{bmatrix} z_{t_1}^T \\ \vdots \\ z_{t_c}^T \end{bmatrix}$$

are all in $\{0, 1\}$. (As an aside, one finds that the inequality above is strict if $z_\ell \notin \operatorname{span}\{z_{t_1}, \ldots, z_{t_c}\}$.) Hence, we have shown that $a^*_{\ell,2}$ from (3.28) satisfies (3.23).

As previously mentioned (in the text following (3.23)), our goal now is to show that there exists a nonzero $\bar{a}_{\ell,2} \in \mathbb{R}^{m-\ell}$ such that $a^*_{\ell,2} + \lambda_\ell \bar{a}_{\ell,2}$ satisfies (3.21a)–(3.21b) for arbitrary λ_ℓ. From (3.21a)–(3.21b), such an $\bar{a}_{\ell,2} \in \mathbb{R}^{m-\ell}$ must satisfy

$$\begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,\ell+1} & \cdots & \psi_{\ell+1,m-1} \end{bmatrix}^T L^T L \begin{bmatrix} 0_\ell \\ \bar{a}_{\ell,2} \end{bmatrix} = 0_{m-(\ell+1)}. \tag{3.33}$$

Since

$$\begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,\ell+1} & \cdots & \psi_{\ell+1,m-1} \end{bmatrix}^T L^T L \in \mathbb{R}^{(m-(\ell+1))\times m}, \tag{3.34}$$

it follows that this matrix has a null space of dimension at least $\ell + 1$, i.e., there exist at least $\ell + 1$ linearly independent vectors in $\mathbb{R}^m$ belonging to the null space of this matrix. Let $N_{\ell+1} \in \mathbb{R}^{m\times(\ell+1)}$ be a matrix whose columns are $\ell + 1$ linearly independent vectors in $\mathbb{R}^m$ lying in the null space of (3.34). Since this null space matrix has $\ell + 1$ columns, there exists a nonzero vector $\zeta_{\ell+1} \in \mathbb{R}^{\ell+1}$ such that the first ℓ elements of $N_{\ell+1}\zeta_{\ell+1}$ are zero. Letting

$$[\bar{a}_{\ell,2}]_t := [N_{\ell+1}\zeta_{\ell+1}]_{\ell+t} \text{ for all } t \in \{1, \ldots, m-\ell\},$$

one finds that $\bar{a}_{\ell,2}$ satisfies (3.33), as desired. Consequently, as stated in the text following (3.23), it follows by the fact that the left-hand side of (3.21c) is a strongly convex quadratic in the unknown λ_ℓ and the fact that (3.23) holds that we can claim that there exists $\lambda_\ell^* \in \mathbb{R}$ such that $a_{\ell,2} = a_{\ell,2}^* + \lambda_\ell^* \bar{a}_{\ell,2} \in \mathbb{R}^{m-\ell}$ satisfies (3.21).

Combining all previous aspects of the proof, we have shown the existence of $A \in \mathbb{R}^{m\times(m-1)}$ and $b \in \mathbb{R}^{m-1}$ such that, with $\tilde{Y}_{1:m} \in \mathbb{R}^{n\times m}$ defined as in (3.7), the equations (3.9) hold. The remaining desired conclusions, namely, that (3.10) holds and that $\text{BFGS}(W, S_{0:m}, Y_{0:m}) = \text{BFGS}(W, S_{1:m}, \tilde{Y}_{1:m})$, follow from the existence of $\tilde{Y}_{1:m} \in \mathbb{R}^{n\times m}$ (that we have proved), the fact that the equations in (3.10) are a subset of the equations in (3.12a), and the fact that (3.9) was derived explicitly to ensure that, with $\tilde{Y}_{1:m}$ satisfying (3.7), one would find that (3.6) holds. □

3.4 Implementing Agg-BFGS

We now discuss how one may implement our Agg-BFGS scheme to iteratively aggregate displacement information in the context of an optimization algorithm employing BFGS approximations. We also discuss the dominant computational costs of applying the scheme, and comment on certain numerical considerations that one should take into account in a software implementation. The procedures presented in this section are guided by our proof of Theorem 3.2.

As will become clear in our overall approach, in contrast to a traditional limited-memory scheme in which one always maintains the most recent curvature pairs to define a Hessian approximation, the pairs used in our approach might come from a subset of the previous iterations, with the gradient displacements potentially having been modified through our aggregation mechanism. For concreteness, suppose that during the course of the run of an optimization algorithm for solving (1.1), one has accumulated a set of curvature pairs, stored in the sets

$$\mathcal{S} := \{s_{k_0}, \ldots, s_{k_{m-1}}\} \text{ and } \mathcal{Y} := \{y_{k_0}, \ldots, y_{k_{m-1}}\}$$

where $\{k_i\}_{i=0}^{m-1} \subset \mathbb{N}$ with $k_i < k_{i+1}$ for all $i \in \{0, \ldots, m-2\}$, such that the vectors in the former set (i.e., the iterate displacements) are linearly independent. (Here, the

elements of $\mathcal{Y}$ are not necessarily the gradient displacements computed in iterations $\{k_0, \ldots, k_{m-1}\}$, but, for simplicity of notation, we denote them as $\mathcal{Y}$ even though they might have been modified during a previous application of our aggregation scheme.) Then, suppose that a new curvature pair (s_{k_m}, y_{k_m}) for $k_m \in \mathbb{N}$ with $k_{m-1} < k_m$ is available. Our goal in this section is to show how one may add and, if needed, aggregate the information in these pairs in order to form new sets

$$\tilde{\mathcal{S}} \subseteq \mathcal{S} \cup \{s_{k_m}\} \text{ and } \tilde{\mathcal{Y}}$$

such that

(i) the sets $\tilde{\mathcal{S}}$ and $\tilde{\mathcal{Y}}$ have the same cardinality, which is either m or $m+1$,
(ii) the vectors in the set $\tilde{\mathcal{S}}$ are linearly independent, and
(iii) the BFGS inverse Hessian approximation generated from the data in $(\mathcal{S} \cup \{s_{k_m}\}, \mathcal{Y} \cup \{y_{k_m}\})$ is the same as the approximation generated from $(\tilde{\mathcal{S}}, \tilde{\mathcal{Y}})$.

As in the previous section, we henceforth simplify the subscript notation and refer to the "previously stored" displacement vectors as those in the sets $\{s_0, \ldots, s_{m-1}\}$ and $\{y_0, \ldots, y_{m-1}\}$, and refer to the "newly computed" curvature pair as (s_m, y_m).

Once the newly computed pair (s_m, y_m) is available, there are three cases.

Case 1. The set of vectors $\{s_0, s_1, \ldots, s_m\}$ is linearly independent. In this case, one simply adds the new curvature pair, which leads to the $(m+1)$-element sets

$$\tilde{\mathcal{S}} = \{s_0, \ldots, s_{m-1}, s_m\} \text{ and } \tilde{\mathcal{Y}} = \{y_0, \ldots, y_{m-1}, y_m\}.$$

Notice that if $m = n$, then this case is not possible.

