Limited-Memory BFGS with Displacement Aggregation

Albert S. Berahas

Department of Industrial and Systems Engineering, Lehigh University

Frank E. Curtis

Department of Industrial and Systems Engineering, Lehigh University

Baoyu Zhou

Department of Industrial and Systems Engineering, Lehigh University

COR@L Technical Report 19T-001

Limited-Memory BFGS with Displacement Aggregation

Albert S. Berahas*[1], Frank E. Curtis†[2], and Baoyu Zhou‡[3]

[1]Department of Industrial and Systems Engineering, Lehigh University
[2]Department of Industrial and Systems Engineering, Lehigh University
[3]Department of Industrial and Systems Engineering, Lehigh University

March 8, 2019

Abstract

A displacement aggregation strategy is proposed for the curvature pairs stored in a limited-memory 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.

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. 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. Quasi-Newton methods lie between the aforementioned methods; they only require gradient information, yet by updating and employing (inverse) 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 [13]. For further information on their theoretical properties, typically when coupled with line search mechanisms to ensure convergence, see, e.g., [5, 7, 14, 15, 26, 27, 28, 29, 30]. Within the class of quasi-Newton methods, algorithms that employ Broyden–Fletcher–Goldfarb–Shanno (BFGS) approximations of (inverse) Hessian matrices have enjoyed particular success; see [4, 16, 18, 33]. This is true when minimizing smooth objectives, as is the focus in the

*E-mail: albertberahas@gmail.com
†E-mail: frank.e.curtis@gmail.com
‡E-mail: baoyu.zhou@lehigh.edu

aforementioned references, but also when minimizing nonsmooth [3, 10, 11, 20, 22, 23, 34] or stochastic [1, 8, 9, 19, 21, 24, 32, 35] functions.

For solving large-scale problems, i.e., for minimizing objective functions involving thousands or millions of variables, *limited-memory* variants of quasi-Newton methods, such as the limited-memory variant of BFGS [25] (known as L-BFGS), have been successful for various applications. The main benefit of a limited-memory scheme is that one need not store the often-dense (inverse) Hessian approximation; rather, one need only store a few pairs of vectors, known as *curvature pairs*, with dimension equal to the number of variables. 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 [2] has attempted to develop an adaptive limited-memory 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.

We are motivated by the following question: Is it possible to design an *adaptive* limited-memory BFGS scheme that stores/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*[1] BFGS scheme that behaves *exactly* as a full-memory BFGS scheme—in the sense that the sequence of (inverse) 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 in order to capture the information that would be stored in a full-memory BFGS (inverse) Hessian approximation.

We remark at the outset that the strategy proposed in this paper might not immediately offer practical advantages over full-memory or limited-memory schemes already in the literature. Indeed, to attain full theoretical benefits, 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, we contend that our displacement aggregation strategy and corresponding theoretical results establish a solid foundation upon which one may design practically efficient approaches. Accordingly, we discuss how our approach can be implemented as efficiently as possible, and in our concluding remarks we outline alternative adaptive limited-memory BFGS schemes—that, like other efficient limited-memory schemes, maintain only a relatively small number of pairs—that might be built upon our strategy. For such adaptive schemes, we argue that one would be able to maintain *more history* in the same number of curvature pairs than in previously proposed limited-memory methods. Our concluding remarks also include discussions of how our approach can be extended to Davidon-Fletcher-Powell (DFP) quasi-Newton updating [13], and the opportunities and challenges of extending it to the entire Broyden class of updates [15].

1.1 Contributions

We propose and analyze a *displacement aggregation* strategy for modifying the displacement pairs stored in a limited-memory BFGS (i.e., 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 (inverse) 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

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

the dimension of the problem while generating the same sequence of (inverse) Hessian approximations that would have been generated in a full-memory BFGS scheme. Employed within a minimization algorithm, this leads to an adaptive limited-memory BFGS scheme—for which the number of stored pairs need never be more than the dimension of the problem—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 refer to our proposed "aggregated BFGS" scheme as `Agg-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

$$\min_{x\in\mathbb{R}^n} \; f(x), \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable. Corresponding to derivatives of f, we also 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 problem (1), we append a natural number as a subscript for a quantity to denote its value during each 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 (inverse) 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 §2, we provide background on BFGS updating. In §3, we motivate, present, and analyze our displacement aggregation approach, referred to as `Agg-BFGS`. The results of numerical experiments with `Agg-BFGS` are provided in §4, where our main goal is to demonstrate that the approach can be used in practice to accurately represent the BFGS inverse Hessian approximations that would result from a full-memory BFGS approach. Concluding remarks are given in §5.

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 $s_{k+1} \leftarrow \alpha_{k+1} d_{k+1} = -\alpha_{k+1} M_{k+1}^{-1} g_{k+1}$, where α_{k+1} is chosen by a line search for f from x_{k+1} along d_{k+1}. With f_{k+1} and g_{k+1} determined by function and gradient evaluations at x_{k+1}, respectively, all that remains toward specifying the model m_{k+1} is to choose M_{k+1}. In a quasi-Newton method [13], 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*

$$M_{k+1} s_k = g_{k+1} - g_k =: y_k. \tag{2}$$

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

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

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}); see [26]. 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 $k \in \mathbb{N}$; again, one can refer to [26].

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 the Hessian approximations $\{M_k\}$ 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 [26]. For our purposes, it will be convenient to refer to two ways of constructing such approximations: an iterative form and a compact form [6]. 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.,

$$S = \begin{bmatrix} s_1 & \cdots & s_m \end{bmatrix} \in \mathbb{R}^{n \times m} \tag{3a}$$

$$\text{and } \ Y = \begin{bmatrix} y_1 & \cdots & y_m \end{bmatrix} \in \mathbb{R}^{n \times m} \tag{3b}$$

where

$$\rho = \begin{bmatrix} \frac{1}{s_1^T y_1} & \cdots & \frac{1}{s_m^T y_m} \end{bmatrix}^T \in \mathbb{R}^m_{>0}. \tag{4}$$

Algorithm 1 computes the (L-)BFGS inverse Hessian approximation from an initial matrix $W \succ 0$ and the displacement information in (3). In the context of an optimization algorithm using BFGS updating, this matrix would be set with a single update in each iteration rather than re-generated from scratch in every iteration using historical iterate/gradient displacements. However, we write the strategy in this iterative form for convenience of our analysis. The output $\overline{W}$ represents the matrix to be employed in the step computation of the (outer) optimization algorithm. As a function of the inputs, we denote the output as $\overline{W} = \text{BFGS}(W, S, Y)$.

