Compact representations of structured BFGS matrices

Johannes J. Brust · Zichao (Wendy) Di · Sven Leyffer · Cosmin G. Petra

Received: 30 July 2020 / Accepted: 22 June 2021 / Published online: 9 July 2021
© This is a U.S. government work and not under copyright protection in the U.S.; foreign copyright protection may apply 2021

Abstract
For general large-scale optimization problems compact representations exist in which recursive quasi-Newton update formulas are represented as compact matrix factorizations. For problems in which the objective function contains additional structure, recent structured quasi-Newton methods exploit available second-derivative information and approximate unavailable second derivatives. This article develops the compact representations of two structured Broyden-Fletcher-Goldfarb-Shanno update formulas. The compact representations enable efficient limited memory and initialization strategies. Two limited memory line search algorithms are described for which extensive numerical results demonstrate the efficacy of the algorithms, including comparisons to IPOPT on large machine learning problems, and to L-BFGS on a real world large scale ptychographic imaging application.

Keywords Quasi-Newton method · Limited memory method · Large-scale optimization · Compact representation · BFGS method

Submitted to the editors July 2020. This material was based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research ASCR under Contract DE-AC02-06CH11347.

✉ Johannes J. Brust
jbrust@anl.gov

Zichao (Wendy) Di
wendydi@mcs.anl.gov

Sven Leyffer
leyffer@mcs.anl.gov

Cosmin G. Petra
petra1@llnl.gov

1 Argonne National Laboratory, Lemont, IL, USA

2 Lawrence Livermore National Laboratory, Livermore, CA, USA
1 Introduction

The unconstrained minimization problem is
\[
\min_{x \in \mathbb{R}^n} f(x),
\]
where \( f : \mathbb{R}^n \to \mathbb{R} \) is assumed to be twice continuously differentiable. If the Hessian matrix \( \nabla^2 f(x) \in \mathbb{R}^{n \times n} \) is unavailable, because it is unknown or difficult to compute, then quasi-Newton approaches are effective methods, which approximate properties of the Hessian at each iteration, \( \nabla^2 f(x_{k+1}) \approx B_{k+1} \) \([9]\). Arguably, the most widely used quasi-Newton matrix update is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) update \([4, 13, 16, 24]\), because of its desirable results on many problems. Given \( s_k \equiv x_{k+1} - x_k \) and \( y_k \equiv \nabla f(x_{k+1}) - \nabla f(x_k) \) the BFGS recursive update formula is
\[
B_{k+1} = B_k - \frac{1}{s_k^T B_k s_k} B_k s_k y_k^T + \frac{1}{y_k y_k^T} y_k^T y_k.
\]
For a symmetric positive definite initialization \( B_0 \in \mathbb{R}^{n \times n} \) \((2)\) generates symmetric positive definite matrices as long as \( s_k^T y_k > 0 \) for all \( k \geq 0 \) (see \([13, \text{Sect. 2}\])).

1.1 BFGS compact representation

Byrd et al. \([7]\) propose the compact representation of the recursive formula \((2)\). The compact representation has been successfully used for large-scale unconstrained and constrained optimization \([30]\). Let the sequence of pairs \( \{s_i, y_i\}_{i=0}^{k-1} \) be given, and let these vectors be collected in the matrices \( S_k = [s_0, \ldots, s_{k-1}] \in \mathbb{R}^{n \times k} \) and \( Y_k = [y_0, \ldots, y_{k-1}] \in \mathbb{R}^{n \times k} \). Moreover, let \( S_k^T Y_k = L_k + R_k \), where \( L_k \in \mathbb{R}^{k \times k} \) is the strictly lower triangular matrix, \( R_k \in \mathbb{R}^{k \times k} \) is the upper triangular matrix (including the diagonal), and \( D_k = \text{diag}(s_0^T y_0, \ldots, s_{k-1}^T y_{k-1}) \in \mathbb{R}^{k \times k} \) is the diagonal part of \( S_k^T Y_k \). The compact representation of the BFGS formula \((2)\) is \([7, \text{Theorem 2.3}]:\)
\[
B_k = B_0 - \begin{bmatrix} B_0 S_k & Y_k \end{bmatrix} \begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & D_k \end{bmatrix}^{-1} \begin{bmatrix} S_k^T B_0 \\ Y_k^T \end{bmatrix}.
\]
For large optimization problems limited memory versions of the compact representation in \((3)\) are used. The limited memory versions typically store only the last \( m \geq 0 \) pairs \( \{s_i, y_i\}_{i=k-m}^{k-1} \) when \( k \geq m \). In limited memory BFGS (L-BFGS) the dimensions of \( S_k \) and \( Y_k \) are consequently \( n \times m \). Usually the memory parameter is much smaller than the problem size, namely, \( m \ll n \). A typical range for this parameter is \( 5 \leq m \leq 50 \) (see Boggs and Byrd in \([2]\)). Moreover, in line search L-BFGS methods the initialization is frequently chosen as \( B_0 = \hat{\sigma}_k I_n \), where \( \hat{\sigma}_k = y_{k-1}^T y_{k-1} / s_{k-1}^T y_{k-1} \). Such an initialization enables efficient computations with the formula in \((3)\), and adds extra information through the parameter \( \hat{\sigma}_k \), which depends on the iteration \( k \).
1.2 Structured problems

When additional information about the structure of the objective function is known, it is desirable to include this information in a quasi-Newton update. Initial research efforts on structured quasi-Newton methods were in the context of nonlinear least squares problems. These include the work of Gill and Murray [15], Dennis et al. [10, 11], and Yabe and Takahashi [29]. Recently, Petra et al. [22] formulated the general structured minimization problem as

$$
\minimize_{x \in \mathbb{R}^n} f(x), \quad f(x) = \hat{k}(x) + \hat{u}(x),
$$

(4)

where $\hat{k} : \mathbb{R}^n \to \mathbb{R}$ has known gradients and known Hessians and $\hat{u} : \mathbb{R}^n \to \mathbb{R}$ has known gradients but unknown Hessians. For instance, objective functions composed of a general nonlinear function plus a regularizer or penalty term are described with (4). Thus, applications such as regularized logistic regressions [27] or optimal control problems contain structure that may be exploited, when we assume that the Hessian of the regularizer is known. We note that nonlinear least squares problems typically do not have the form as in (4), yet available second derivatives may also be used for this class of problems after reformulating the quasi-Newton vectors. We will describe an image reconstruction application in the numerical experiments, Sect. 4. Even though approximating the Hessian of the objective function in (4) by formula (2) or (3) is possible, this would not exploit the known parts of the Hessian. Therefore in [22] structured BFGS (S-BFGS) updates are derived, which combine known Hessian information with BFGS approximations for the unknown Hessian components. At each iteration the Hessian of the objective is approximated as $\nabla^2 f(x_{k+1}) \approx \nabla^2 \hat{k}(x_{k+1}) + A_{k+1}$, where $A_{k+1}$ approximates the unknown Hessian, that is, $A_{k+1} \approx \nabla^2 \hat{u}(x_{k+1})$. Given the known Hessian $\nabla^2 \hat{k}(x_{k+1}) \equiv K_{k+1}$ and the gradients of $\hat{u}$, let

$$
u_k \equiv K_{k+1}s_k + (\nabla \hat{u}(x_{k+1}) - \nabla \hat{u}(x_k)).
$$

(5)

One of two structured approximations from [22] is the structured BFGS-Minus (S-BFGS-M) update

$$
A^M_{k+1} = B^M_k - K_{k+1} - \frac{1}{s_k^T B^M_k s_k} B^M_k s_k s_k^T B^M_k + \frac{1}{s_k^T u_k} u_k u_k^T,
$$

(6)

where $B^M_k \equiv A^M_k + K_k$. By adding $K_{k+1}$ to both sides, the update from (6) implies a formula for $B^M_{k+1}$ that resembles (2), in which $B_{k+1}$, $B_k$, and $y_k$ are replaced by $B^M_{k+1}$, $B^M_k$, and $u_k$, respectively. Consequently, $B^M_{k+1}$ is symmetric positive definite given a symmetric positive definite initialization $B^M_0$ as long as $s_k^T u_k > 0$ for $k \geq 0$. A second formula is the structured BFGS-Plus (S-BFGS-P) update

$$
A^P_{k+1} = A^P_k - \frac{1}{s_k^T \hat{B}^P_k s_k} \hat{B}^P_k s_k \hat{B}^P_k + \frac{1}{s_k^T u_k} u_k u_k^T,
$$