Case 2. The new iterate displacement vector s_m is parallel to the most recently stored iterate displacement vector, i.e., $s_{m-1} = \tau s_m$ for some $\tau \in \mathbb{R}$. In this case, one should discard the most recently stored pair and replace it with the newly computed one, which leads to the m-element sets

$$\tilde{\mathcal{S}} = \{s_0, \ldots, s_{m-2}, s_m\} \text{ and } \tilde{\mathcal{Y}} = \{y_0, \ldots, y_{m-2}, y_m\}.$$

This choice is justified by Theorem 3.1.

Case 3. For some $j \in \{0, \ldots, m-2\}$, an iterate displacement vector s_j lies in the span of the subsequent iterate displacements vectors, i.e., $s_j \in \operatorname{span}\{s_{j+1}, \ldots, s_m\}$. In this case, one should apply our aggregation scheme to determine the vectors $\{\tilde{y}_{j+1}, \ldots, \tilde{y}_m\}$, then remove the pair (s_j, y_j), leading to the m-element sets

$$\tilde{\mathcal{S}} = \{s_0, \ldots, s_{j-1}, s_{j+1}, \ldots, s_m\} \text{ and } \tilde{\mathcal{Y}} = \{y_0, \ldots, y_{j-1}, \tilde{y}_{j+1}, \ldots, \tilde{y}_m\}.$$

This choice is justified by Theorem 3.2.

Computationally, the first step is to determine which of the three cases occurs. One way to do this efficiently is to maintain a Cholesky factorization of an inner product matrix corresponding to the previously stored iterate displacement vectors,

then attempt to add to it a new row/column corresponding to the newly computed iterate displacement, checking whether the procedure breaks down. Specifically, before considering the newly computed pair (s_m, y_m), suppose that one has a lower triangular matrix $\Theta \in \mathbb{R}^{m\times m}$ with positive diagonal elements such that

$$\begin{bmatrix} s_{m-1} & \cdots & s_0 \end{bmatrix}^T \begin{bmatrix} s_{m-1} & \cdots & s_0 \end{bmatrix} = \Theta\Theta^T, \tag{3.35}$$

which exists due to the fact that the vectors in $\{s_0, \ldots, s_{m-1}\}$ are linearly independent. A Cholesky factorization of an augmented inner product matrix would consist of a scalar $\mu \in \mathbb{R}_{>0}$, vector $\delta \in \mathbb{R}^m$, and lower triangular $\Delta \in \mathbb{R}^{m\times m}$ with

$$\begin{bmatrix} s_m & s_{m-1} & \cdots & s_0 \end{bmatrix}^T \begin{bmatrix} s_m & s_{m-1} & \cdots & s_0 \end{bmatrix} = \begin{bmatrix} \mu & 0 \\ \delta & \Delta \end{bmatrix} \begin{bmatrix} \mu & \delta^T \\ 0 & \Delta^T \end{bmatrix}. \tag{3.36}$$

As is well known, equating terms in (3.35) and (3.36) one must have $\mu = \|s_m\|$, $\delta^T = [s_m^T s_{m-1} \ \cdots \ s_m^T s_0]/\mu$, and $\Delta\Delta^T = \Theta\Theta^T - \delta\delta^T$, meaning that Δ can be obtained from Θ through a *rank-one downdate*; e.g., see [19]. If this downdate does not break down–meaning that all diagonal elements of Δ are computed to be positive–then one is in Case 1 and the newly updated Cholesky factorization has been made available when yet another curvature pair is considered (after a subsequent optimization algorithm iteration). Otherwise, if the downdate does break down, then it is due to a computed diagonal element being equal to zero. This means that, for some smallest $i \in \{1, \ldots, m\}$, one has found a lower triangular matrix $\Xi \in \mathbb{R}^{i\times i}$ with positive diagonal elements and a vector $\xi \in \mathbb{R}^i$ such that

$$\begin{bmatrix} s_m & s_{m-1} & \cdots & s_{m-i} \end{bmatrix}^T \begin{bmatrix} s_m & s_{m-1} & \cdots & s_{m-i} \end{bmatrix} = \begin{bmatrix} \Xi & 0 \\ \xi^T & 0 \end{bmatrix} \begin{bmatrix} \Xi^T & \xi \\ 0 & 0 \end{bmatrix}. \tag{3.37}$$

Letting $\tau \in \mathbb{R}^i$ be the unique vector satisfying $\Xi^T\tau = \xi$, one finds that the vector $[\tau^T, -1]^T$ lies in the null space of (3.37), from which it follows that

$$\begin{bmatrix} s_m & s_{m-1} & \cdots & s_{m-i+1} \end{bmatrix} \tau = s_{m-i},$$

where the first element of τ must be nonzero since $\{s_{m-1}, \ldots, s_{m-i}\}$ is a set of linearly independent vectors. If the breakdown occurs for $i = 1$, then one is in Case 2. If the breakdown occurs for $i > 1$, then one is in Case 3 with the vector τ that one needs to apply our aggregation scheme to remove the pair (s_{m-i}, y_{m-i}).

Notice that if the breakdown occurs in the rank-one downdate as described in the previous paragraph, then one can continue with standard Cholesky factorization updating procedures in order to have the factorization of

$$\begin{bmatrix} s_m & \cdots & s_{m-i+1} & s_{m-i-1} & \cdots & s_0 \end{bmatrix}^T \begin{bmatrix} s_m & \cdots & s_{m-i+1} & s_{m-i-1} & \cdots & s_0 \end{bmatrix}$$

available in subsequent iterations. For brevity and since it is outside of our main scope, we do not discuss this in detail. Overall, the computational costs so far are $\mathcal{O}(mn)$

(for computing the inner products $\{s_m^T s_{m-1}, \ldots, s_m^T s_0\}$) plus $\mathcal{O}(m^2)$ (for updating the Cholesky factorization and, in Case 2 or Case 3, computing τ).

If Case 1 occurs, then no additional computation is necessary; one simply adds the new curvature pair as previously described. Similarly, if Case 2 occurs, then again no additional computation is necessary; one simply replaces the most recent stored pair with the newly computed one. Therefore, we may assume for the remainder of this section that Case 3 occurs, we have identified an index j ($= m - i$ using the notation above) such that $s_j \in \text{span}\{s_{j+1}, \ldots, s_m\}$, and we have $\tau \in \mathbb{R}^{m-j}$ such that $s_j = S_{j+1:m}\tau$. Our goal then is to apply our aggregation scheme to modify the gradient displacement vectors $\{y_{j+1}, \ldots, y_m\}$ to compute

$$\tilde{Y}_{j+1:m} = W_{0:j-1}^{-1} S_{j+1:m} \begin{bmatrix} A & 0 \end{bmatrix} + y_j \begin{bmatrix} b \\ 0 \end{bmatrix}^T + Y_{j+1:m}, \tag{3.38}$$

where $W_{0:j-1}$ represents the BFGS inverse Hessian approximation defined by some initial positive definite matrix $W \succ 0$ and the curvature pairs $\{s_i, y_i\}_{i=0}^{j-1}$, and where $A \in \mathbb{R}^{(m-j)\times(m-j-1)}$ and $b \in \mathbb{R}^{m-j-1}$ are the unknowns to be determined.