The updates performed in the loop of Algorithm 1 correspond to a set of projections and corresponding corrections. In particular, for each $j \in \{1, \dots, m\}$, 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 [12, Appendix B]. In this manner, one can understand the well-known fact that each update in a (L-)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 (L-)BFGS approximation from the initial approximation by combining all low-rank changes directly. The scheme in Algorithm 2, which shows the compact form of the updates, generates the same output as Algorithm 1; see [6].

We note in passing that for computing a matrix-vector product with an L-BFGS approximation without constructing the approximation matrix itself, it is well-known that one can use the so-called two-loop recursion [25].

Algorithm 1 : (L-)BFGS Matrix Construction, Iterative Form

```
Require: $W \succ 0$ and $(S, Y)$ as in (3), with $\rho$ as in (4).
1: Initialize $\overline{W} \leftarrow W$.
2: for $j = 1, \dots, m$ do
3:     Set
           $U_j \leftarrow I - \rho_j y_j s_j^T$,   (5a)
           $V_j \leftarrow \rho_j s_j s_j^T$,   (5b)
       and $\overline{W} \leftarrow U_j^T \overline{W} U_j + V_j$.   (5c)
4: end for
5: return $\overline{W} \equiv \text{BFGS}(W, S, Y)$
```

Algorithm 2 : (L-)BFGS Matrix Construction, Compact Form

```
Require: $W \succ 0$ and $(S, Y)$ as in (3), with $\rho$ as in (4).
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 $1 \le i \le j \le m$ and $D_{i,i} \leftarrow s_i^T y_i$
   for all $i \in \{1, \dots, m\}$ (with all other elements being zero), i.e.,
       $R \leftarrow \begin{bmatrix} s_1^T y_1 & \cdots & s_1^T y_m \\ & \ddots & \vdots \\ & & s_m^T y_m \end{bmatrix}$ and $D \leftarrow \begin{bmatrix} s_1^T y_1 & & \\ & \ddots & \\ & & s_m^T y_m \end{bmatrix}$.   (6)
2: Set
       $\overline{W} \leftarrow W + \begin{bmatrix} S & WY \end{bmatrix} \begin{bmatrix} R^{-T}(D + Y^T W Y) R^{-1} & -R^{-T} \\ -R^{-1} & 0 \end{bmatrix} \begin{bmatrix} S^T \\ Y^T W \end{bmatrix}$.   (7)
3: return $\overline{W} \equiv \text{BFGS}(W, S, Y)$
```

2.2 Convergence and Local Rate Guarantees

An optimization algorithm that uses a BFGS scheme for updating (inverse) 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, 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. For further information, the interested reader can consult [14, 15].

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 BFGS. For concreteness, we state one such result as the following theorem; see Theorem 6.6 in [26]. 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*

$$d_k \leftarrow -W_k g_k \quad \text{and} \quad x_{k+1} \leftarrow x_k + \alpha_k d_k,$$

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 to satisfy the Armijo-Wolfe conditions; see equation (3.6) in [26]. In addition, suppose that the algorithm converges to a point $x_* \in \mathbb{R}^n$ at which the Hessian is Lipschitz continuous in such *a way that*

$$\sum_{k=1}^{\infty} \|x_k - x_*\| < \infty.$$

Then, $\{x_k\}$ converges to x_* at a superlinear rate.

Such a result cannot be proved for L-BFGS [25]. Common theoretical results for L-BFGS merely show that if the (inverse) Hessian approximations are sufficiently positive definite and bounded, then 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 the context of 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* (3)*, with ρ as in* (4)*, and suppose that $s_j = \tau s_{j+1}$ for some $j \in \{1, \dots, m-1\}$ 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 (and 2) 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 $j \in \{1, \dots, m-1\}$, 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 (5),

$$W_{1:j+1} = U_{j+1}^T U_j^T W_{1:j-1} U_j U_{j+1} + U_{j+1}^T V_j U_{j+1} + V_{j+1}. \tag{8}$$

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

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

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 (8), 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 (5) 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 updating. 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 whether the gradient displacement vectors are the same or different—and as a result the same matrix is derived if the former update is skipped.

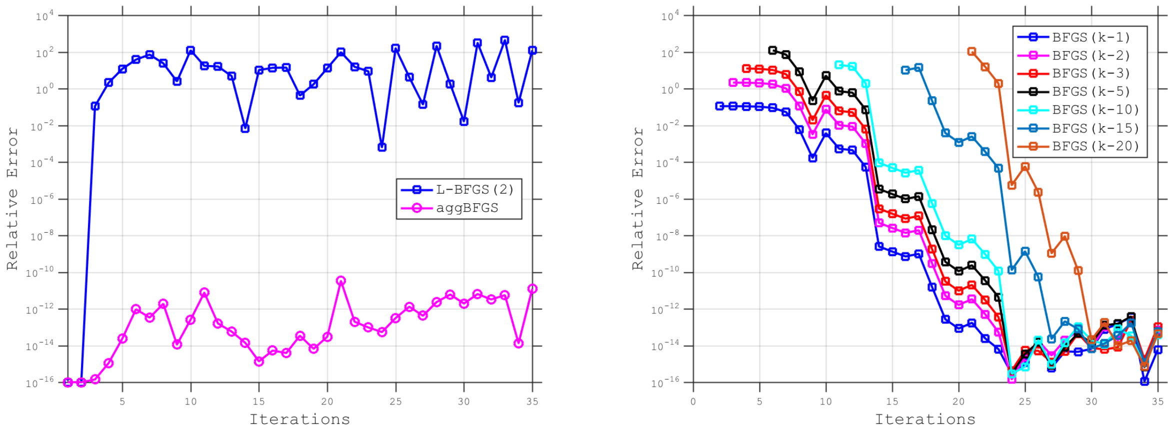
What if a previous iterate displacement is not parallel with a subsequent displacement, but is in the span of multiple subsequent displacements? One might suspect that the information in the previous displacement might get over-written and could be ignored. As it happens, *it is not so simple.* Allow us to 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 $\mathbb{R}^n$. Then, 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, even for $k > n$. However, one does not find this to be the case. In Figure 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 [31]

$$f(x_{(1)}, x_{(2)}) = 100(x_{(2)} - x_{(1)}^2)^2 + (1 - x_{(1)})^2. \tag{9}$$

  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 Figure 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 + 1st iteration. One finds, for example, that the information from the first pair lingers—in the sense that it influences the BFGS approximation in a non-trivial manner—for iteration numbers beyond $n = 2$.



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

Figure 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.

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 (inverse) Hessian of the quadratic can be recovered *exactly* if n iterations are performed; see [15, 26]. 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^TWx 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 [15, 26].