(7)
where, for more readable notation, we define \( \hat{B}_k^p \equiv A_k^p + K_{k+1} \). After adding \( K_{k+1} \) to both sides in (7), the left hand side \( (B_{k+1}^p) = A_{k+1}^p + K_{k+1} \), which is important for computing search directions later on, is positive definite if \( B_k^p \) is positive definite and \( s^TU_k > 0 \). However, in general, \( B_{k+1} \) does not have to be positive definite, because \( B_k^p \) may not be positive definite when the known Hessian, \( K_{k+1} \), is not. Similar to (2) the update in (7) is also rank 2. Both of the updates in Eqs. (6) and (7) were implemented in a line search algorithm and compared with the BFGS formula (2) in [22]. The structured updates obtained better results in terms of iteration count and function evaluations than did the unstructured counterparts. Unlike the BFGS formula from (2), which recursively defines \( B_{k+1} \) as a rank-2 update to \( B_k \), the formulas for \( A_{k+1}^M \) and \( A_{k+1}^P \) in Eqs. (6), (7) additionally depend on the known Hessians \( K_{k+1} \) and \( K_k \). For this reason the compact representations of \( A_{k+1}^M \) and \( A_{k+1}^P \) are different from the one for \( B_{k+1} \) in (3) and have not yet been developed. The updates in (6) and (7) are dense in general, and hence neither are suitable for large-scale optimization. Hence, here, we develop first a compact representation of (6) and (7) and then show how to exploit them to develop structured limited-memory quasi-Newton updates.

1.3 Article contributions

In this article we develop the compact representations of the structured BFGS updates \( A_{k+1}^M \) and \( A_{k+1}^P \) from Eqs. (6) and (7) that lead to practical large-scale limited-memory implementations. Unwinding the update formula in (7) is challenging, however by using an induction technique we are able to derive the explicit expression of the compact representation. We propose the limited memory versions of the compact structured BFGS (L-S-BFGS) matrices and describe line search algorithms (with slightly modified Wolfe conditions) that implement them. We exploit the compact representations in order to compute search directions by means of the Sherman-Morrison-Woodbury formula and implement effective initialization strategies. Numerical experiments of the proposed L-S-BFGS methods on various problems are presented.

2 Compact representations of structured BFGS updates

To develop the compact representations of the structured BFGS formulas, we define

\[
U_k = \begin{bmatrix} u_0, & \ldots, & u_{k-1} \end{bmatrix}, \quad S_k^T U_k = L_k^U + R_k^U, \quad \text{diag}(S_k^T U_k) = D_k^U, \tag{8}
\]

where \( U_k \in \mathbb{R}^{n \times k} \) collects all \( u_k \) for \( k \geq 0 \) and where \( L_k^U \in \mathbb{R}^{k \times k} \) is a strictly lower triangular matrix, \( R_k^U \in \mathbb{R}^{k \times k} \) is an upper triangular matrix (including the diagonal), and \( D_k^U \in \mathbb{R}^{k \times k} \) is the diagonal part of \( S_k^T U_k \).

2.1 Compact representation of \( A_k^M \)

Theorem 1 contains the compact representation of \( A_k^M \).

Springer
Theorem 1 The compact representation of $A_k^M$ in the update formula (6) is

$$A_k^M = B_0^M - K_k - \left[ B_0^M S_k \ U_k \right] \left[ S_k^T B_0^M \ U_k \begin{pmatrix} L_k \ U_k \ L_k^T \ D_k \ U_k^T \end{pmatrix} - \mathbf{I} \right]^{-1} \left[ S_k^T (B_0^M)^T \ U_k^T \right],$$

(9)

where $S_k$ is as defined in (3), $U_k$, $L_k^T$, and $D_k^T$ are defined in (8), and $B_0^M = A_0^M + K_0$.

Proof Observe that by adding $K_{k+1}$ to both sides of (6) the update formula of $B_{k+1}^M$ becomes

$$B_{k+1}^M = B_k^M - \frac{1}{s_k} B_k^M s_k B_k^M + \frac{1}{s_k} u_k u_k^T.$$ This expression is the same as (2) when $B_{k+1}^M$ is relabeled as $B_{k+1}$, and therefore the compact representation of (2) is given by (3), and the compact representation of $B_k^M$ is given by (3) with $Y_k$ replaced by $U_k$ and $B_0$ replaced by $B_k$. Then (9) is obtained by subtracting $K_k$ from the compact representation of $B_k^M$, and noting that $B_0^M = A_0^M + K_0$. Since $B_k^M$ is symmetric positive definite as long as $B_0^M$ is symmetric positive definite and $s_k^2 u_k > 0$ for $k ≥ 0$, the inverse in the right-hand side of (9) is nonsingular as long as $B_0^M$ is symmetric positive definite and $s_k^2 u_k > 0$ for $k ≥ 0$.

Corollary 1 describes the compact representation of the inverse $H_k^M = (K_k + A_k^M)^{-1}$, which is used to compute search directions in a line search algorithm (e.g., $p_k^M = -H_k^M \nabla f(x_k)$).

Corollary 1 The inverse $H_k^M = (K_k + A_k^M)^{-1}$, with the compact representation of $A_k^M$ from (9), is given as

$$H_k^M = H_0^M + \left[ S_k \ H_0^M \ U_k \right] \left[ \begin{pmatrix} (T_k^U)^T (D_k^U & U_k T_k^U & -T_k^U) & 0_{k \times k} \end{pmatrix} \right] \left[ S_k^T (H_0^M)^T \right],$$

(10)

where $H_0^M = (B_0^M)^{-1} = (A_0^M + K_0)^{-1}$, and where $T_k^U = (R_k^U)^{-1}$ with $S_k$, $U_k$, $D_k^U$, and $R_k^U$ defined in Theorem 1 and (8).

Proof Define

$$\Xi_k \equiv \begin{bmatrix} B_0^M S_k & U_k \end{bmatrix}, \quad M_k \equiv \begin{bmatrix} S_k^T B_0^M S_k & L_k^T \ U_k^T \end{bmatrix},$$

so that for the compact representation of $A_k^M$ in (9) we have
\[ K_k + A_k^M = B_0^M - \Xi_k M_k^{-1} \Xi_k^T. \]

We will make use of the matrix \( M_k - \Xi_k^T (B_0^M)^{-1} \Xi_k \), which can be simplified based on the identity \( S_k^T U_k = L_k^U = R_k^U \). Specifically, the individual blocks of this matrix are

\[
M_k - \Xi_k^T (B_0^M)^{-1} \Xi_k = \begin{bmatrix} S_k^T B_k^M S_k & L_k^U \\ (L_k^U)^T & -D_k^U \end{bmatrix} - \begin{bmatrix} S_k^T B_k^M S_k & S_k^T U_k \\ U_k^T S_k & U_k^T (B_0^M)^{-1} U_k \end{bmatrix},
\]

or equivalently

\[
M_k - \Xi_k^T (B_0^M)^{-1} \Xi_k = \begin{bmatrix} 0 & -R_k^U \\ -(R_k^U)^T - D_k^U - U_k^T (B_0^M)^{-1} U_k \end{bmatrix}.
\]

Let \( H_k^M = (B_0^M)^{-1} \) then the expression of \( H_k^M \) is obtained by the Sherman-Morrison-Woodbury identity:

\[
\begin{aligned}
H_k^M &= (K_k + A_k^M)^{-1} \\
&= (B_0^M - \Xi_k M_k^{-1} \Xi_k^T)^{-1} \\
&= (B_0^M)^{-1} + (B_0^M)^{-1} \Xi_k [M_k - \Xi_k^T (B_0^M)^{-1} \Xi_k]^{-1} \Xi_k^T (B_0^M)^{-1} \\
&= H_0^M + H_0^M \Xi_k \begin{bmatrix} 0_{k \times k} R_k^U (R_k^U)^T D_k^U + U_k^T H_0^M U_k \end{bmatrix}^{-1} \Xi_k^T H_0^M \\
&= H_0^M + H_0^M \Xi_k \begin{bmatrix} (R_k^U)^T (D_k^U + U_k^T H_0^M U_k) (R_k^U)^{-1} - (R_k^U)^T \end{bmatrix} \Xi_k^T H_0^M
\end{aligned}
\]

where the third equality is obtained from applying the Sherman-Morrison-Woodbury inverse, the fourth equality is based on the individual blocks of \( M_k - \Xi_k^T (B_0^M)^{-1} \Xi_k \), and the fifth equality is obtained by explicitly computing the inverse of the \( 4k \times 4k \) block matrix. Finally, with \( (R_k^U)^{-1} = T_k^U \) and \( (B_0^M)^{-1} \Xi_k = (B_0^M)^{-1} [B_0^M S_k, U_k] \) the expression in (10) is recovered.