For simplicity in the remainder of this section, let us suppose that $j = 0$ so that the pair (s_0, y_0) is to be removed as in the notation of Sect. 3.2 and Sect. 3.3. As one might expect, this is the value of j for which the computational costs of computing $A \in \mathbb{R}^{m\times(m-1)}$ and $b \in \mathbb{R}^{m-1}$ are the highest. For all other values of j, there is a cost for computing $W_{0:j-1}^{-1} S_{j+1:m}$ as needed in (3.38). This matrix can be computed *without* forming the BFGS Hessian approximation $W_{0:j-1}^{-1}$; it can be constructed, say, by computing matrix-vector products with a compact representation of this approximation for a total cost of $\mathcal{O}(j(m-j)n) \leq \mathcal{O}(m^2 n)$; see Sect. 7.2 in [31].

Let us now describe how one may implement our aggregation scheme such that, given $S_{1:m}$ with full column rank, $Y_{1:m}$, $\rho_{1:m} > 0$, $\tau \in \mathbb{R}^m$ satisfying $s_0 = S_{1:m}\tau$, y_0, and $\rho_0 > 0$, one may compute $A \in \mathbb{R}^{m\times(m-1)}$ and $b \in \mathbb{R}^{m-1}$ in order to obtain $\tilde{Y}_{1:m}$ as in (3.7). By Theorem 3.2, it follows that real values for A and b exist to satisfy (3.9). The computation of the vector b is straightforward; it can be computed by the formula (3.12b). Assuming that the products in $S_{1:m}^T Y_{1:m-1}$ have already been computed (which, using previously computed inner products and recomputing any as needed if/when gradient displacements have been modified by aggregation, costs $\mathcal{O}((m-j)^2 n)$), the cost of computing b is $\mathcal{O}(m^2)$.

For computing A, let us first establish some notation since the elements of this matrix will be computed with a specific order. As in the proof of Theorem 3.2, let $A = [a_1 \ \cdots \ a_{m-1}]$ where $a_\ell \in \mathbb{R}^m$ for all $\ell \in \{1, \ldots, m-1\}$, and, as in (3.14), let

$$a_\ell = Q^{-1} \begin{bmatrix} a_{\ell,1} \\ a_{\ell,2} \end{bmatrix}, \text{ where } a_{\ell,1} \in \mathbb{R}^\ell, \ a_{\ell,2} \in \mathbb{R}^{m-\ell}, \text{ and } Q := S_{1:m}^T W^{-1} S_{1:m} \succ 0.$$

Here and going forward, our computations require products with Q^{-1}. Rather than form this matrix explicitly, one may maintain a Cholesky factorization of Q and add/delete rows/columns–as described previously for the inner product matrix corresponding to the iterate displacements–as the iterate displacement set is updated

throughout the run of the (outer) optimization algorithm. With such a factorization, products with Q^{-1} are obtained by triangular solves in a standard fashion. If $W \succ 0$ is diagonal or defined by a limited memory approximation, then the cost of updating this factorization in each instance is $\mathcal{O}(mn)$ (for computing $W^{-1}s_m$) plus $\mathcal{O}(m^2)$ for updating the factorization. Each backsolve costs $\mathcal{O}(m^2)$.

An approach for computing A is now summarized as Algorithm 4. As explained in the proof of Theorem 3.2, the values of $\{a_{\ell,1}\}_{\ell=1}^{m-1}$ are set to ensure that (3.12a) is satisfied. Assuming that $S_{1:m}^T y_0$ has been computed at cost $\mathcal{O}(mn)$, the cost of computing $\{a_{\ell,1}\}_{\ell=1}^{m-1}$ is $\mathcal{O}(m^2)$. The values for $\{a_{\ell,2}\}_{\ell=1}^{m-1}$ are then computed in reverse order to solve the system of linear and quadratic equations comprising (3.12c). Assuming (as has been mentioned) that a factorization of Q is available, and assuming that the elements of the right-hand side of (3.18) have been pre-computed (at cost $\mathcal{O}(m^3)$), the cost of computing $a_{m-1,2}$ is $\mathcal{O}(1)$. As for computing the remaining vectors, the presented scheme follows the proof of Theorem 3.2. The most expensive operation in each iteration of this scheme is the QR factorization of the matrix in (3.34), which for each ℓ is of size $(m - (\ell + 1)) \times m$. Summing the cost of these for $\ell = m - 2$ to $\ell = 1$, the total cost is found to be $\mathcal{O}(m^4)$.

Algorithm 4 : Displacement Aggregation, Computation of A

1: For each $\ell \in \{1, \ldots, m-1\}$, compute the ℓ-element vector $a_{\ell,1}$ by (3.15).
2: Compute $a_{m-1,2}$ by solving the quadratic equation (3.21c).
3: **for** $\ell = m-2, \ldots, 1$ **do**
4: Compute β_ℓ by (3.29).
5: Compute $a^*_{\ell,2}$ by (3.28).
6: Compute nonzero vector $\bar{a}_{\ell,2}$ to satisfy (3.33).
7: Compute $\lambda^*_\ell \in \mathbb{R}$ such that $a_{\ell,2} = a^*_{\ell,2} + \lambda^*_\ell \bar{a}_{\ell,2}$ solves the quadratic equation (3.21c).
8: **end for**
9: **return** $A = \begin{bmatrix} a_1 & \cdots & a_{m-1} \end{bmatrix}$ with $\{a_j\}_{j=1}^{m-1}$ defined by (3.14).

This completes our description of a manner in which our `Agg-BFGS` scheme can be implemented. Observing the computational costs that have been cited, one finds that a conservative estimate of the total cost is $\mathcal{O}(m^2 n) + \mathcal{O}(m^4)$. Upon closer inspection, one finds that for $j \approx 0$ the cost is dominated by the cost of computing/updating $S_{1:m}^T \tilde{Y}_{1:m}$ *after* aggregation has been performed, whereas for $j \approx m$ the cost is dominated by the cost of computing $W_{0:j-1}^{-1} S_{j+1:m}$ *before* aggregation is performed. In either case, if $m \ll n$ (say with $3 \leq m \leq 10$, as is typical in practice), then the overall cost is not too dissimilar from $\mathcal{O}(4mn)$ per iteration, which is the cost of other limited-memory schemes such as L-BFGS [8].

We end this section by stating the following result, for closure.

Theorem 3.3 *If one applies* `Agg-BFGS` *as described in this section, then one need only store at most $m \leq n$ curvature pairs such that the corresponding BFGS inverse Hessian approximations are equivalent to those in a full-memory BFGS scheme. Consequently, under the same conditions as in Theorem 2.1, the resulting optimization algorithm produces $\{x_k\}$ that converges to x_* at a superlinear rate.*

4 Numerical accuracy of aggregation

Our goals in this and the next section are to provide additional numerical demonstrations of applications of `Agg-BFGS`. (Recall that a preview demonstration has been provided in Fig. 1a.) Our goal in this section is to show empirically that limited-memory-type BFGS inverse Hessian approximations provided by `Agg-BFGS` accurately represent the approximations provided by full-memory BFGS. We show that while numerical errors might accumulate to some degree as `Agg-BFGS` is applied over a sequence of iterations, the inverse Hessian approximations provided by `Agg-BFGS` are not necessarily poor after multiple iterations.