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 (3) with inner products yielding (4), one also has a previous curvature pair

$$(s_0, y_0) \in \mathbb{R}^n \times \mathbb{R}^n \quad \text{with} \quad \rho_0 = 1/s_0^T y_0 > 0 \tag{10}$$

such that

$$s_0 = S_{1:m}\tau \quad \text{for some} \quad \tau \in \mathbb{R}^m. \tag{11}$$

(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 remainder of the paper.) Given the linear dependence of the iterate displacement s_0 on the iterate displacements in the matrix $S_{1:m}$, our goal is to determine *aggregated* gradient displacements

$$\tilde{Y}_{1:m} := \begin{bmatrix}\tilde{y}_1 & \cdots & \tilde{y}_m\end{bmatrix} \tag{12}$$

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

$$\text{BFGS}(W, S_{0:m}, Y_{0:m}) = \text{BFGS}(W, S_{1:m}, \tilde{Y}_{1:m}). \tag{13}$$

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 3.1. *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* $\{k_1, \dots, k_m\}$ *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 §3.4.*

A basic view of our approach is stated as Algorithm 3, wherein we invoke the compact form of BFGS updates (recall (7)). However, while Algorithm 3 provides an understanding of the intent of our displacement aggregation scheme, it does not specify the manner in which $\tilde{Y}_{1:m}$ yielding (13) can be found. It is also does not address the fact that multiple such matrices might exist. Toward a specific scheme, one can expand the compact forms of $\text{BFGS}(W, S_{0:m}, Y_{0:m})$ and $\text{BFGS}(W, S_{1:m}, \tilde{Y}_{1:m})$, specify that the aggregated displacements have the form

$$\tilde{Y}_{1:m} = W^{-1} S_{1:m} \begin{bmatrix} A & 0 \end{bmatrix} + y_0 \begin{bmatrix} b \\ 0 \end{bmatrix}^T + Y_{1:m} \tag{14}$$

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 (13) holds. The zero blocks in the variable matrices in (14) are motivated by Theorem 3.1, since in the case of $m = 1$ one can set $\tilde{Y}_{1:1} = Y_{1:1}$; otherwise, the values of A and b need to be computed to satisfy (13).

Following these steps, one obtains the more detailed view of our scheme that is stated in Algorithm 4, where the three key equations are stated as (16).

Algorithm 4 does not specify a precise scheme for computing (A, b) in order to satisfy (16). We specify such a scheme in §3.4 after first showing in the following subsection that real values for (A, b) always exist to satisfy (16).

3.3 Existence of Real Solutions for `Agg-BFGS`

Our goal in this subsection 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* (3) *with the columns of $S_{1:m}$ being linearly independent and the vector $\rho_{1:m}$ as defined in* (4), *and*
- (s_0, y_0, ρ_0) *defined as in* (10) *such that* (11) *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 (14), *the equations* (16) *hold. Consequently, for this $\tilde{Y}_{1:m}$, one finds that*

$$s_i^T \tilde{y}_i = s_i^T y_i > 0 \ \ \textit{for all} \ \ i \in \{1, \dots, m\} \tag{17}$$

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

```
Algorithm 3 : Displacement Aggregation, Basic View
Require: $W \succ 0$, $(S_{1:m}, Y_{1:m}, \rho_{1:m})$ as in (3)–(4), and $(s_0, y_0, \rho_0)$ as in (10)–(11).
1: Set $\tilde{Y}_{1:m}$ as in (12) such that, with
     $$\tilde{R}_{1:m} := \begin{bmatrix} s_1^T\tilde{y}_1 & \cdots & s_1^T\tilde{y}_m \\ & \ddots & \vdots \\ & & s_m^T\tilde{y}_m \end{bmatrix} \text{ and } \tilde{D}_{1:m} := \begin{bmatrix} s_1^T\tilde{y}_1 & & \\ & \ddots & \\ & & s_m^T\tilde{y}_m \end{bmatrix}, \quad (15)$$
   one finds with
     $$R_{0:m} := \begin{bmatrix} s_0^Ty_0 & s_0^TY_{1:m} \\ & R_{1:m} \end{bmatrix} \text{ and } D_{0:m} := \begin{bmatrix} s_0^Ty_0 & \\ & D_{1:m} \end{bmatrix}$$
   that
     $$\begin{aligned} &\text{BFGS}(W, S_{0:m}, Y_{0:m}) \\ &\equiv W + \begin{bmatrix} S_{0:m} & WY_{0:m} \end{bmatrix} \begin{bmatrix} R_{0:m}^{-T}(D_{0:m} + Y_{0:m}^TWY_{0:m})R_{0:m}^{-1} & -R_{0:m}^{-T} \\ -R_{0:m}^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_{0:m}^T \\ Y_{0:m}^TW \end{bmatrix} \\ &= W + \begin{bmatrix} S_{1:m} & W\tilde{Y}_{1:m} \end{bmatrix} \begin{bmatrix} \tilde{R}_{1:m}^{-T}(\tilde{D}_{1:m} + \tilde{Y}_{1:m}^TW\tilde{Y}_{1:m})\tilde{R}_{1:m}^{-1} & -\tilde{R}_{1:m}^{-T} \\ -\tilde{R}_{1:m}^{-1} & 0 \end{bmatrix} \begin{bmatrix} S_{1:m}^T \\ \tilde{Y}_{1:m}^TW \end{bmatrix} \\ &\equiv \text{BFGS}(W, S_{1:m}, \tilde{Y}_{1:m}). \end{aligned}$$
2: return $\tilde{Y}_{1:m}$.
```

Proof. First, observe that there are $(m+1)(m-1)$ unknowns in the formula (14) 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 (16), first notice that (14) imposes $\tilde{y}_m = y_m$. Hence, by ignoring the last column of $R_{1:m}$ to define the submatrix

$$P := \begin{bmatrix} s_1^Ty_1 & \cdots & s_1^Ty_{m-1} \\ 0 & \ddots & \vdots \\ \vdots & \ddots & s_{m-1}^Ty_{m-1} \\ 0 & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{m\times(m-1)}, \tag{18}$$

and similarly defining $\tilde{P}$ with size $m \times (m-1)$ as a submatrix of $\tilde{R}_{1:m}$, the key equations in (16) can be simplified to have the following form:

$$\tilde{P} = P, \tag{19a}$$

$$b = -\rho_0(S_{1:m}^TY_{1:m-1} - P)^T\tau, \tag{19b}$$

$$\begin{aligned} \text{and} \quad &(\tilde{Y}_{1:m-1} - Y_{1:m-1})^TW(\tilde{Y}_{1:m-1} - Y_{1:m-1}) \\ &= \frac{\chi_0}{\rho_0}bb^T - A^T(S_{1:m}^TY_{1:m-1} - P) - (S_{1:m}^TY_{1:m-1} - P)^TA. \end{aligned} \tag{19c}$$

Observing the number of nonzero entries in (19a) (recall (18)) and recognizing the symmetry in (19c), one finds that the number of equations to be satisfied are:

$$\underbrace{m(m-1)/2}_{\text{(in (19a))}} + \underbrace{(m-1)}_{\text{(in (19b))}} + \underbrace{m(m-1)/2}_{\text{(in (19c))}} = \underbrace{(m+1)(m-1)}_{\text{(in (19))}}.$$

Hence, (19) (and so (16)) is a square system of linear and quadratic equations to be solved for the unknowns in the matrix A and vector b.