\[ \square \]

### 2.2 Compact representation of \( A_k^P \)

Developing the compact representation of (7) is more challenging and requires an inductive argument. Specifically, we define

\[ v_k \equiv K_{k+1} s_k \quad (11) \]

in addition to the expressions in (8) and

\[
V_k = \begin{bmatrix} v_0, \ldots, v_{k-1} \end{bmatrix}, \quad S_k^T V_k = L_k^V + R_k^V, \quad \text{diag}(S_k^T V_k) = D_k^V, \quad (12)
\]
where $V_k \in \mathbb{R}^{n \times k}$ collects all $v_k$ for $k \geq 0$ and where $L_k^V \in \mathbb{R}^{k \times k}$ is the strictly lower triangular matrix, $R_k^V \in \mathbb{R}^{k \times k}$ is the upper triangular matrix (including the diagonal), and $D_k^V \in \mathbb{R}^{k \times k}$ is the diagonal part of $S_k^TV_k$. Theorem 2 contains the compact representation of $A_k^P$.

**Theorem 2**  
The compact representation of $A_k^P$ in the update formula (7) is

$$A_k^P = A_0^P - \left[ Q_k \quad U_k \right] \left[ \begin{array}{cc} D_k^V + L_k^V + (L_k^V)^T + S_k^T A_0^P S_k & L_k^U - D_k \end{array} \right]^{-1} \left[ \begin{array}{c} Q_k^T \\ U_k^T \end{array} \right],$$  

where

$$Q_k \equiv V_k + A_0^P S_k,$$

and where $S_k, U_k, D_k^U$, and $L_k^U$ are defined in (2) and $V_k, L_k^V$, and $D_k^V$ are defined in (12).

**Proof**  
The proof of (13) is by induction. As such, the case $k = 1$ directly follows from (7). Next assume that (13) is valid for $k \geq 1$, and in particular let it be represented as

$$A_k^P = A_0^P - \left[ Q_k \quad U_k \right] \left( \begin{array}{cc} (M_k)_{11} & (M_k)_{12} \\ (M_k)_{12}^T & (M_k)_{22} \end{array} \right)^{-1} \left[ \begin{array}{c} Q_k^T \\ U_k^T \end{array} \right],$$  

where

$$(M_k)_{11} = D_k^V + L_k^V + (L_k^V)^T + S_k^T A_0^P S_k, \quad (M_k)_{12} = L_k^U, \quad (M_k)_{22} = -D_k^U.$$  

We verify the validity of (14) by substituting it in the update formula (7), and then seek the representation (14) for $k + 1$:

$$A_{k+1}^P = A_0^P - \left[ Q_{k+1} \quad U_{k+1} \right] \left( \begin{array}{cc} (M_{k+1})_{11} & (M_{k+1})_{12} \\ (M_{k+1})_{12}^T & (M_{k+1})_{22} \end{array} \right)^{-1} \left[ \begin{array}{c} Q_{k+1}^T \\ U_{k+1}^T \end{array} \right].$$

First let

$$q_k = v_k + A_0^P s_k, \quad w_k = Q_k^T s_k, \quad r_k = U_k^T s_k, \quad \xi_k = \begin{bmatrix} w_k \\ r_k \end{bmatrix}.$$  

Note that in (7) with the definition of $v_k$ from (11) it holds that
\[
A_k^P s_k + v_k = (A_k^P + K_{k+1}) s_k \\
= A_0^P s_k - \left[ \begin{array}{c}
Q_k \\
U_k
\end{array} \right] [M_k]^{-1} \left[ \begin{array}{c}
Q^T s_k \\
U^T s_k
\end{array} \right] + v_k \\
\equiv q_k - \left[ \begin{array}{c}
Q_k \\
U_k
\end{array} \right] [M_k]^{-1} \begin{bmatrix}
w_k \\
r_k
\end{bmatrix} \\
\equiv q_k - \left[ \begin{array}{c}
Q_k \\
U_k
\end{array} \right] [M_k]^{-1} \xi_k.
\]

Using the fourth equivalence in (17) the product \((A_k^P s_k + v_k)(A_k^P s_k + v_k)^T\) is

\[
(A_k^P s_k + v_k)(A_k^P s_k + v_k)^T = (q_k - \left[ \begin{array}{c}
Q_k \\
U_k
\end{array} \right] [M_k]^{-1} \xi_k) (q_k^T - \xi_k^T [M_k]^{-1} \left[ \begin{array}{c}
Q_k^T \\
U_k^T
\end{array} \right]).
\]

By multiplying all terms in (18) and grouping the columns as \([Q_{k+1} \ U_{k+1} \ q_k]\) we obtain the \(2k \times 2k\) block representation

\[
(A_k^P s_k + v_k)(A_k^P s_k + v_k)^T = \left[ \begin{array}{c}
Q_k \\
U_k \ q_k \\
\end{array} \right] \left[ \begin{array}{c}
M_k^{-1} \xi_k \xi_k^T M_k^{-1} - M_k^{-1} \xi_k \\
- \xi_k^T M_k^{-1} \\
1
\end{array} \right] \left[ \begin{array}{c}
Q_k^T \\
U_k^T \\
q_k^T
\end{array} \right].
\]

Next we define \(\sigma_k^P = 1/s_k^T (A_k^P + K_{k+1}) s_k\) and obtain the following representation of \(A_{k+1}^P\) using (14) in the second equality, and an explicit \(2k \times 2k\) block inverse in the third equality:

\[
A_{k+1}^P = A_k^P - \sigma_k^P (A_k^P s_k + v_k)(A_k^P s_k + v_k)^T + \frac{1}{s_k^T u_k} u_k u_k^T \\
= A_0^P - \sigma_k^P \left[ \begin{array}{c}
Q_k \\
U_k \ q_k \\
\end{array} \right] \left[ \begin{array}{c}
M_k^{-1} + M_k^{-1} \xi_k \xi_k^T M_k^{-1} - M_k^{-1} \xi_k \\
- \xi_k^T M_k^{-1} \\
1
\end{array} \right] \left[ \begin{array}{c}
Q_k^T \\
U_k^T \\
q_k^T
\end{array} \right] \\
+ \frac{1}{s_k^T u_k} u_k u_k^T \\
= A_0^P - \left[ \begin{array}{c}
Q_k \\
U_k \ q_k \\
\end{array} \right] \left[ \begin{array}{c}
M_k \\
\begin{bmatrix}
w_k \\
r_k
\end{bmatrix} \\
s_k^T q_k
\end{array} \right]^{-1} \left[ \begin{array}{c}
Q_k^T \\
U_k^T \\
q_k^T
\end{array} \right] + \frac{1}{s_k^T u_k} u_k u_k^T.
\]

Using the permutation matrix \(P = \left[ \begin{array}{c}
e_1 \\
\cdots \\
e_k \\
e_{k+1} \\
\cdots \\
e_{2k}
\end{array} \right] \) (which permutes the last column after the first \(k \times k\) block), we represent \(A_{k+1}^P\) as
\[ A_{k+1}^P = A_0^P - \left[ Q_k \ U_k \ q_k \right] PP^T \begin{bmatrix} M_k & w_k \\ w_k^T & r_k^T \end{bmatrix}^{-1} \begin{bmatrix} w_k \\ r_k^T \end{bmatrix} PP^T \begin{bmatrix} Q_k^T \\ q_k \end{bmatrix} \]

\[ + \frac{1}{s_k} u_k u_k^T \]

\[ = A_0^P - \left[ Q_k \ q_k \ U_k \ u_k \right] \]

Now we first verify that the identities hold:

\[ Q_{k+1} = \left[ Q_k \ q_k \right] = \left[ V_k + A_0^P s_k \ v_k + A_0^P s_k \right] = V_{k+1} + A_0^P s_{k+1}, \]

\[ U_{k+1} = \left[ U_k \ u_k \right]. \]