We implemented `Agg-BFGS` in MATLAB and ran two sets of experiments. First, for $(n, m) \in \{4, 8, 16, 32, 64, 128\}^2$ with $m \leq n$, we generated data to show the error of applying `Agg-BFGS` to aggregate a single curvature pair. For each (n, m), we generated 100 datasets using the following randomized procedure. First, MATLAB's built-in `sprandsym` function was used to generate a random positive definite matrix with condition number approximately 10^4. This matrix defined a quadratic function. Second, a random fixed point was determined using MATLAB's built-in `randn` routine. From this point, a mock optimization procedure for minimizing the generated quadratic was run to generate $\{(s_k, y_k)\}_{k=1}^m$; in particular, for $k \in \{1, \ldots, m\}$, starting with the fixed point, a descent direction was chosen as the negative gradient plus noise (specifically, the norm of the gradient divided by 10 times a random vector generated with `randn`) and a stepsize was chosen by an exact line search to compute the subsequent iterate and gradient displacement pair. Third, a vector $\tau \in \mathbb{R}^m$ was generated randomly using `randn` in order to define $s_0 = [s_1 \ \cdots \ s_m]\tau$. The corresponding gradient displacement y_0 was determined by stepping *backward* from the fixed point along s_0. In this manner, we obtained $\{(s_k, y_k)\}_{k=0}^m$ from a mock optimization procedure from some initial point in such a way that s_0 was guaranteed to lie the span of the subsequent iterate displacements.

For each dataset starting from $W_0 = I$, we computed the BFGS inverse Hessian approximation from $\{(s_k, y_k)\}_{k=0}^m$ and constructed the inverse Hessian approximation corresponding to the set of pairs when `Agg-BFGS` was used to aggregate the information into $\{(s_k, \tilde{y}_k)\}_{k=1}^m$. Fig. 2 shows box plots for relative errors between each pair of inverse Hessian approximations, where error is measured in terms of the maximum component-wise absolute difference between matrix entries divided by the largest absolute value of an element of the BFGS inverse Hessian approximation. The results show that while the errors are larger as n increases and as m is closer to n, they remain accurate relative to machine precision for all (n, m).

As a second experiment, for $(n, m) \in \{(8, 8), (32, 32), (128, 128)\}$, we aimed to investigate how errors might accumulate as `Agg-BFGS` is applied over a sequence of iterations. For these experiments, we generated data using a similar mock optimization procedure as in the previous experiment. Specifically, from some randomly generated starting point, we computed a random step as in the aforementioned procedure, continuing until $n+8$ iterations were performed. Fig. 3 shows the errors for each iteration beyond $k = n$. As in Fig. 1a, these results show that the errors do not accumulate too poorly as k increases. We conjecture that part of the reason for this is that errors that result from each application of `Agg-BFGS` can be over-written eventually, at least

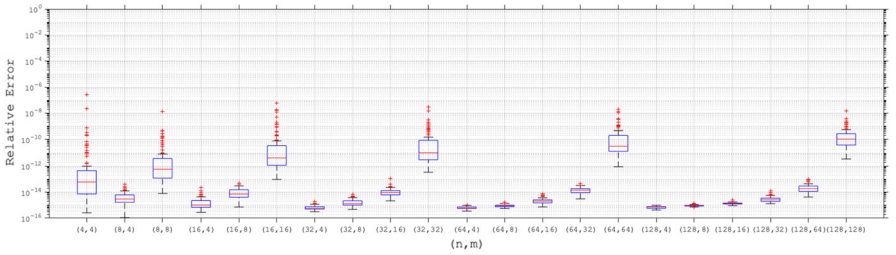
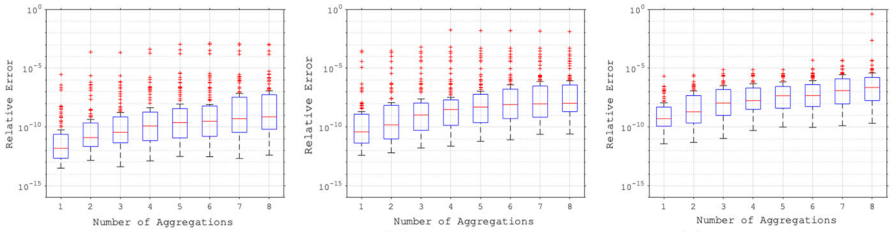


Fig. 2 Relative errors of the differences between BFGS inverse Hessian approximations (computed from $\{(s_k, y_k)\}_{k=0}^{m}$) and the corresponding inverse Hessian approximations represented after applying `Agg-BFGS` to aggregate the information from a single curvature pair (into $\{(s_k, \tilde{y}_k)\}_{k=1}^{m}$). For each (n, m), the box plots show the results for 100 randomly generated instances. Relative error is the maximum absolute difference between corresponding matrix entries divided by the largest absolute value of an element of the BFGS inverse Hessian approximation



(a) $n = m = 8$ **(b)** $n = m = 32$ **(c)** $n = m = 128$

Fig. 3 Accumulation of relative errors of the differences between BFGS inverse Hessian approximations and the corresponding inverse Hessian approximations after applying `Agg-BFGS` to aggregate the information. For each (n, m), the box plots show the results for 100 randomly generated instances. Relative error is the maximum absolute difference between corresponding matrix entries divided by the largest absolute value of an element of the BFGS inverse Hessian approximation

to some extent, similar to the manner in which curvature information is ultimately over-written in full-memory BFGS.

5 A practical adaptive L-BFGS method using `Agg-BFGS`

Our goal now is to provide the results of numerical experiments with an adaptive L-BFGS method that uses our `Agg-BFGS` scheme and show that over a diverse test set it can outperform a standard L-BFGS approach. These experiments are run in the practical regime when the number of pairs used is small relative to n.

To demonstrate the use of `Agg-BFGS` in a minimization algorithm, we compare the results of three algorithms using quasi-Newton inverse Hessian approximations for computing the search directions, where for each algorithm the same weak Wolfe line search was used for computing stepsizes. The first algorithm employed L-BFGS approximations. The second algorithm also employed L-BFGS approximations, but with `Agg-BFGS` used to aggregate information when deemed appropriate in the manner described below. Since this second algorithm does not always aggregate the oldest

pair, but rather uses a particular scheme for choosing which historical pair to use in the aggregation approach (as explained later on), we also considered a third algorithm. This algorithm uses the same scheme as the second to determine which pair to consider, but simply removes this pair rather than perform aggregation with it. Consequently, the comparison between the second and third algorithms shows the effect of aggregation itself, rather than only the difference between removing the oldest versus some other historical pair in an L-BFGS scheme.

We performed experiments with problems from the CUTEst collection [21]. We chose all problems from the collection for which the number of variables could be chosen in the interval $[10, 3000]$. For most problems, $n \gg m$, meaning that the costs of performing aggregation was negligible compared to the costs of computing search directions, which were the same for all algorithms using the standard two-loop recursion for L-BFGS. (See Tables 1 and 2 for n for all test problems used.) All algorithms were run until an iteration, call it $k \in \mathbb{N}$, was reached with $\|g_k\|_\infty \leq 10^{-6} \max\{1, \|g_0\|_\infty\}$, or until an iteration limit of 10^5 was surpassed.