Algorithm 4 : Displacement Aggregation, Detailed View

```
Require: $W \succ 0$, $(S_{1:m}, Y_{1:m}, \rho_{1:m})$ as in (3)–(4), and $(s_0, y_0, \rho_0)$ as in (10)–(11).
1: Set $\tilde{Y}_{1:m}$ as in (14) such that, with $\chi_0 := 1 + \rho_0 \|y_0\|_W^2$, one finds
       $\tilde{R}_{1:m} = R_{1:m},$   (16a)
       $\begin{bmatrix} b \\ 0 \end{bmatrix} = -\rho_0 (S_{1:m}^T Y_{1:m} - R_{1:m})^T \tau,$ and   (16b)
       $(\tilde{Y}_{1:m} - Y_{1:m})^T W (\tilde{Y}_{1:m} - Y_{1:m}) = \frac{\chi_0}{\rho_0} \begin{bmatrix} b \\ 0 \end{bmatrix} \begin{bmatrix} b \\ 0 \end{bmatrix}^T - \begin{bmatrix} A & 0 \end{bmatrix}^T (S_{1:m}^T Y_{1:m} - R_{1:m}) - (S_{1:m}^T Y_{1:m} - R_{1:m})^T \begin{bmatrix} A & 0 \end{bmatrix}.$   (16c)
2: return $\tilde{Y}_{1:m}$.
```

Equation (19b) 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

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

Using this notation, one finds that (19a) reduces to the system of affine equations

$$S_{1:j}^T W^{-1} S_{1:m} a_j = -b_j S_{1:j}^T y_0 \text{ for all } j \in \{1, \dots, m-1\}. \tag{20}$$

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

$$a_j = Q^{-1} \begin{bmatrix} a_{j,1} \\ a_{j,2} \end{bmatrix}, \text{ where } Q := S_{1:m}^T W^{-1} S_{1:m} \succ 0, \tag{21}$$

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 (20) to be satisfied, it follows that one must have

$$a_{j,1} = -b_j S_{1:j}^T y_0 \in \mathbb{R}^j. \tag{22}$$

Moreover, with this value for $a_{j,1}$, it follows that (20), and hence (19a), is satisfied for any $a_{j,2}$. Going forward, our goal is to determine the existence of $a_{j,2} \in \mathbb{R}^{m-j}$ for all $j \in \{1, \dots, m-1\}$ such that (19c) holds, which will complete the proof.

Turning our attention to (19c), one finds with (14) that it may be written as

$$A^T Q A + \Omega^T A + A^T \Omega - \omega \omega^T = 0, \tag{23}$$

where

$$\Omega := S_{1:m}^T y_0 b^T + S_{1:m}^T Y_{1:m-1} - P \in \mathbb{R}^{m \times (m-1)}, \tag{24a}$$

$$\text{and } \omega := \frac{b}{\sqrt{\rho_0}} \in \mathbb{R}^{m-1}. \tag{24b}$$

One may rewrite equation (23) as

$$(QA + \Omega)^T Q^{-1} (QA + \Omega) = \omega \omega^T + \Omega^T Q^{-1} \Omega. \tag{25}$$

Let us now rewrite the equations in (25) in a particular form that will be useful for the purposes of our proof going forward. Consider the matrix $QA + \Omega$ in (25). By the definitions of $a_{j,1}$, $a_{j,2}$, Q, and Ω in (21)–(24), as well as of P from (18), 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 $j \in \{1,\dots,m-1\}$,

$$\phi_j(a_{j,2}) := L\left(\begin{bmatrix} 0_j \\ a_{j,2} \end{bmatrix} + \begin{bmatrix} 0_j \\ S_{j+1:m}^T (b_j y_0 + y_j) \end{bmatrix}\right), \tag{26}$$

it follows that the $(i,j) \in \{1,\dots,m-1\} \times \{1,\dots,m-1\}$ element of (25) is

$$\phi_i(a_{i,2})^T \phi_j(a_{j,2}) = z_i^T z_j. \tag{27}$$

Using the notation introduced in the preceding paragraph, let us use an inductive argument to prove the existence of a real solution of (25). This induction will follow the indices $\{1,\dots,m-1\}$ *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 (27) is $\|\phi_{m-1}(a_{m-1,2})\|_2^2$. This is a strongly convex quadratic in λ_{m-1}. Since $\|\phi_{m-1}(a_{m-1,2}^*)\|_2 = 0$ and $z_{m-1}^T z_{m-1} \geq 0$, it follows that 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 (27).

Now suppose that there exists real $\{a_{\ell+1,2},\dots,a_{m-1,2}\}$ such that (27) holds for all (i,j) with $i \in \{\ell+1,\dots,m-1\}$ and $j \in \{i,\dots,m-1\}$. (By symmetry, these values also satisfy the (j,i) elements of (27) 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 (27) for all (i,j) with $i \in \{j,\dots,m-1\}$ and $j = \ell$, i.e., solving the following system for $a_{\ell,2}$:

$$\phi_{m-1}(a_{m-1,2})^T \phi_\ell(a_{\ell,2}) = z_{m-1}^T z_\ell \tag{28a}$$

$$\vdots$$

$$\phi_{\ell+1}(a_{\ell+1,2})^T \phi_\ell(a_{\ell,2}) = z_{\ell+1}^T z_\ell \tag{28b}$$

$$\phi_\ell(a_{\ell,2})^T \phi_\ell(a_{\ell,2}) = z_\ell^T z_\ell. \tag{28c}$$

Notice that (28a)–(28b) are affine equations in $a_{\ell,2}$, whereas (28c) is a quadratic equation in $a_{\ell,2}$. For all $t \in \{\ell+1,\dots,m-1\}$, let

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

so that, by (26), one may write

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

Our strategy is first to find $a^*_{\ell,2} \in \mathbb{R}^{m-\ell}$ satisfying (28a)–(28b) such that

$$a^*_{\ell,2} + S^T_{\ell+1:m}(b_\ell y_0 + y_\ell) \in \operatorname{span}\left\{ \begin{bmatrix} 0 \\ \psi_{\ell+1,\ell+1} \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ \psi_{\ell+1,m-1} \end{bmatrix} \right\} \tag{29}$$

and (cf. (28c))

$$\phi_\ell(a^*_{\ell,2})^T \phi_\ell(a^*_{\ell,2}) \le z_\ell^T z_\ell. \tag{30}$$

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 (28a)–(28b) for arbitrary one-dimensional λ_ℓ. From here, it will follow by the fact that the left-hand side of (28c) is a strongly convex quadratic in the unknown λ_ℓ and the fact that (30) 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 (28).