Next, note that \( L_k^U \) is the strictly lower triangular part of \( S_k^T V_k \) (i.e., \( L_k^U = \text{tril}(S_k^T U_k, -1) \)) so that \( r_k^T \) from (16) is

\[ r_k^T = s_k^T U_k = e_k^T \]

Similarly, since \( L_k^V \) is the strictly lower triangular part of \( S_k^T V_k \) and using \( Q_k = V_k + A_0^P s_k \) we deduce that \( w_k \) from (16) is

\[ w_k = Q_k^T s_k = (V_k^T + S_k^T A_0^P) s_k = (L_{k+1}^V)^T e_{k+1} + S_k^T A_0^P s_{k+1} e_{k+1}. \]

Now recall from (15) that \( (M_k)_{11} = D_k^V + L_k^V + (L_k^V)^T + S_k^T A_0^P s_k, \) \( (M_k)_{12} = L_k^U \) and \( (M_k)_{22} = D_k^U \), which enables identifying the individual blocks as

\[ (M_{k+1})_{11} = \begin{bmatrix} (M_k)_{11} & w_k \\ w_k^T & s_k^T q_k \end{bmatrix} = D_{k+1}^V + L_{k+1}^V + (L_{k+1}^V)^T + S_{k+1}^T A_0^P s_{k+1}, \]

\[ (M_{k+1})_{12} = \begin{bmatrix} (M_k)_{12} & 0 \\ 0 & r_k^T \end{bmatrix} = L_{k+1}^U, \]

\[ (M_{k+1})_{22} = \begin{bmatrix} (M_k)_{22} & 0 \\ 0 & -s_k^T u_k \end{bmatrix} = -D_{k+1}^U. \]

Therefore we conclude that \( A_{k+1}^P \) is of the form (13) with \( k + 1 \) replacing the indices \( k. \)

\[ \square \]

2.2.1 Sherman-Morrison-Woodbury inverse of \( K_k + A_k^P \)

In order to compute search directions, solves with the matrix \( B_k^P = K_k + A_k^P \) have to be done. Because of the structure of the compact representation of \( A_k^P, \) the Sherman-Morrison-Woodbury (SMW) inverse can be applied. However, the efficacy of the SMW formula largely depends on how efficiently solves with \( K_k + A_0^P \) can be done. Typically
\( \mathbf{A}_0^p \) is set as a multiple of the identity matrix. This enables two generic situations with advantageous structure of \( \mathbf{K}_k + \mathbf{A}_0^p \). First, if \( \mathbf{K}_k \) is diagonal or highly sparse then solves with the matrix \( \mathbf{K}_0^0 \equiv \mathbf{K}_k + \mathbf{A}_0^p \) are expected to be on the order of \( \mathcal{O}(nl) \) multiplications (for a constant \( l < n \)). Second, if \( \mathbf{K}_k \) is defined to be a low rank update itself, say \( \mathbf{K}_k = \mathbf{D}_k + \mathbf{K}_k^l (\mathbf{K}_k^l)^T \) where \( \mathbf{D}_k \) is diagonal and \( \mathbf{K}_k^l \in \mathbb{R}^{n \times d} \) (with constant \( l \)) then solves with \( \mathbf{K}_k + \mathbf{A}_0^p \) and subsequently \( \mathbf{K}_k + \mathbf{A}_0^p \) can be done efficiently by applying the SMW inverse twice. Note that the recursive update formula in Eq. (7) generates dense \( \mathbf{A}_k^p \) matrices, and consequently does not enable the exploitation of structure in \( \mathbf{K}_k \). Defining the \( 2k \times 2k \) middle matrix in Theorem 2 by \( \mathbf{M}_k \), the rectangular matrix \( [ \mathbf{Q}_k \mathbf{U}_k ] \equiv \mathbf{E}_k \) and \( \hat{\mathbf{K}}_0 = \mathbf{K}_k + \mathbf{A}_0^p \) then application of the SMW inverse yields

\[
(\mathbf{K}_k + \mathbf{A}_0^p)^{-1} = \hat{\mathbf{K}}_0^{-1} + \mathbf{K}_0^{-1} \mathbf{E}_k \left( \mathbf{M}_k - \mathbf{E}_k^T \hat{\mathbf{K}}_0^{-1} \mathbf{E}_k \right)^{-1} \mathbf{E}_k^T \hat{\mathbf{K}}_0^{-1}.
\]

A search direction can then be computed as \( \mathbf{p}_k = -(\mathbf{K}_k + \mathbf{A}_0^p)^{-1} \mathbf{g}_k \). Later, in Sect. 3.4.2, we describe further details on computing the inverse with the compact \( \mathbf{A}_k^p \) matrix.

### 2.3 Limited memory compact structured BFGS

The limited memory representations of Eqs. (9) and (13) are obtained by storing only the last \( m \geq 1 \) columns of \( \mathbf{S}_k \mathbf{U}_k \) and \( \mathbf{V}_k \). By setting \( m \ll n \) limited memory strategies enable computational efficiencies and lower storage requirements, see e.g., [20]. Updating \( \mathbf{S}_k \mathbf{U}_k \) and \( \mathbf{V}_k \) requires replacing or inserting one column at each iteration. Let an underline below a matrix represent the matrix with its first column removed. That is, \( \mathbf{S}_k \) represents \( \mathbf{S}_k \) without its first column. With this notation, a column update of a matrix, say \( \mathbf{S}_k \), by a vector \( \mathbf{s}_k \) is defined as follows.

\[
\text{colUpdate}(\mathbf{S}_k, \mathbf{s}_k) \equiv \begin{cases} 
\begin{bmatrix} \mathbf{S}_k & \mathbf{s}_k \end{bmatrix} & \text{if } k < m \\
\begin{bmatrix} \mathbf{S}_k & \mathbf{s}_k \end{bmatrix} & \text{if } k \geq m.
\end{cases}
\]

Such a column update either directly appends a column to a matrix or first removes a column and then appends one. This column update will be used, for instance, to obtain \( \mathbf{S}_{k+1} \) from \( \mathbf{S}_k \) and \( \mathbf{s}_k \), i.e., \( \mathbf{S}_{k+1} = \text{colUpdate}(\mathbf{S}_k, \mathbf{s}_k) \). Next, let an underline above a matrix represent the matrix with its first row removed. That is, \( \mathbf{S}_k^T \mathbf{U}_k \) represents \( \mathbf{S}_k^T \mathbf{U}_k \) without its first row. With this notation, a product update of, say \( \mathbf{S}_k^T \mathbf{U}_k \), by matrices \( \mathbf{S}_k \), \( \mathbf{U}_k \) and vectors \( \mathbf{s}_k, \mathbf{u}_k \) is defined as:

\[
\text{prodUpdate}(\mathbf{S}_k^T \mathbf{U}_k, \mathbf{S}_k, \mathbf{U}_k, \mathbf{s}_k, \mathbf{u}_k) \equiv \begin{cases} 
\begin{bmatrix} \mathbf{S}_k^T \mathbf{U}_k & \mathbf{S}_k^T \mathbf{u}_k \\
\mathbf{s}_k^T \mathbf{U}_k & \mathbf{s}_k^T \mathbf{u}_k \end{bmatrix} & \text{if } k < m \\
\begin{bmatrix} \mathbf{s}_k^T \mathbf{U}_k \\
\mathbf{s}_k^T \mathbf{u}_k \end{bmatrix} & \text{if } k \geq m.
\end{cases}
\]

This product update is used to compute matrix products, such as, \( \mathbf{S}_k^T \mathbf{U}_{k+1} \), with \( \mathcal{O}(2mn) \) multiplications, instead of \( \mathcal{O}(m^2n) \) when the product \( \mathbf{S}_k^T \mathbf{U}_k \) had previously been stored. Note that a diagonal matrix can be updated in this
way by setting the rectangular matrices (e.g., $S_k, U_k$) to zero, such that e.g., $D_{k+1}^U = \text{prodUpdate}(D_k^U, 0, 0, s_k, u_k)$. An upper triangular matrix can be updated in a similar way, e.g., $R_{k+1}^U = \text{prodUpdate}(R_k^U, S_k, 0, s_k, u_k)$. To save computations, products with zeros matrices are never formed explicitly. Section 3 discusses computational and memory aspects in greater detail.