For the second algorithm, `Agg-BFGS` was employed in the following manner. Suppose in iteration $k \in \mathbb{N}$ that the algorithm had in storage the displacement pairs $\{(s_{k_j}, \tilde{y}_{k_j})\}_{j=1}^{\overline{m}}$ for some indices $\{k_1, \ldots, k_{\overline{m}}\} \subseteq \{1, \ldots, k-1\}$ with $\overline{m} \leq m$ and the new pair (s_k, y_k) has been computed. Using techniques described in Sect. 3.4, starting with $j = \overline{m}$ and iterating down to $j = 1$, we determined if s_{k_j} lay approximately in $\text{span}\{s_{k_{j+1}}, \ldots, s_{k_{\overline{m}}}, s_k\}$ (see next paragraph). If so, then we applied `Agg-BFGS` to aggregate the information in the pair with index k_j; otherwise, if no such iterate displacement was determined, then either the pair (s_k, y_k) was simply added to the set of pairs in storage (if $\overline{m} < m$) or the pair with index k_1 was dropped (if $\overline{m} = m$, as in standard L-BFGS). In this manner, the number of pairs stored and employed remained less than or equal to m, as usual for L-BFGS.

To determine if s_{k_j} lay approximately in $\text{span}\{s_{k_{j+1}}, \ldots, s_{k_{\overline{m}}}, s_k\}$, we first computed the orthogonal projection of s_{k_j} onto this subspace, call it $\hat{s}_{k_j}$. Computing this projection is inexpensive given a Cholesky factorization of the inner product matrix $[s_{k_{j+1}} \ \cdots \ s_{k_{\overline{m}}} \ s_k]^T [s_{k_{j+1}} \ \cdots \ s_{k_{\overline{m}}} \ s_k]$, which can be updated with only $\mathcal{O}(mn)$ cost in each outer iteration as described in Sect. 3.4. If the condition

$$\|s_{k_j} - \hat{s}_{k_j}\|_2 \leq \texttt{tol} \cdot \|\hat{s}_{k_j}\|_2 \tag{5.1}$$

held for some $\texttt{tol} \in (0, 1)$, then it was determined that s_{k_j} lay approximately in the subspace, since in this case the norm of the projection $\hat{s}_{k_j}$ was sufficiently large compared to the orthogonal component $s_{k_j} - \hat{s}_{k_j}$. For $j = \{\overline{m}, \ldots, 2\}$, we used $\texttt{tol} = 10^{-8}$, while for $j = 1$ we loosened the tolerance to $\texttt{tol} = 10^{-4}$ to promote aggregation of the oldest pair. If (5.1) held for some index k_j, then aggregation was performed with the pair $(\hat{s}_{k_j}, \tilde{y}_{k_j})$. Notice that this ensures that the aggregation is performed with an iterate displacement vector (namely, the projected vector $\hat{s}_{k_j}$) that lies in the subspace defined by subsequent displacements, as required in our algorithms and theoretical results in Sect. 3. (We remark in passing that, using this scheme, it is possible that aggregation could be triggered multiple times in a single iteration. Our software allows for this, but it did not occur in our experiments.)

One might expect that, under these conditions, Agg-BFGS might not perform aggregation often. However, on the contrary, our results show that aggregation was performed quite often.[3] Tables 1 and 2 show detailed results for the first two algorithms in our experiments for $m = 5$. (We do not include the third algorithm since, as shown below, it was inferior to the first and second algorithms.) For conciseness, we refer to the algorithm that employed a standard LBFGS(5) strategy as LBFGS(5), and we refer to the strategy that employed Agg-BFGS as AggBFGS(5). For AggBFGS(5), the table reports the number of aggregations performed, which in many cases was significant compared to the number of iterations.

One finds in our results that, generally speaking, AggBFGS(m) outperforms LBFGS(m). By contrast, the third algorithm in our experiments, which we refer to as LBFGS-Alt(m), if anything often performs worse than even LBFGS(m). This can be seen more clearly in performance profiles. In Figs. 4 and 5, we present the results in the form of two types of profiles: Dolan and Moré performance profiles [17] and Morales outperforming factor profiles [29]. The former type of profile has a graph for each algorithm, where if a graph for an algorithm passes through the point $(\alpha, 0.\beta)$, where β is a two-digit integer, then on $\beta\%$ of the problems the measure required by the algorithm was less than 2^{α} times the measure required by the best algorithm in terms of the measure. In this manner, an algorithm has performed better than the other if its graph is above and to the left of the graph of the other algorithm. The second type of profile shows a bar plot of logarithmic outperforming factors, with a bar for each problem in the test set. In our case, ignoring the performance of the third algorithm (LBFGS-Alt(m)), the bar for each problem takes on the value $-\log_2(meas_{\text{AggBFGS}(m)}/meas_{\text{LBFGS}(m)})$, where $meas_{\text{AggBFGS}(m)}$ is the performance measure for AggBFGS(m) and $meas_{\text{LBFGS}(m)}$ is the performance measure for LBFGS(m). An upward-pointing bar shows by how much AggBFGS(m) outperformed LBFGS(m) on a particular problem, and vice versa for the downward-pointing bars. We sort the bars by value for ease of visualization. Figure 4 shows profiles using the number of iterations as the performance measure, and Fig. 5 shows profiles using the number of function evaluations required as the performance measure. One may conclude from the profiles that Agg-BFGS helps performance more often than not, with the benefits being even more significant for smaller values of m.

Much remains to be investigated in terms of the practical use of Agg-BFGS, including perhaps more sophisticated techniques for implementing the scheme, handling numerical errors, and tuning parameters so that the results may be even better on a wider variety of problems. Our preliminary experiments in this section motivate such further investigations into the practical use of Agg-BFGS.

6 Conclusion

We have presented a technique for aggregating the curvature pair information in a limited-memory-type BFGS approach such that the corresponding Hessian (and

[3] This provides evidence for the belief, held by some optimization researchers, that when solving certain large-scale problems one often observes that consecutive steps lie approximately in low-dimensional subspaces.

Table 1 Numbers of iterations, function evaluations, and aggregations when algorithms are applied to solve problems from the CUTEst set with $n \in [10, 3000]$

Name	n	AggBFGS(5)			LBFGS(5)	
		Iters.	Funcs.	Aggs.	Iters.	Funcs.
ARGLINA	200	3	12	1	3	12
ARGLINB	200	3	3	1	3	3
ARGLINC	200	3	3	1	3	3
ARGTRIGLS	200	1494	10220	0	1494	10220
ARWHEAD	1000	2	2	0	2	2
BA-L1LS	57	60	218	0	60	218
BA-L1SPLS	57	62	248	0	62	248
BDQRTIC	1000	160	734	28	399	2051
BOX	1000	16	76	6	–	–
BOXPOWER	1000	20	86	11	12	29
BROWNAL	200	3	4	0	3	4
BROYDN3DLS	1000	39	45	0	39	45
BROYDN7D	1000	1444	2996	0	1444	2996
BROYDNBDLS	1000	71	151	0	71	151
BRYBND	1000	71	151	0	71	151
CHAINWOO	1000	563	2791	11	506	2577
CHNROSNB	50	186	355	0	186	355
CHNRSNBM	50	533	1593	0	533	1593
COSINE	1000	112	467	0	112	467
CRAGGLVY	1000	247	1189	10	241	1153
CURLY10	1000	–	–	–	–	–
CURLY20	1000	–	–	–	–	–
CURLY30	1000	–	–	–	–	–
DIXMAANA	300	16	48	10	20	60
DIXMAANB	300	22	78	6	24	97
DIXMAANC	300	20	106	5	18	78
DIXMAAND	300	25	93	6	35	135
DIXMAANE	300	400	1865	5	378	1798
DIXMAANF	300	383	1906	6	332	1728
DIXMAANG	300	264	1428	8	319	1526
DIXMAANH	300	425	2148	34	526	2668
DIXMAANI	300	1763	9092	33	2054	10878
DIXMAANJ	300	307	1380	6	406	2006
DIXMAANK	300	333	1589	15	331	1618
DIXMAANL	300	302	1553	29	363	1860
DIXMAANM	300	2901	14233	22	5220	6061