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 $\{t_1, \dots, t_c\} \subseteq \{\ell+1, \dots, m-1\}$ with

$$\operatorname{span}\left\{ \begin{bmatrix} 0 \\ \psi_{\ell+1,t_1} \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ \psi_{\ell+1,t_c} \end{bmatrix} \right\} = \operatorname{span}\left\{ \begin{bmatrix} 0 \\ \psi_{\ell+1,\ell+1} \end{bmatrix}, \dots, \begin{bmatrix} 0 \\ \psi_{\ell+1,m-1} \end{bmatrix} \right\}. \tag{31}$$

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

$$\psi_{\ell+1,t} = 0_{m-(\ell+1)} \ \text{ and } \ \phi_t(a_{t,2}) = 0_m \ \text{ for all } \ t \in \{\ell+1, \dots, m-1\}. \tag{32}$$

Hence, by our induction hypothesis, it follows from (27) and (32) that

$$z_t = 0_{m-1} \ \text{ for all } \ t \in \{\ell+1, \dots, m-1\}. \tag{33}$$

Consequently from (32) and (33), the affine equations (28a)–(28b) 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 (26) that

$$\phi_\ell(a^*_{\ell,2}) = L\left(\begin{bmatrix} 0_\ell \\ a^*_{\ell,2} + S^T_{\ell+1:m}(b_\ell y_0 + y_\ell) \end{bmatrix} \right) = 0_m, \tag{34}$$

which shows that this choice satisfies (30). Now consider the case when $c > 0$. For $a^*_{\ell,2}$ to satisfy (29), it follows with (31) that we must have

$$a^*_{\ell,2} + S^T_{\ell+1:m}(b_\ell y_0 + y_\ell) = \begin{bmatrix} 0 & \cdots & 0 \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix} \beta_\ell, \tag{35}$$

where β_ℓ has length c. Choosing

$$\beta_\ell := \left(\begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix}^T L^T L \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix} \right)^{-1} \begin{bmatrix} z^T_{t_1} \\ \vdots \\ z^T_{t_c} \end{bmatrix} z_\ell \in \mathbb{R}^c, \tag{36}$$

it follows with (35) that, for any $t \in \{t_1, \dots, t_c\}$, one finds

$$\begin{aligned} \phi_t(a_{t,2})^T \phi_\ell(a^*_{\ell,2}) &= \begin{bmatrix} 0_{\ell+1} \\ \psi_{\ell+1,t} \end{bmatrix}^T L^T L \begin{bmatrix} 0_\ell \\ a^*_{\ell,2} + S^T_{\ell+1:m}(b_\ell y_0 + y_\ell) \end{bmatrix} \\ &= \begin{bmatrix} 0_{\ell+1} \\ \psi_{\ell+1,t} \end{bmatrix}^T L^T L \begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,t_1} & \cdots & \psi_{\ell+1,t_c} \end{bmatrix} \beta_\ell = z_t^T z_\ell. \end{aligned} \tag{37}$$

We shall now prove that, for any $t \in \{\ell+1, \dots, m-1\} \backslash \{t_1, \dots, t_c\}$, 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 (31) 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 (31) along with the inductive hypothesis that, for any pair (i,j) with $i \in \{\ell+1,\dots,m-1\}$ and $j \in \{i,\dots,m-1\}$, one has

$$\begin{bmatrix} 0_{\ell+1} \\ \psi_{\ell+1,i} \end{bmatrix}^T L^T L \begin{bmatrix} 0_{\ell+1} \\ \psi_{\ell+1,j} \end{bmatrix} = \phi_i(a_{i,2})^T \phi_j(a_{j,2}) = z_i^T z_j, \tag{38}$$

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

$$\begin{aligned} \operatorname{rank}\left(\begin{bmatrix} z_{\ell+1} & \cdots & z_{m-1} \end{bmatrix}\right) &= \operatorname{rank}\left(\begin{bmatrix} \phi_{\ell+1}(a_{\ell+1,2}) & \cdots & \phi_{m-1}(a_{m-1,2}) \end{bmatrix}\right) \\ &= \operatorname{rank}\left(\begin{bmatrix} \phi_{t_1}(a_{t_1,2}) & \cdots & \phi_{t_c}(a_{t_c,2}) \end{bmatrix}\right) \\ &= \operatorname{rank}\left(\begin{bmatrix} z_{t_1} & \cdots & z_{t_c} \end{bmatrix}\right) = c. \end{aligned} \tag{39}$$

From (39), it follows that for any $t \in \{\ell+1,\dots,m-1\} \backslash \{t_1,\dots,t_c\}$ 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 (38), 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 $\gamma_\ell = \bar{\gamma}_\ell$. Thus, with (37) and the definitions of γ_ℓ and $\bar{\gamma}_\ell$, it follows for any $t \in \{\ell+1,\dots,m-1\} \backslash \{t_1,\dots,t_c\}$ that

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

Combining this with (37), it follows that $a_{\ell,2}^*$ from (35) with β_ℓ from (36) satisfies (28a)–(28b), as desired. Let us show now that this $a_{\ell,2}^*$ also satisfies (30). Indeed, by (35), (36), and (38), it follows that

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

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

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

$$\begin{bmatrix} z_{\ell_1} & \cdots & z_{\ell_c} \end{bmatrix} \left(\begin{bmatrix} z_{\ell_1}^T \\ \vdots \\ z_{\ell_c}^T \end{bmatrix} \begin{bmatrix} z_{\ell_1} & \cdots & z_{\ell_c} \end{bmatrix} \right)^{-1} \begin{bmatrix} z_{\ell_1}^T \\ \vdots \\ z_{\ell_c}^T \end{bmatrix}$$

all lie in the set $\{0, 1\}$. (As an aside, one finds that the inequality above is strict if $z_\ell \notin \text{span}\{z_{t_1}, \dots, z_{t_c}\}$.) Hence, we have shown that $a_{\ell,2}^*$ from (35) satisfies (30).