3 Limited-memory structured BFGS line search algorithms

This section describes two line search algorithms with limited memory structured BFGS matrices. Specifically, the algorithm using the limited memory compact representation from Theorem 1 and Corollary 1 is called L-S-BFGS-M (corresponding to the minus update formula), while the algorithm using the limited memory compact representation from Theorem 2 is called L-S-BFGS-P (for the plus update formula). The compact representations enable efficient reinitialization strategies and search directions, and we discuss these two components first, before presenting the overall algorithms.

3.1 Initializations

For the limited memory BFGS matrix based on (3) one commonly uses the initializations

$$B_0^{(k)} = \hat{\sigma}_k I_n, \quad \text{where} \quad \hat{\sigma}_k = y_{k-1}^T y_{k-1} / s_{k-1}^T s_{k-1}, \quad (19)$$

(c.f. [7]). Choosing the initialization as a multiple of the identity matrix enables fast computations with the matrix in (3). In particular, the inverse of this matrix may be computed efficiently by the Sherman-Morrison-Woodbury identity. Because at the outset it is not necessarily obvious which initializations to use for the limited memory structured-BFGS (L-S-BFGS) matrices based on Eqs. (9) and (13), we investigate different approaches. We use the analysis in [1], which proposed formula (19). Additionally, in that work a second initialization $\hat{\sigma}_k^{(2)} = s_{k-1}^T y_{k-1} / s_{k-1}^T s_{k-1}$ was proposed. Because in the S-BFGS methods the vectors $\hat{u}_k$ and $\hat{u}_k$ are used instead of $y_k$ (unstructured BFGS), the initializations in this article are the below.

$$\sigma_{k+1} = \begin{cases} \frac{u_k^T u_k}{s_k^T u_k} & \text{Init. 1} \\ \frac{s_k^T \hat{u}_k}{s_k^T s_k} & \text{Init. 2} \\ \frac{\hat{u}_k^T u_k}{s_k^T s_k} & \text{Init. 3} \\ \frac{s_k^T \hat{u}_k}{s_k^T s_k} & \text{Init. 4} \end{cases} \quad (20)$$

Note that Init. 1 and Init. 2 are extensions of $\hat{\sigma}_k$ to structured methods. Instead of using $y_k$ these initializations are defined by $\hat{u}_k$ and $\hat{u}_k$. Init. 3 and Init. 4 extend $\hat{\sigma}_k^{(2)}$. Observe that the vectors $\hat{u}_k = \nabla \hat{u}(x_{k+1}) - \nabla \hat{u}(x_k)$ depend only on gradient information of $\hat{u}(x)$. In contrast, $u_k = K_{k+1} s_k + \hat{u}_k$ depends on known second-derivative
information, too. Because the initial matrices $A_0^M$ and $A_0^P$ affect the compact representations from Theorems 1 and 2 differently, we accordingly adjust our initialization strategies for these two matrices. In particular, for L-S-BFGS-M the compact limited memory formula for $B_k^M$ simplifies if we take $B_0^M$ as a multiple of the identity matrix:

$$B_0^M = A_0^M + K_0 \equiv \sigma_k I.$$  \hspace{1cm} (21)

The advantage of this choice is that it has similar computational complexities to the L-BFGS formula from (3). However by setting this default initialization for $B_0^M$ the corresponding limited memory matrices $B_k^M$ are not equivalent anymore to the full-memory matrices $B_k^M$ defined by (6), even when $k < m$. In Sect. 3.4.1 computational techniques are discussed when $B_0^M$ is not taken as a multiple of the identity matrix. For L-S-BFGS-P we set $P_0^k = \sum_k$. This initialization, as long as $\sum_k$ remains constant, implies that the limited memory compact representations from Theorem 1 and the update formulas from (7) produce the same matrices when $k < m$.

3.2 Search directions

The search directions for line search algorithms, with the structured BFGS approximations, are computed as

$$p_k = -(K_k + A_k)^{-1}g_k, \hspace{1cm} (22)$$

where $g_k = \nabla f(x_k)$ and where $A_k$ is either the limited memory version of $A_k^M$ from (9) or $A_k^P$ from (13). When $A_k^M$ is used, we apply the expression of the inverse from Corollary 1, in order to compute search directions. In particular, with the initialization strategy $B_0^M = \sigma_k I$ from the preceding section, the search directions (22) are computed efficiently by

$$p_k^M = -\frac{g_k}{\sigma_k} - [S_k U_k] \begin{bmatrix} (T_k^U)^T (D_k^U + 1/\sigma_k U_k^T U_k) T_k^U - \frac{(T_k^U)^T}{\sigma_k} & 0_{m \times m} \end{bmatrix} \begin{bmatrix} [S_k g_k] \\ [U_k g_k] \end{bmatrix},\hspace{1cm} (23)$$

where $T_k^U$ is defined in Corollary 1. This computation is done efficiently assuming that all matrices have been updated before, such as $U_k^T U_k$. Omitting terms of order $m$, the multiplication complexity for this search direction is $O(n(4m + 1) + 3m^2)$. In particular, computing $p_k^M$ can be done by: two vector multiplies with the $n \times 2m$ matrix $[S_k U_k]$ (order $4nm$), the scaling $\frac{g_k}{\sigma_k}$ (order $n$) and a matrix vector product with a structured $2m \times 2m$ matrix. Since $T_k^U$ represents a solve with an $m \times m$ upper triangular matrix the vector product with the middle $2m \times 2m$ matrix is done in order $3m^2$. When $A_k^P$ is used, search directions are computed by solves of the linear system $(K_k + A_k^P)p_k^P = -g_k$. Because of the compact representation of $A_k^P$ we can exploit structure in solving this system, as is described in Sect. 3.4.2.
3.3 Algorithms

Similar to Petra et al. [22], we use a strong Wolfe line search in our implementations of the new limited-memory compact representations.

For nonnegative constants \(0 < c_1 \leq c_2\), the current iterate \(x_k\) and search direction \(p_k\), the strong Wolfe conditions define the step length parameter \(\alpha\) by two inequalities

\[
\begin{align*}
    f(x_k + \alpha p_k) &\leq f(x_k) + c_1 \alpha (p_k^T \nabla f(x_k)), \quad \text{and} \\
    \left| p_k^T \nabla f(x_k + \alpha p_k) \right| &\leq c_2 \left| p_k^T \nabla f(x_k) \right|.
\end{align*}
\]

Because the S-BFGS-M matrix from (9) is positive definite as long as \(s_k^T u_k > 0\) for \(k \geq 0\) (rather than \(s_k^T y_k > 0\) for \(k \geq 0\) for L-BFGS), the line searches in our algorithms include this condition. As in [22, "Appendix A"] a variant of the Moré-Thuente [19] line search is used. This line search is identical to the one of Moré-Thuente, except for one condition. Specifically, given a trial step length \(\alpha_t\) and trial \(x u_t\), our line search terminates when the conditions in (24) and additionally \(s_k^T x u_t > 0\) holds. [22, Proposition 17] ensures the existence of a step length \(\alpha\) that satisfies all of the above conditions. Such a line search variant is straightforward to implement, by adding one additional condition to a Moré-Thuente line search. Moreover, when S-BFGS-M is used, new search directions are computed by using the inverse from Corollary 1. In contrast, because the S-BFGS-P matrix from (13) is not necessarily positive definite even if \(s_k^T u_k > 0\) for \(k \geq 0\) (see [22]), our implementation checks whether \(K_k + A_k^P\) is positive definite, before computing a new search direction. However, if it is known that \(K_k\) is positive definite for all \(k \geq 0\) (which is often the case in applications), then ensuring that \(s_k^T u_k > 0\) for \(k \geq 0\) ensure positive definiteness, in this case too. If this matrix is positive definite, then a new search direction is computed by solving the linear system \((K_k + A_k^P) p_k^P = -g_k\). Otherwise the search direction is computed by solving the system \((K_k + A_k^P + \delta I_n) p_k^P = -g_k\), where the scalar \(\delta > 0\) ensures that \((K_k + A_k^P + \delta I_n) > 0\) (Here \(\delta\) is chosen as the the first \(\delta = 10^j, j = 0, 1, \ldots\) that yields a positive definite matrix). The proposed limited memory line search algorithms are listed in Algorithm 1 and Algorithm 2.
Algorithm 1 Limited Memory Structured-BFGS-Minus (L-S-BFGS-M)

1: Initialize: $k = 0$, $m > 0$, $\epsilon > 0$, $\sigma_k > 0$, $0 < c_1 \leq c_2$, $x_k$, $g_k = \nabla f(x_k) = \nabla \hat{k}(x_k) + \nabla \hat{u}(x_k)$, $S_k = 0$, $U_k = 0$, $D_k^U = 0$, $(R_k^U)^{-1} = 0$, $U_k^T U_k = 0$, $H_0 = (1/\sigma_k)I$, $\Xi_k = [S_k H_0 U_k]$

2: while $\|g_k\|_\infty > \epsilon$

3: Compute:

$$
p_k = -H_0 g_k + \Xi_k (M_k (\Xi_k^T g_k)),
$$

where

$$
M_k = \begin{bmatrix}
(R_k^U)^{-T} (D_k^U + U_k^T H_0 U_k) (R_k^U)^{-1} & -(R_k^U)^{-T} \\
-(R_k^U)^{-1} & 0
\end{bmatrix}.
$$

4: Strong Wolfe line search:

$$
x_{k+1} = x_k + \alpha p_k,
$$

where $\alpha > 0$, $x_{k+1}$ satisfies strong Wolfe conditions (cf. [22] and (24)), $s_k = x_{k+1} - x_k$, $s_k^T u_k > 0$.