Table 1 continued

Name	n	AggBFGS(5)			LBFGS(5)	
		Iters.	Funcs.	Aggs.	Iters.	Funcs.
DIXMAANN	300	701	3749	15	1623	8508
DIXMAANO	300	694	3423	48	720	3642
DIXMAANP	300	555	2662	52	677	3421
DIXON3DQ	1000	5247	5978	0	5247	5978
DMN15102LS	66	206	962	44	1835	13002
DMN15103LS	99	853	4535	111	1696	9250
DMN15332LS	66	1253	6574	194	978	5241
DMN15333LS	99	11244	39107	26	4339	16755
DMN37142LS	66	639	3314	105	710	3744
DMN37143LS	99	12	128	0	12	128
DQDRTIC	1000	40	191	21	23	75
DQRTIC	1000	311	1838	6	511	2973
EDENSCH	36	110	547	11	138	673
EG2	1000	5	5	3	5	5
EIGENALS	110	661	1898	0	661	1898
EIGENBLS	110	1239	3765	0	1239	3765
EIGENCLS	462	1898	5431	0	1898	5431
ENGVAL1	1000	50	74	1	50	74
ERRINROS	50	900	4597	92	3648	16633
ERRINRSM	50	662	3225	60	–	–
EXTROSNB	1000	301	1231	1	307	1350
FLETBV3M	1000	92	451	3	114	518
FLETCBV2	1000	1014	1022	0	1014	1022
FLETCBV3	1000	–	–	–	–	–
FLETCHBV	1000	–	–	–	–	–
FLETCHCR	1000	176	1625	0	176	1625
FMINSRF2	961	374	429	0	374	429
FMINSURF	961	245	309	0	245	309
FREUROTH	1000	23	64	0	23	64
GENHUMPS	1000	3368	9766	0	3368	9766
GENROSE	500	2299	13271	0	2299	13271
HILBERTA	10	31	145	9	55	240

inverse Hessian) approximations are the same as those that would be computed in a full-memory BFGS approach. The key idea is that if one finds that a stored iterate displacement vector lies in the span of subsequent iterate displacements, then the gradient displacement vectors can be modified in such a way that the pair involving the linearly dependent iterate displacement can be removed with no information being lost. To the best of our knowledge, this is the first limited-memory-type approach

Table 2 Numbers of iterations, function evaluations, and aggregations when algorithms are applied to solve problems from the CUTEst set with $n \in [10, 3000]$

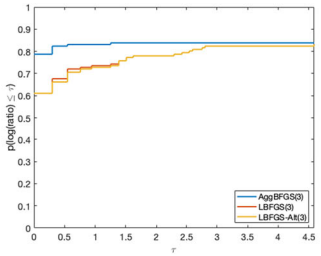
Name	n	AggBFGS(5)			LBFGS(5)	
		Iters.	Funcs.	Aggs.	Iters.	Funcs.
HILBERTB	50	14	38	6	14	38
HYDC20LS	99	1802	4128	0	1802	4128
INDEF	1000	–	–	–	–	–
INDEFM	1000	–	–	–	–	–
INTEQNELS	102	12	13	0	12	13
JIMACK	81	1584	3833	0	1584	3833
LIARWHD	1000	21	96	18	19	78
LUKSAN11LS	100	1332	5926	0	1332	5926
LUKSAN12LS	98	290	639	0	290	639
LUKSAN13LS	98	73	147	0	73	147
LUKSAN14LS	98	224	1041	0	224	1041
LUKSAN15LS	100	40	116	0	40	116
LUKSAN16LS	100	44	88	0	44	88
LUKSAN17LS	100	402	1917	0	402	1917
LUKSAN21LS	100	702	1964	0	702	1964
LUKSAN22LS	100	224	852	0	224	852
MANCINO	100	44	182	11	58	257
MNISTS0LS	494	2	2	0	2	2
MNISTS5LS	494	2	2	0	2	2
MODBEALE	200	–	–	–	–	–
MOREBV	1000	333	1211	0	333	1211
MSQRTALS	529	3855	7337	0	3855	7337
MSQRTBLS	529	2671	5099	0	2671	5099
NCB20	1010	929	4353	0	929	4353
NCB20B	1000	–	–	–	–	–
NONCVXU2	1000	4819	24493	714	5246	27413
NONCVXUN	1000	4014	20743	516	5863	30231
NONDIA	1000	5	5	2	5	5
NONDQUAR	1000	638	3400	108	741	3936
NONMSQRT	529	–	–	–	–	–
OSBORNEB	11	232	420	0	232	420
OSCIGRAD	1000	60	97	0	60	97
OSCIPATH	100	–	–	–	–	–
PARKCH	15	1205	3844	0	1205	3844
PENALTY1	1000	6	25	4	6	25
PENALTY2	200	–	–	–	–	–
PENALTY3	200	75	190	1	75	163

Table 2 continued

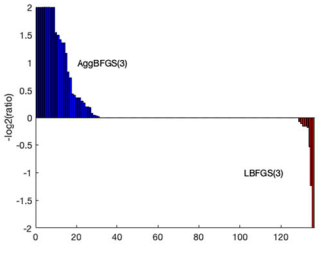
Name	n	AggBFGS(5)			LBFGS(5)	
		Iters.	Funcs.	Aggs.	Iters.	Funcs.
POWELLSG	1000	30	70	24	33	122
POWER	1000	159	806	34	586	4460
QUARTC	1000	311	1838	6	511	2973
SBRYBND	1000	–	–	–	–	–
SCHMVETT	1000	219	1121	0	219	1121
SCOSINE	1000	–	–	–	–	–
SCURLY10	1000	124	675	13	544	3986
SCURLY20	1000	122	720	17	800	6386
SCURLY30	1000	49	306	7	154	1088
SENSORS	100	62	151	0	62	151
SINQUAD	1000	–	–	–	23	78
SPARSINE	1000	474	474	0	474	474
SPARSQUR	1000	67	294	4	111	580
SPMSRTLS	1000	167	369	0	167	369
SROSENBR	1000	12	35	9	12	35
SSBRYBND	1000	4932	28846	0	4932	28846
SSCOSINE	1000	–	–	–	–	–
STRATEC	10	–	–	–	–	–
TESTQUAD	1000	2499	12256	12	1725	7207
TOINTGOR	50	237	254	0	237	254
TOINTGSS	1000	17	52	1	17	52
TOINTPSP	50	108	186	0	108	186
TOINTQOR	50	73	100	0	73	100
TQUARTIC	1000	14	53	10	15	56
TRIDIA	1000	344	354	0	344	354
VARDIM	200	2	2	0	2	2
VAREIGVL	100	34	52	0	34	52
WATSON	31	594	2447	0	594	2447
WOODS	1000	20	58	1	20	58
YATP1LS	120	47	203	43	38	188
YATP2LS	120	10	19	7	11	20

that can behave equivalently to a full-memory method, meaning that it can offer all theoretical properties of a full-memory scheme, such as local superlinear guarantees for the (outer) optimization method. We have also shown that the application of our aggregation scheme within an L-BFGS method can lead to performance gains over standard L-BFGS.