As previously mentioned (in the text following (30)), our goal now is to show that there exists a nonzero vector $\bar{a}_{\ell,2} \in \mathbb{R}^{m-\ell}$ such that $a_{\ell,2}^* + \lambda_\ell \bar{a}_{\ell,2}$ satisfies (28a)–(28b) for arbitrary λ_ℓ. From (28a)–(28b), such an $\bar{a}_{\ell,2} \in \mathbb{R}^{m-\ell}$ must satisfy

$$\begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1,\ell+1} & \cdots & \psi_{\ell+1,m-1} \end{bmatrix}^T L^T L \begin{bmatrix} 0_\ell \\ \bar{a}_{\ell,2} \end{bmatrix} = 0_{m-(\ell+1)}. \tag{40}$$

Since

$$\begin{bmatrix} 0_{\ell+1} & \cdots & 0_{\ell+1} \\ \psi_{\ell+1} & \cdots & \psi_{m-1} \end{bmatrix}^T L^T L \in \mathbb{R}^{(m-(\ell+1))\times m}, \tag{41}$$

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 (41). 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

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

one finds that $\bar{a}_{\ell,2}$ satisfies (40), as desired. Consequently, as stated in the text following (30), it follows by the fact that the left-hand side of (28c) is a strongly convex quadratic in the unknown λ_ℓ and the fact that (30) 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 (28).

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 (14), the equations (16) hold. The remaining desired conclusions of the theorem, namely, that (17) 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 (17) are a subset of the equations in (19a), and the fact that (16) were derived explicitly to ensure that, with $\tilde{Y}_{1:m}$ satisfying (14), one would find that (13) 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 an (inverse) Hessian approximation, the pairs used in our approach might come from a subset of all previous iterations, with the gradient displacement vectors 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), one has accumulated a set of curvature pairs, stored in the sets

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

where $\{k_i\}_{i=0}^{m-1} \subset \mathbb{N}$ with $k_i < k_{i+1}$ for all $i \in \{0, \dots, m-2\}$, 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 $\{k_0, \dots, k_{m-1}\}$, but, for simplicity of notation, we write them as such 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 $k_{m-1} < k_m$ has been made 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 shall henceforth simplify the subscript notation and refer to the "previously stored" displacement vectors as those in the sets $\{s_0, \dots, s_{m-1}\}$ and $\{y_0, \dots, y_{m-1}\}$, 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 $\{s_0, s_1, \dots, s_m\}$ is linearly independent. In this case, one simply adds the new curvature pair, which leads to the $(m+1)$-element sets

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

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

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

This choice is justified by Theorem 3.1.

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

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

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

$$\begin{bmatrix} s_{m-1} & \cdots & s_0 \end{bmatrix}^T \begin{bmatrix} s_{m-1} & \cdots & s_0 \end{bmatrix} = \Theta\Theta^T, \tag{42}$$

which exists due to the fact that the vectors $\{s_0, \dots, s_{m-1}\}$ 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 matrix $\Delta \in \mathbb{R}^{m\times m}$ such that

$$\begin{bmatrix} s_m & s_{m-1} & \cdots & s_0 \end{bmatrix}^T \begin{bmatrix} s_m & s_{m-1} & \cdots & s_0 \end{bmatrix} = \begin{bmatrix} \mu & \\ \delta & \Delta \end{bmatrix} \begin{bmatrix} \mu & \delta^T \\ & \Delta^T \end{bmatrix}. \tag{43}$$

As is well known, equating terms in (42) and (43) 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 [17]. 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). On the other hand, 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 $i \in \{1, \dots, m\}$, 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

$$\begin{bmatrix} s_m & s_{m-1} & \cdots & s_{m-i} \end{bmatrix}^T \begin{bmatrix} s_m & s_{m-1} & \cdots & s_{m-i} \end{bmatrix} = \begin{bmatrix} \Xi & \\ \xi^T & 0 \end{bmatrix} \begin{bmatrix} \Xi^T & \xi \\ & 0 \end{bmatrix}. \tag{44}$$

Letting $\tau \in \mathbb{R}^i$ be the unique vector satisfying $\Xi^T \tau = \xi$, one finds that the vector $[\tau, -1]^T$ lies in the null space of (44), 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 $\{s_{m-1}, \dots, s_{m-i}\}$ 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}).

Before moving forward, 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 $\{s_m^T s_{m-1}, \dots, s_m^T s_0\}$) 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 $s_j \in \operatorname{span}\{s_{j+1}, \dots, s_m\}$, and we have a vector $\tau \in \mathbb{R}^{m-j}$ such that $s_j = S_{j+1:m}\tau$. Our goal in this case is to apply our aggregation scheme to modify the gradient displacement vectors $\{y_{j+1}, \dots, y_m\}$ in order to compute

$$\tilde{Y}_{j+1:m} = W_{0:j-1}^{-1} S_{j+1:m} \begin{bmatrix} A & 0 \end{bmatrix} + y_0 \begin{bmatrix} b \\ 0 \end{bmatrix}^T + Y_{j+1:m}, \tag{45}$$

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 §3.2 and §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 (45). This matrix can be computed *without* needing to form 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 $\mathcal{O}(jmn) \leq \mathcal{O}(m^2 n)$; see §7.2 in [26].

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 the matrix $A \in \mathbb{R}^{m\times(m-1)}$ and vector $b \in \mathbb{R}^{m-1}$ in order to obtain $\tilde{Y}_{1:m}$ as in (14). By Theorem 3.2, it follows that real values for A and b exist to satisfy (16). The computation of the vector b is straightforward; it can be computed by the formula (19b). Assuming that the products in $S_{1:m}^T Y_{1:m-1}$ have already been computed (at cost $\mathcal{O}(m^2 n)$), 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 $\ell \in \{1, \dots, m-1\}$, and, as in (21), 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 $\mathcal{O}(m^2)$.

Using this established notation, let us now describe an approach for computing A. For each $\ell \in \{1, \dots, m-1\}$, the ℓ-element vector $a_{\ell,1}$ is given by the formula (22). (As explained in the proof of Theorem 3.2, these values are set to ensure that (19a) 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 $\mathcal{O}(m^2)$. As for $\{a_{\ell,2}\}_{\ell=1}^{m-1}$ (which are set to solve the system of linear and quadratic equations comprising (19c)), these are computed in reverse order, i.e., from $\ell = m-1$ to $\ell = 1$. For $\ell = m-1$, the unknown $a_{m-1,2}$ is a scalar, which one needs to compute by solving the quadratic equation (28c). Assuming that a factorization of Q is available, as has been mentioned, and assuming that the elements of the right-hand side of (25) have been pre-computed (at cost $\mathcal{O}(m^3)$), the cost of solving this quadratic equation is $\mathcal{O}(1)$. For the remaining vectors, namely, $\{a_{\ell,2}\}_{\ell=1}^{m-2}$, one needs to compute each to solve the affine equations stated as (28a)–(28b) in addition to the quadratic equation (28c). Following the proof of Theorem 3.2, one can compute in order for $\ell = m-2$ to $\ell = 1$ the values

(a) $a_{\ell,2}^*$ to satisfy (35)–(36),

(b) $\bar{a}_{\ell,2}$ to form a basis of the null space of the matrix in (41), which can be obtained via a QR factorization of the transpose of this matrix,

(c) and $\lambda_\ell^* \in \mathbb{R}$ such that $a_{j,2} = a_{j,2}^* + \lambda_j^* \bar{a}_{j,2}$ solves the quadratic equation (28c).