5: Updates: $g_{k+1} = \nabla f(x_{k+1})$, $u_k = \nabla^2 \hat{k}(x_{k+1}) s_k + (\nabla \hat{u}(x_{k+1}) - \nabla \hat{u}(x_k))$

6: $S_{k+1} = \text{colUpdate}(S_k, s_k)$

7: $U_{k+1} = \text{colUpdate}(U_k, u_k)$

8: $R_{k+1} = \text{prodUpdate}(R_k^U, S_k, 0, s_k, u_k)$

9: $U_{k+1}^T U_{k+1} = \text{prodUpdate}(U_k^T U_k, U_k, U_k, u_k, u_k)$

10: $D_{k+1}^U = \text{prodUpdate}(D_k^U, 0, 0, s_k, u_k)$

11: Compute: $\sigma_{k+1}$

12: $H_0 = (1/\sigma_{k+1})I$, update $M_{k+1}$, $\Xi_{k+1}$ using Theorem 1, $k = k + 1$

13: end while

14: return $x_k$

Note that $\Xi_k^T g_k$ on Line 3 in Algorithm 1 is computed as $\begin{bmatrix} S_k^T g_k \\ H_0 U_k^T g_k \end{bmatrix}$ so that only one linear solve with $H_0 = (K_0 + A_0^M)^{-1}$ is needed, when the algorithm does not use a multiple of the identity as the initialization.
Compact representations of structured BFGS matrices

Algorithm 2 Limited Memory Structured-BFGS-Plus (L-S-BFGS-P)

1: Initialize: $k = 0$, $m > 0$, $l > 0$, $\epsilon > 0$, $\sigma_k > 0$, $0 < c_1 < c_2$, $x_k$, $g_k = \nabla f(x_k) = \nabla k(x_k) + \nabla u(x_k)$, $K_k = V^2 k(x_k)$, $S_k = 0$, $U_k = 0$, $V_k = 0$, $D^U_k = 0$, $L^U_k = 0$, $D^V_k = 0$, $L^V_k = 0$, $\Xi_k = 0$, $S^T_k S_k = 0$, $M_k = 0$, $A_k = \sigma_k I$

2: if $\text{nnz}(K_k) > nl$ and $K_k \neq \text{(Diagonal + Low Rank)}$ then

3: print Warning: ‘‘L-S-BFGS-P is most effective for sparse or structured $K”$

4: print ‘‘Consider using L-S-BFGS-M (Algorithm 1) or L-BFGS”

5: end if

6: while $\|g_k\|_\infty > \epsilon$ do

7: if $(K_k + A_k) \neq 0$ {Comment: // Only if not known that $K_k \geq 0$} then

8: Find $\delta > 0$ such that $(K_k + A_k + \delta I_n) > 0$; Update $K_k \leftarrow K_k + \delta I_n$

9: end if

10: Solve $(K_k + A_k)p_k = -g_k$ (for structured $K_k$ (e.g., diagonal, sparse, diagonal + low rank) use the Sherman-Morrison-Woodbury inverse in Sec. 3.4.2):

$$p_k = -g_k - \Xi_k \left( (M_k - \Xi_k^T (K_k + \sigma_k I_n)^{-1} \Xi_k) \right)^{-1} (\Xi_k^T g_k),$$

11: Strong Wolfe line search:

$$x_{k+1} = x_k + \alpha p_k,$$

where $\alpha > 0$, $x_{k+1}$ satisfies strong Wolfe conditions (cf. [22] and (24)), $s_k = x_{k+1} - x_k$.

12: Updates: $\mathbf{g}_k^{k+1} = \nabla f(x_{k+1})$, $\mathbf{K}_k^{k+1} = \nabla^2 f(x_{k+1})$, $\mathbf{v}_k = K_k^{k+1} s_k$, $u_k = v_k + (\nabla u(x_{k+1}) - \nabla u(x_k))$

13: $S_{k+1} = \text{colUpdate}(S_k, s_k)$

14: $U_{k+1} = \text{colUpdate}(U_k, u_k)$

15: $V_{k+1} = \text{colUpdate}(V_k, v_k)$

16: $L^U_{k+1} = \text{prodUpdate}(L^U_k, 0, U_k, s_k, 0)$

17: $L^V_{k+1} = \text{prodUpdate}(L^V_k, 0, V_k, s_k, 0)$

18: $S^T_{k+1} S_{k+1} = \text{prodUpdate}(S^T_k S_k, S_k, S_k, s_k, s_k)$

19: $D^U_{k+1} = \text{prodUpdate}(D^U_k, 0, 0, s_k, u_k)$

20: $D^V_{k+1} = \text{prodUpdate}(D^V_k, 0, 0, s_k, v_k)$

21: Compute: $\sigma_{k+1}$

22: $A_0 = \sigma_{k+1} I$, update $\Xi_{k+1} = [V_{k+1} + A_0 S_{k+1} U_{k+1}]$

23: $M_{k+1} = \left[ D^U_{k+1} \right]^{-1}$

$\{\text{Comment: // } A_{k+1} = A_0 - \Xi_{k+1} M_{k+1} \Xi_{k+1}^T \}$

24: $k = k + 1$

25: end while

26: return $x_k$

Algorithm 2 is expected to be computationally more expensive than Algorithm 1 because it tests for the positive definiteness of $K_k + A_k$ in Line 7 and it computes search directions by the solve in Line 10. However, the structured quasi-Newton approximation in Algorithm 2 may be a more accurate approximation of the true Hessian (see [22]), which may result in fewer iterations or better convergence properties. Note that as in [22, Sect. 3.1.2] computational efforts for ensuring positive definiteness may largely be reduced by e.g., defining
\[ \delta = \max(0, (\epsilon - (u_k + v_k)^T s_k)/\|s_k\|^2) \], for \( 0 < \epsilon \). Unlike Algorithm 2, Algorithm 1 does not require solves involving large linear systems.

### 3.4 Large-scale computation considerations

This section discusses computational complexity and memory requirements of the structured Hessian approximations when the problems are large. In particular, if \( n \) is large the Hessian matrices \( \mathbf{K}_k \) typically exhibit additional structure, such as being diagonal or sparse. When \( \mathbf{K}_k \) is sparse and solves with it can be done efficiently, the compact representation of \( \mathbf{A}_k^M \) and \( \mathbf{A}_k^P \) can be exploited to compute inverses of \( \mathbf{K}_k + \mathbf{A}_k \) efficiently. Note that Algorithm 1 is directly applicable to large problems, because the formula in (23) does not use solves with \( \mathbf{K}_k \). Nevertheless, observe that the matrices \( \mathbf{K}_k + \mathbf{A}_k \), (with limited memory \( \mathbf{A}_k \) from Theorem 1 or Theorem 2, respectively), have the form with \( m \ll n \):

\[
\mathbf{K}_k + \mathbf{A}_k \equiv \hat{\mathbf{K}}_0 - \begin{bmatrix} \Xi_k \end{bmatrix} \left[ \mathbf{M}_k \right]^{-1} \begin{bmatrix} \Xi_k^T \end{bmatrix}, \tag{25}
\]

for some \( \hat{\mathbf{K}}_0 \). If \( \mathbf{A}_k^M \) is used in (25) then \( \hat{\mathbf{K}}_0 = \mathbf{K}_0 + \mathbf{A}_0^M \) and \( \Xi_k, \mathbf{M}_k \) correspond to the remaining terms in Theorem 1. Using \( \mathbf{A}_k^P \) in (25) then \( \hat{\mathbf{K}}_0 = \mathbf{K}_k + \mathbf{A}_0^P \) and \( \Xi_k, \mathbf{M}_k \) correspond to the remaining terms in Theorem 2. Because of its structure the matrix in (25) can be inverted efficiently by the Sherman-Morrison-Woodbury formula as long as solves with \( \hat{\mathbf{K}}_0 \) can be done efficiently. Next, L-S-BFGS-M and L-S-BFGS-P are discussed in the situation when solves with \( \hat{\mathbf{K}}_0 \) are done efficiently. Afterwards we relate these methods to S-BFGS-M, S-BFGS-P and BFGS, L-BFGS.