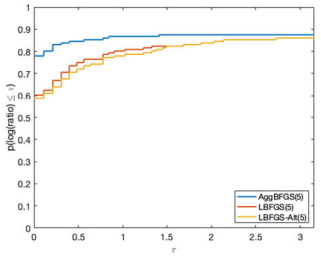
Our methodology could be extended to other quasi-Newton schemes. In particular, by the well-known symmetry between the BFGS and DFP updating, it is straightfor-



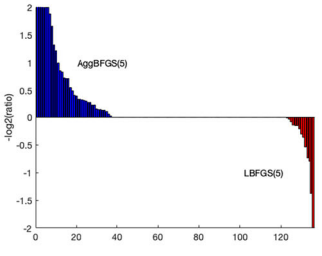
(a) Dolan and Moré Profile, $m = 3$



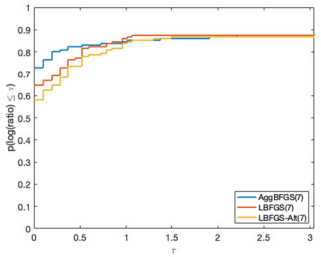
(b) Morales Outperformance Profile, $m = 3$



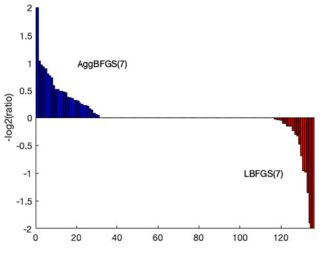
(c) Dolan and Moré Profile, $m = 5$



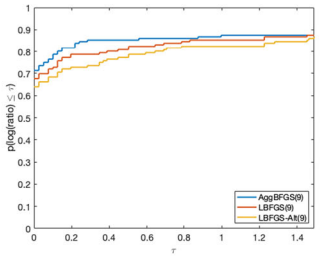
(d) Morales Outperformance Profile, $m = 5$



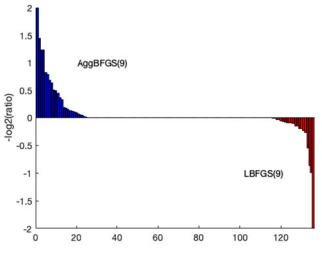
(e) Dolan and Moré Profile, $m = 7$



(f) Morales Outperformance Profile, $m = 7$

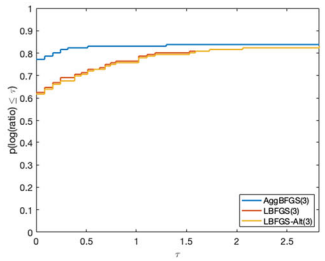


(g) Dolan and Moré Profile, $m = 9$

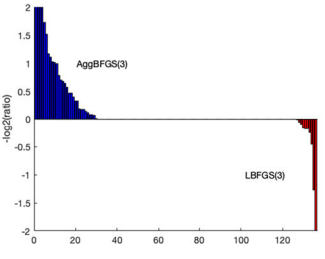


(h) Morales Outperformance Profile, $m = 9$

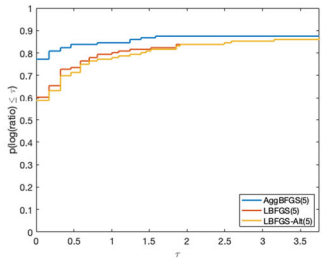
Fig. 4 Performance profiles for iterations



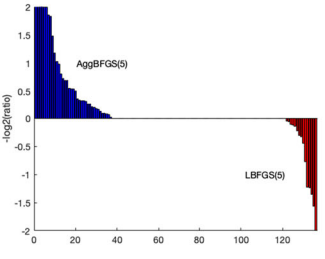
(a) Dolan and Moré Profile, $m = 3$



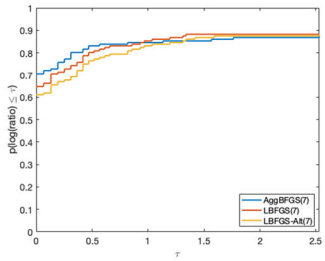
(b) Morales Outperformance Profile, $m = 3$



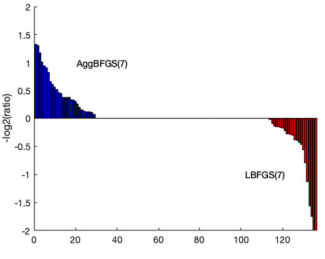
(c) Dolan and Moré Profile, $m = 5$



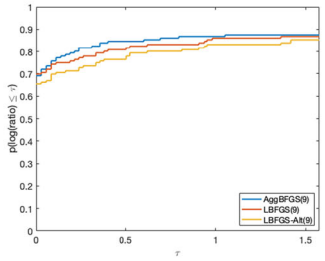
(d) Morales Outperformance Profile, $m = 5$



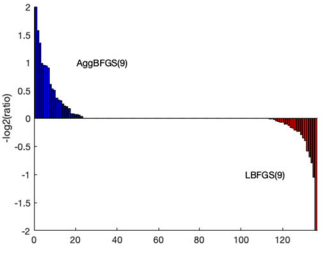
(e) Dolan and Moré Profile, $m = 7$



(f) Morales Outperformance Profile, $m = 7$



(g) Dolan and Moré Profile, $m = 9$



(h) Morales Outperformance Profile, $m = 9$

Fig. 5 Performance profiles for function evaluations

ward to extend our approach for DFP. In particular, with DFP, rather than looking for linear dependence between iterate displacements, one should look for linear dependence between gradient displacements. If linear dependence is observed, then one can aggregate the iterate displacements to obtain

$$\tilde{S}_{1:m} = M^{-1} Y_{1:m} \begin{bmatrix} A & 0 \end{bmatrix} + s_0 \begin{bmatrix} b \\ 0 \end{bmatrix}^T + S_{1:m} \tag{6.1}$$

(cf. (3.7)) such that $\mathrm{DFP}(M, S_{0:m}, Y_{0:m}) = \mathrm{DFP}(M, \tilde{S}_{1:m}, Y_{1:m})$ for $M \succ 0$. There might also be opportunities for extending our approach for the other members of the Broyden class of updates, although such extensions are not as straightforward. Indeed, for members of the class besides BFGS and DFP, one likely needs to modify both iterate and gradient displacements to aggregate curvature information.

There are also other opportunities for designing practical adaptations of our scheme. Perhaps the most straightforward idea is the following: Suppose that one is employing an L-BFGS(m) approach, one has m curvature pairs already stored, and one performs a new optimization algorithm iteration to yield a new curvature pair. Rather than simply discard the oldest stored curvature pair if a previous iterate displacement does not lie in the span of subsequent displacements, one could project this pair's iterate displacement onto the span of the subsequent displacements (and possibly project the pair's gradient displacement onto a subspace), then apply our aggregation scheme. This offers the opportunity to maintain more historical curvature information while still only storing/employing at most m pairs of vectors. One might also imagine other approaches for projecting information into smaller-dimensional subspaces in order to employ our scheme, rather than simply discarding old information. Such techniques might not attain the theoretical properties of a full-memory approach, but could lead to practical benefits.