The most expensive operation in each iteration of this iterative scheme is the QR factorization of the matrix in (41), 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 $\mathcal{O}(m^4)$.

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 the total cost is $\mathcal{O}(m^2 n)$ (for computing the inner products of n-vectors required in the scheme, many of which are required by other full-memory and limited-memory schemes as well) plus $\mathcal{O}(m^4)$. This latter dominant cost is due as described above for computing the "lower half" of a matrix used to determine A. As mentioned in §1, this scheme is less efficient even than full-memory BFGS when $m = n$. However, our discussion here shows that the scheme is efficient for small m, which offers hope for certain limited-memory variants that one can imagine; see our discussion in §5.

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 Demonstrations

Our goal in this section is to provide additional numerical demonstrations of the application of `Agg-BFGS`. (Recall that a preview demonstration has been provided in Figure 1a.) In particular, our goal 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 also demonstrate that while numerical errors might accumulate 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 $W = I$, we computed the BFGS inverse Hessian approximation from $\{(s_k, y_k)\}_{k=0}^m$ and explicitly constructed the inverse Hessian approximation corresponding to the set of pairs when `Agg-BFGS` was used to aggregate the information into $\{(s_k, \tilde{y}_k)\}_{k=1}^m$. Figure 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 slightly larger as n increases and as m is closer to n, they remain accurate for all (n, m) relative to machine precision.

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. Figure 3 shows the errors for each iteration beyond $k = n$. As in Figure 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 to some extent, similar to the manner in which curvature information is ultimately over-written in full-memory BFGS.

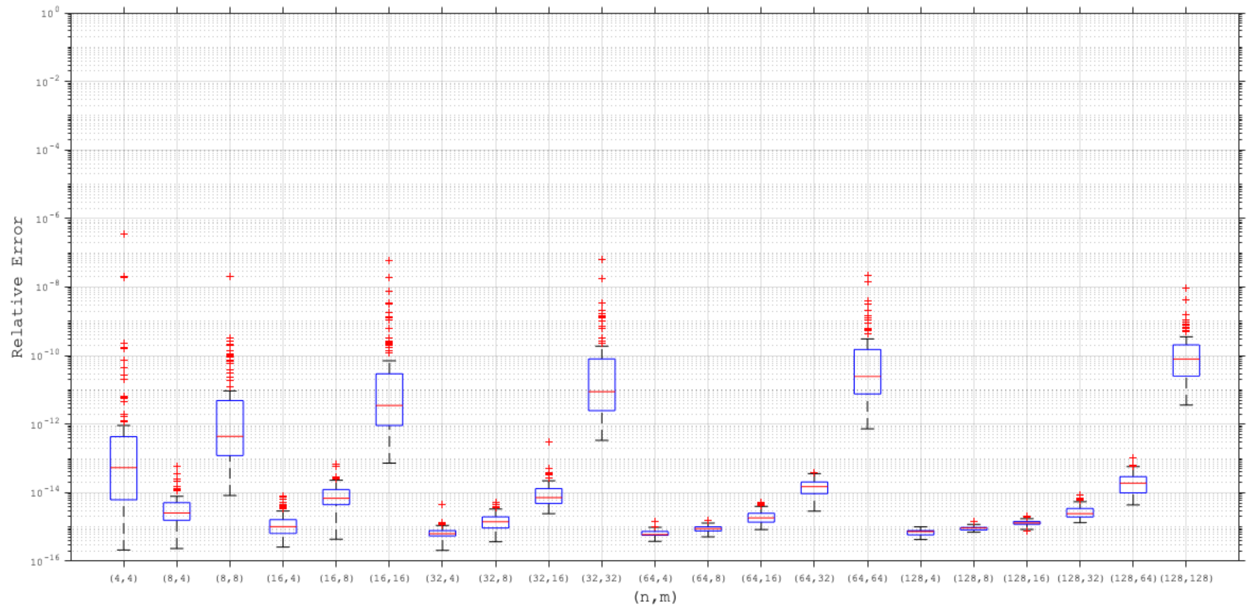


Figure 2: Relative errors of the differences between BFGS inverse Hessian approximations (computed from $\{(s_k, y_k)\}_{k=0}^m$) and the corresponding inverse Hessian approximations represented after applying `Agg-BFGS` to aggregate the information from a single curvature pair (into $\{(s_k, \tilde{y}_k)\}_{k=1}^m$). 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.

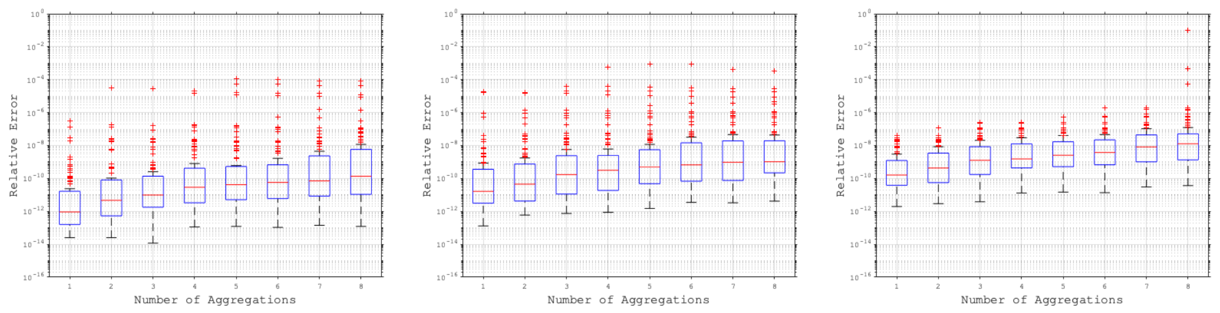
5 Conclusion

We have presented a technique for aggregating the curvature pair information in a limited-memory-type BFGS approach such that the corresponding (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 displacement vectors, 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 curvature information being lost. To the best of our knowledge, this is the first limited-memory-type approach 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 under certain conditions.