#### 3.4.1 Computations for L-S-BFGS-M

The most efficient computations are achieved when \( \hat{\mathbf{K}}_0 \) is set as a multiple of the identity matrix \( \sigma_k \mathbf{I} \) (cf. (3.2) with \( \mathcal{O}(n(4m + 1) + 3m^2) \) multiplications). This approach however omits the \( \mathbf{K}_0 \) term. Nevertheless, when \( \mathbf{K}_0 \) has additional structure such that factorizations and/or solves with it can be done in, say \( nl \) multiplications (e.g., a diagonal or highly sparse matrix), search directions can be computed efficiently in this case, without omitting \( \mathbf{K}_0 \). In particular, the search direction is computed as \( \mathbf{p}_k^M = -(\mathbf{K}_k + \mathbf{A}_k^M)^{-1}\mathbf{g}_k = -\mathbf{H}_k^M \mathbf{g}_k \) where \( \mathbf{H}_k^M \) is the inverse from (1). The initialization matrix is \( \mathbf{H}_0^M = (\sigma_k \mathbf{I} + \mathbf{K}_0)^{-1} \). To determine the search direction two matrix vector products with the \( n \times 2m \) matrices \( \{ \mathbf{S}_k, \mathbf{H}_0^M \mathbf{U}_k \} \) are required, at complexity \( \mathcal{O}(4nm + 2nl) \). The product with the \( 2m \times 2m \) middle matrix is done at \( \mathcal{O}(2mn + nl + 2m^2) \). Subsequently, \( -\mathbf{H}_0^M \mathbf{g}_k \) is obtained at \( nl \) multiplications. The total complexity is thus \( \mathcal{O}(n(6m + 4l) + 2m^2) \). Note that if \( \sigma_k \) is set to a constant value, say \( \sigma_k = \bar{\sigma} \), then the complexity can be further reduced by storing the matrix.
$\hat{U}_k = [\hat{u}_{k-m} \ldots \hat{u}_{k-1}]$, where $\hat{u}_i = (K_0 + \delta I)^{-1} u_i$. The computational cost in this situation is $O(n(4m + l) + 3m^3)$, excluding the updating cost of the vector $\hat{u}_i$ at order $nl$. With a constant $\sigma_k$ only one factorization of $(K_0 + \delta I)$ is required.

### 3.4.2 Computations for L-S-BFGS-P

When $A^p_k$ is used in (25) with $\hat{K}_0 = (K_k + A^p_k)$ (after possibly updating $K_k \leftarrow K_k + \delta I$) and setting $\hat{Q}_k = \hat{K}_0^{-1} Q_k$, $\hat{U}_k = \hat{K}_0^{-1} U_k$ the inverse has the form

$$(K_k + A^p_k)^{-1} = \hat{K}_0^{-1} \left( I_n + \Xi_k \left( M_k - \Xi_k^T \hat{K}_0^{-1} \Xi_k \right)^{-1} \Xi_k^T \hat{K}_0^{-1} \right),$$

where $\Xi_k^T \hat{K}_0^{-1} \Xi_k = \left[ Q^T_k \hat{Q}_k \ Q^T_k \hat{U}_k \ U^T_k \hat{Q}_k \ U^T_k \hat{U}_k \right]$ and $\Xi_k$, $M_k$ are defined in (2). Assuming that $M_k$, $Q_k$, $U_k$ had previously been updated, computing the search direction $p_k^p = -(K_k + A^p_k)^{-1} g_k$ may be done as follows; First, $\hat{Q}_k$, $\hat{U}_k$ are computed in $O(2nlm)$ multiplications. Then the $2m \times 2m$ matrix $\Xi_k^T \hat{K}_0^{-1} \Xi_k$ is formed in $O(3nm^2)$ multiplications. Combining the former terms and solving with the (small) $2m \times 2m$ matrix explicitly, the direction $p_k^p$ is computed in $O((2nl + 3m^2 + 4m + 1) + m^3)$ multiplications. Note that this approach requires an additional $2nm$ storage locations for the matrices $\hat{Q}_k$, $\hat{U}_k$. Two additional remarks; first, since $Q_k = V_k + A^p_0 S_k$, the update of $Q_k$ uses $O(nl)$ multiplications to form a new $x\nu_k$ and additional $nm$ multiplications if $A^p_0 = \sigma_k I$. If $\sigma_k$ remains constant, say $\sigma_k = \tilde{\sigma}$, then the update of $Q_k$ is done at only $O(nl)$ multiplications, because $A^p_0 S_k$ does not need to be recomputed each iteration. Second, if $K_k = K_0$, in other words if $K_k$ is a constant matrix then Theorems 1 and 2 reduce to the same expressions yielding the same computational complexities.

### 3.4.3 Memory usage and comparison

This section addresses the memory usage of the proposed representations and relates their computational complexities to existing methods. As an overall guideline, the representations from (25) use $2nm + 4m^2$ storage locations, excluding the $\hat{K}_0$ term. This estimate is refined if the particular structure of the matrix $M_k$ is taken into consideration. For example, the matrices $T^U_k$ and $D^U_k$ from Theorem 1 are upper triangular and diagonal, respectively. Thus, when $H^M_0 = \sigma_k I$, and when the matrix $U^T_k U_k \in \mathbb{R}^{m \times m}$ is stored and updated, the memory requirement for the limited memory version of $H^M_k$ in (1) are $O(2nm + \frac{3}{2}m^2 + m)$ locations. We summarize the computational demands of the different methods in a table.

Note that when $m \ll n$ and $l \ll n$ L-BFGS, L-S-BFGS-M and L-S-BFGS-P enable computations with complexity lower than $n^2$ and therefore allow for large values of $n$. Moreover, Table 1 shows that the proposed limited-memory BFGS methods have similar search direction complexity to unstructured L-BFGS, but higher update cost.
4 Numerical experiments

This section describes the numerical experiments for the proposed methods in Sect. 3. The numerical experiments are carried out in MATLAB 2016a on a MacBook Pro @2.6 GHz Intel Core i7, with 32 GB of memory. The experiments are divided into five parts. In Experiment I, we investigate initialization strategies. Experiment II compares the limited memory methods with the full-memory methods. The tests in this experiment are on the same 61 CUTEst [17] problems as in [22], unless otherwise noted. In Experiment III, we use classification data from LIBSVM (a library for support vector machines [8]) in order to solve regularized logistic regression problems with the proposed methods. We also include an application to PDE constrained optimization. In Experiment IV, the proposed methods and L-BFGS with the IPOPT [28] solver are compared. Experiment V, describes a real world application from image reconstruction.