One could employ our aggregation scheme with no modifications necessary if one employs an optimization algorithm that intentionally computes sequences of steps in low-dimensional subspaces, such as in block-coordinate descent. Another setting of interest is large-scale constrained optimization where the number of degrees of freedom (i.e., number of variables minus the number of active constraints) is small relative to n. For such a problem, various algorithms compute search directions that lie in subspaces that are low-dimensional relative to n.

References

1. Berahas, A. S., J. Nocedal, and M. Takáč. A multi-batch L-BFGS method for machine learning. In: Advances in Neural Information Processing Systems, pp. 1055–1063 (2016)
2. Berahas, A.S., Takáč, M.: A robust multi-batch L-BFGS method for machine learning. Optim. Methods Softw. **35**(1), 191–219 (2020)
3. Boggs, P.T., Byrd, R.H.: Adaptive, limited-memory BFGS algorithms for unconstrained optimization. SIAM J. Optim. **29**(2), 1282–1299 (2019)
4. Bonnans, J.F., Gilbert, JCh., Lemaréchal, C., Sagastizábal, C.A.: A family of variable metric proximal methods. Math. Progr. **68**(1), 15–47 (1995)
5. Broyden, C.G.: The convergence of a class of double-rank minimization algorithms. J. Inst. Math. Appl. **6**(1), 76–90 (1970)

6. Byrd, R.H., Hansen, S.L., Nocedal, J., Singer, Y.: A stochastic quasi-Newton method for large-scale optimization. SIAM J. Optim. **26**(2), 1008–1031 (2016)
7. Byrd, R.H., Nocedal, J.: A tool for the analysis of quasi-Newton methods with application to unconstrained minimization. SIAM J. Numer. Anal. **26**(3), 727–739 (1989)
8. Byrd, R.H., Nocedal, J., Schnabel, R.B.: Representations of quasi-Newton matrices and their use in limited memory methods. Math. Program. **63**, 129–156 (1994)
9. Byrd, R.H., Nocedal, J., Yuan, Y.: Global convergence of a class of quasi-Newton methods on convex problems. SIAM J. Numer. Anal. **24**(5), 1171–1189 (1987)
10. Curtis, F. E.: A self-correcting variable-metric algorithm for stochastic optimization. In: Proceedings of the 48th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 48, pp. 632–641, New York, USA (2016)
11. Curtis, F.E., Que, X.: An adaptive gradient sampling algorithm for nonsmooth optimization. Opt. Meth. Softw. **28**(6), 1302–1324 (2013)
12. Curtis, F.E., Que, X.: A quasi-Newton algorithm for nonconvex, nonsmooth optimization with global convergence guarantees. Math. Program. Comput. **7**, 399–428 (2015)
13. Curtis, F.E., Robinson, D.P., Zhou, B.: A self-correcting variable-metric algorithm framework for nonsmooth optimization. IMA J. Numer. Anal. **40**(2), 1154–1187 (2019)
14. Davidon, W.C.: Variable metric method for minimization. SIAM J. Optim. **1**(1), 1–17 (1991)
15. Dennis, J.E., Moré, J.J.: A characterization of superlinear convergence and its application to quasi-Newton methods. Math. Comput. **28**(126), 549–560 (1974)
16. Dennis, J.E., Schnabel, R.B.: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Society for Industrial and Applied Mathematics (SIAM), Philadelphia (1996)
17. Dolan, E.D., More, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**(2), 201–213 (2002)
18. Fletcher, R.: A new approach to variable metric algorithms. Comput. J. **13**(3), 317–322 (1970)
19. Gill, P.E., Golub, G.H., Murray, W., Saunders, M.A.: Methods for modifying matrix factorizations. Math. Comput. **126**(28), 505–535 (1974)
20. Goldfarb, D.: A family of variable metric updates derived by variational means. Math. Comput. **24**(109), 23–26 (1970)
21. Gould, N.I.M., Orban, D., Toint, PhL: CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization. Comput. Optim. Appl. **60**(3), 545–557 (2015)
22. Gower, R. , Goldfarb, D., Richtárik, P.: Stochastic block BFGS: squeezing more curvature out of data. In: International Conference on Machine Learning, pp. 1869–1878 (2016)
23. Haarala, N., Miettinen, K., Mäkelä, M.M.: New limited memory bundle method for large-scale nonsmooth optimization. Optim. Methods Softw. **19**(6), 673–692 (2004)
24. Keskar, N. S., Berahas, A. S.: ADAQN: an adaptive quasi-newton algorithm for training RNNs. In: *Joint European conference on machine learning and knowledge discovery in databases*, pp 1–16. Springer (2016)
25. Kolda, T.G., O'Leary, D.P., Nazareth, L.: BFGS with update skipping and varying memory. SIAM J. Optim. **8**(4), 1060–1083 (1998)
26. Lewis, A.S., Overton, M.L.: Nonsmooth optimization via quasi-Newton methods. Math. Program. **141**(1), 135–163 (2013)
27. Mifflin, R., Sun, D., Qi, L.: Quasi-Newton bundle-type methods for nondifferentiable convex optimization. SIAM J. Optim. **8**(2), 583–603 (1998)
28. Mokhtari, A., Ribeiro, A.: Global convergence of online limited memory BFGS. J. Mach. Learn. Res. **16**(1), 3151–3181 (2015)
29. Morales, J.L.: A numerical study of limited memory BFGS methods. Appl. Math. Lett. **15**, 481–487 (2002)
30. Nocedal, J.: Updating quasi-Newton matrices With limited storage. Math. Comput. **35**(151), 773–782 (1980)
31. Nocedal, J., Wright, S.J.: Numerical Optimization, 2nd edn. Springer, New York (2006)
32. Pearson, J.D.: Variable metric methods of minimisation. Comput. J. **12**(2), 171–178 (1969)
33. Powell, M.J.D.: Some global convergence properties of a variable metric algorithm for minimization with exact line searches. In: Cottle, R.W., Lemke, C.E. (eds.) Nonlinear Programming, SIAM-AMS Proceedings, Harwell, England, vol. IX. American Mathematical Society (1976)
34. Ritter, K.: Local and superlinear convergence of a class of variable metric methods. Computing **23**(3), 287–297 (1979)

35. Ritter, K.: Global and Superlinear Convergence of a Class of Variable Metric Methods, pp. 178–205. Springer, Berlin (1981)
36. Rosenbrock, H.H.: An automatic method for finding the greatest or least value of a function. Comput. J. **3**(3), 175–184 (1960)
37. Schraudolph, N. N., Yu, J., Günter, S.: A stochastic quasi-Newton method for online convex optimization. In: Artificial Intelligence and Statistics, pp. 436–443 (2007)
38. Shanno, D.F.: Conditioning of quasi-Newton methods for function minimization. Math. Comput. **24**(111), 647–656 (1970)
39. Vlček, J., Lukšan, L.: Globally convergent variable metric method for nonconvex nondifferentiable unconstrained minimization. J. Optim. Theory Appl. **111**(2), 407–430 (2001)
40. Wang, X., Ma, S., Goldfarb, D., Liu, W.: Stochastic quasi-Newton methods for nonconvex stochastic optimization. SIAM J. Optim. **27**(2), 927–956 (2017)