We close with a few remarks related to extensions of our methodology to other quasi-Newton schemes and for practical adaptations. With regard to the former issue, we observe that by the well-known symmetry between the BFGS and DFP updating, it is straightforward to extend our approach for DFP. In particular, in the context of 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 determine

$$\tilde{S}_{1:m} = M^{-1} Y_{1:m} \begin{bmatrix} A & 0 \end{bmatrix} + s_0 \begin{bmatrix} b \\ 0 \end{bmatrix}^T + S_{1:m} \tag{46}$$

(cf. (14)) such that $\mathrm{DFP}(M, S_{0:m}, Y_{0:m}) = \mathrm{DFP}(M, \tilde{S}_{1:m}, Y_{1:m})$. There might also be opportunities for extending our approach for the other members of the Broyden class of updates, although the extension in such cases is not as straightforward. Indeed, for members of the class besides BFGS and DFP, one likely needs to modify both iterate and gradient displacements in order to aggregate curvature information.



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

Figure 3: Accumulation of relative errors of the differences between BFGS inverse Hessian approximations and the corresponding inverse Hessian approximations represented 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.

As for designing practical adaptations of our scheme, we believe that there could be numerous opportunities. 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, 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 as well), 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 can also imagine that there could be numerous 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 potential practical benefits. Allow us also to note that 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 a block-coordinate descent algorithm.

References

[1] Albert S Berahas, Jorge Nocedal, and Martin Takác. A multi-batch L-BFGS method for machine learning. In *Advances in Neural Information Processing Systems*, pages 1055–1063, 2016.

[2] P. T. Boggs and R. H. Byrd. Adaptive, limited-memory BFGS algorithms for unconstrained optimization. Working Paper.

[3] J. F. Bonnans, J. Ch. Gilbert, C. Lemaréchal, and C. A. Sagastizábal. A family of variable metric proximal methods. *Mathematical Programming*, 68(1):15–47, 1995.

[4] C. G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and Its Applications*, 6(1):76–90, 1970.

[5] R. H. Byrd and J. Nocedal. A tool for the analysis of quasi-Newton methods with application to unconstrained minimization. *SIAM Journal on Numerical Analysis*, 26(3):727–739, 1989.

[6] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63:129–156, 1994.

[7] R. H. Byrd, J. Nocedal, and Y. Yuan. Global convergence of a class of quasi-Newton methods on convex problems. *SIAM Journal on Numerical Analysis*, 24(5):1171–1189, 1987.

[8] Richard H Byrd, Samantha L Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-Newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.

[9] F. E. Curtis. A self-correcting variable-metric algorithm for stochastic optimization. In *Proceedings of the 48th International Conference on Machine Learning*, volume 48, pages 632–641, New York, New York, USA, 2016. Proceedings of Machine Learning Research.

[10] F. E. Curtis and X. Que. An adaptive gradient sampling algorithm for nonsmooth optimization. *Optimization Methods and Software*, 28(6):1302–1324, 2013.

[11] F. E. Curtis and X. Que. A quasi-Newton algorithm for nonconvex, nonsmooth optimization with global convergence guarantees. *Mathematical Programming Computation*, 7:399–428, 2015.

[12] F. E. Curtis, D. P. Robinson, and B. Zhou. A self-correcting variable-metric algorithm framework for nonsmooth optimization. *IMA Journal of Numerical Analysis*, to appear, 2019.

[13] W. C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, 1991.

[14] J. E. Dennis and J. J. Moré. A characterization of superlinear convergence and its application to quasi-Newton methods. *Mathematics of Computation*, 28(126):549–560, 1974.

[15] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, USA, 1996.

[16] R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13(3):317–322, 1970.

[17] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders. Methods for modifying matrix factorizations. *Mathematics of Computation*, 126(28):505–535, 1974.

[18] D. Goldfarb. A family of variable metric updates derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970.

[19] Robert Gower, Donald Goldfarb, and Peter Richtárik. Stochastic block BFGS: Squeezing more curvature out of data. In *International Conference on Machine Learning*, pages 1869–1878, 2016.

[20] N. Haarala, K. Miettinen, and M. M. Mäkelä. New limited memory bundle method for large-scale nonsmooth optimization. *Optimization Methods and Software*, 19(6):673–692, 2004.

[21] Nitish Shirish Keskar and Albert S Berahas. ADAQN: An adaptive quasi-Newton algorithm for training RNNs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 1–16. Springer, 2016.

[22] A. S. Lewis and M. L. Overton. Nonsmooth optimization via quasi-Newton methods. *Mathematical Programming*, 141(1):135–163, 2013.

[23] R. Mifflin, D. Sun, and L. Qi. Quasi-Newton bundle-type methods for nondifferentiable convex optimization. *SIAM Journal on Optimization*, 8(2):583–603, 1998.

[24] Aryan Mokhtari and Alejandro Ribeiro. Global convergence of online limited memory BFGS. *The Journal of Machine Learning Research*, 16(1):3151–3181, 2015.

[25] J. Nocedal. Updating quasi-Newton matrices With limited storage. *Mathematics of Computation*, 35(151):773–782, 1980.

[26] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer New York, Second edition, 2006.

[27] J. D. Pearson. Variable metric methods of minimisation. *The Computer Journal*, 12(2):171–178, 1969.

[28] M. J. D. Powell. Some global convergence properties of a variable metric algorithm for minimization with exact line searches. In R. W. Cottle and C. E. Lemke, editors, *Nonlinear Programming, SIAM-AMS Proceedings, Vol. IX*, Harwell, England, 1976. American Mathematical Society.

[29] K. Ritter. Local and superlinear convergence of a class of variable metric methods. *Computing*, 23(3):287–297, 1979.

[30] K. Ritter. *Global and superlinear convergence of a class of variable metric methods*, pages 178–205. Springer Berlin Heidelberg, Berlin, Heidelberg, 1981.

[31] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

[32] Nicol N Schraudolph, Jin Yu, and Simon Günter. A stochastic quasi-Newton method for online convex optimization. In *Artificial Intelligence and Statistics*, pages 436–443, 2007.

[33] D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24(111):647–656, 1970.

[34] J. Vlček and L. Lukšan. Globally convergent variable metric method for nonconvex nondifferentiable unconstrained minimization. *Journal of Optimization Theory and Applications*, 111(2):407–430, 2001.

[35] Xiao Wang, Shiqian Ma, Donald Goldfarb, and Wei Liu. Stochastic quasi-Newton methods for nonconvex stochastic optimization. *SIAM Journal on Optimization*, 27(2):927–956, 2017.