Extended performance profiles as in [18] are provided. These profiles are an extension of the well known profiles of Dolan and Moré [12]. We compare the number of iterations and the total computational time for each solver on the test set of problems, unless otherwise stated. The performance metric $\rho_s(\tau)$ with a given number of test problems $n_p$ is

$$\rho_s(\tau) = \frac{\text{card} \{ p : \pi_{p,s} \leq \tau \}}{n_p} \quad \text{and} \quad \pi_{p,s} = \min_{1 \leq i \leq S, i \neq s} t_{p,i},$$

where $t_{p,s}$ is the “output” (i.e., iterations or time) of “solver” $s$ on problem $p$. Here $S$ denotes the total number of solvers for a given comparison. This metric measures the proportion of how close a given solver is to the best result. The extended performance profiles are the same as the classical ones for $\tau \geq 1$. In the profiles we include...
Compact representations of structured BFGS matrices

4.1 Experiment I

This experiment investigates the initialization strategies from Sect. 3. To this end, the problems in this experiment are not meant to be overly challenging, yet they are meant to enable some variations. Therefore, we define the quadratic functions

\[ Q_i(x; \phi, r) \equiv \frac{1}{2} x^T (\phi \cdot I + Q_i D_i Q_i^T) x, \]

with scalar parameters \( 0 < \phi, 1 \leq r \leq n \) and where \( D_i \in \mathbb{R}^{r \times r} \) is a diagonal matrix and \( Q_i \in \mathbb{R}^{n \times r} \) has orthonormal gaussian columns. Note that \( r \) eigenvalues of the Hessian \( \nabla^2 Q_i \) are the diagonal elements of \( \phi \cdot I + D_i \), while the remaining \( (n - r) \) eigenvalues are \( \phi \). Therefore, by varying \( \phi, r \), and the elements of \( D_i \), Hessian matrices with different spectral properties are formed. In particular, when \( r \ll n \), the eigenvalues are clustered around \( \phi \). In the experiments of this section we investigate \( \phi = 1 \). In "Appendix A", we include tests when \( \phi = 1000 \). The structured objective functions from (4) are defined by

\[ \hat{k}(x) = x^T g + Q_1(x; \phi, r), \quad \hat{u}(x) = Q_2(x; \phi, r). \quad (26) \]

We refer to the objective functions \( f(x) = \hat{k}(x) + \hat{u}(x) \) defined by (26) as structured quadratics. The problems in this experiment have dimensions \( n = j \cdot 100 \) with corresponding \( r = j \cdot 10 \) for \( 1 \leq j \leq 7 \). Since some of the problem data in this experiment is randomly generated (e.g., the orthonormal matrices \( xQ_i \)), the experiments are repeated five times for each \( n \). The reported results are of the average values of the five individual runs. For all solvers we set \( m = 8 \) (memory parameter), \( \epsilon = 5 \times 10^{-6} \) \( (\|g\|_\infty \leq \epsilon) \), and maximum iterations to 10,000 This limit was not reached in the experiments.

4.1.1 Experiment I.A: L-S-BFGS-M

Experiment I.A compares the four L-S-BFGS-M initializations on the structured quadratic objective functions with eigenvalues clustered around 1. In particular, \( \phi = 1 \), and the elements of \( D_i \) are uniformly distributed in the interval \([0, 999]\). The results are displayed in Fig. 1.

We observe that in terms of number of iterations, Init. 4 (red) and Init. 3 (purple) perform similarly and that also Init. 2 (green) and Init. 1 (blue) perform similarly. Overall, Init. 4 and Init. 3 require fewer iterations on the structured quadratics. Moreover, the solid lines are above the dashed ones for both pairs. This indicates that including only gradient information in \( \hat{u}_k \) and in the initialization strategy, as opposed to also including 2nd derivative information from \( u_k \), may be desirable for this problem. Init. 1 and Init. 2 are fastest on these problems. Even though these initializations require a larger number of iterations, they can be faster because the line searches terminate more quickly.
Next, we compare the four L-S-BFGS-P initializations. As before, experiments on problems with eigenvalues clustered at 1 are done. Experiments with eigenvalues clustered around 1000 are included in Appendix A. The respective outcomes are in Fig. 2.

We observe that, similar to Fig. 1, Init. 3 and Init. 4 do best in iterations, while Init. 1 does best in time.

To analyze the properties of the scaling factor $\sigma_k$ in greater detail, Sect. 4.1.2 describes experiments that relate $\sigma_k$ to eigenvalues.

### 4.1.2 Experiment I.B: properties of $\sigma_k$ as scaling factor

It has been observed (for instance, cf. [5, 21]) that the initialization $\hat{\sigma}_k$ from (19) for the L-BFGS matrix can be viewed as a spectral estimate of $\nabla^2 f(x_k)$. The role as a scaling factor of such a scalar initialization can have profound ramifications for the speed of convergence of limited memory quasi-Newton methods. In this experiment, we investigate the dynamics of $\sigma_k$ in the four initialization strategies from (20) on
a fixed problem as the iteration count $k$ increases. In particular, we use one representative run from the average results of the preceding Section’s experiments, where $n = 100$ and $r = 10$. In Fig. 3 the evolution of $\sigma_k$ of all four initializations for both; L-S-BFGS-M and L-S-BFGS-P is displayed on a structured quadratic problem with eigenvalues clustered at 1. In Fig. 4 the same quantities are displayed for structured quadratic problems with eigenvalues clustered at 1000. In green $\bar{\lambda}_{1 \leq n}$ and $\bar{\lambda}_{1 \leq r}$ are displayed, which correspond to the median taken over the first 1, 2, …, $n$ (all) and the first 1, 2, …, $r$ eigenvalues, respectively. Because in Fig. 3 the eigenvalues are clustered around 1, $\bar{\lambda}_{1 \leq n} = 1$. In Fig. 4 the eigenvalues are clustered around 1000 and $\bar{\lambda}_{1 \leq r} = 1000$. In red $\bar{\sigma}_k$ is the average $\sigma_k$ value over all iterations.

Across all plots in Figs. 3 and 4 we observe that the dynamics of $\sigma_k$ for L-S-BFGS-M and L-S-BFGS-P are similar. Moreover, the average $\bar{\sigma}_k$ is higher for Init. 1 and Init. 2 than for Init. 3 and Init. 4. The variability of Init. 2 appears less than that of Init. 1, while the variability of Init. 4 appears less than that of Init. 3. We observe that Init. 1 and 2 approximate a large eigenvalue well, whereas Init. 3 and Init. 4 approximate smaller eigenvalues better (cf. Fig. 3 lower half). Since large $\sigma_k$ values typically result in shorter step lengths (step computations use $1/\sigma_k$), choosing

![Graphs showing the evolution of $\sigma_k$ for different initializations](image)

**Fig. 3** The initialization parameter $\sigma_k$ functions as a scaling estimate of the objective’s Hessian matrix, and thus as a spectral estimate for the Hessian. The eigenvalues are clustered around 1 with $1 \leq \lambda_i \leq 1000$ and $\lambda_{i+1} = \cdots = \lambda_r = 1$. Init. 1 and Init. 2 tend to generate larger values of $\sigma_k$ (estimating larger eigenvalues), whereas Init. 3 and Init. 4 yield smaller ones (estimating smaller eigenvalues). The speed of convergence can improve, when the scaling (through the parameter $\sigma_k$) properly reflects the spectral properties of Hessian.
Init. 1 and Init. 2 result in shorter step lengths on average. Taking shorter average steps can be a desirable conservative strategy when the approximation to the full Hessian matrix is not very accurate. Therefore as a general guideline, Init. 1 and Init. 2 appear more suited for problems in which it is difficult to approximate the Hessian accurately, and Init. 3 and Init. 4 are more suited for problems in which larger step sizes are desirable. A wide ranging distinct spectrum or severe nonlinearities can be regarded as factors that make accurate approximations more challenging. If the parameter $\sigma_k$ does not properly reflect the problem’s scaling then multiple line-search iterations may be invoked in order to correct for an improperly scaled search direction each iteration, slowing convergence speed. This section displayed properties of the 4 initialization strategies of $\sigma_k$ with regards to estimating the spectrum and thus components of a problem’s scaling.

### 4.2 Experiment II

Experiment II compares the limited memory structured formulas with the full-memory update formulas from Petra et al. [22] on the CUTEst problems from [22]. The
Compact representations of structured BFGS matrices

full-memory algorithms from [22], which use Eqs. (6) and (7), are called S-BFGS-M and S-BFGS-P, respectively. The line search procedures of the limited memory structured BFGS algorithms (Algorithms 1 and 2) are the same as for the full memory algorithms. Moreover, the initializations in the full memory algorithms are set as \( A_0^M = \sigma I_n \) for S-BFGS-M, and \( A_0^P = \sigma I_n \) for S-BFGS-P, where \( \sigma = 10^i \) for the first \( i \geq 0 \) that satisfies \( (10^{i} I_n + K_0) > 0 \) (usually \( i = 0 \)). The experiments are divided into two main parts. Experiment II.A. tests the limited memory structured BFGS-Minus versions corresponding to Algorithm 1. Experiment II.A. is further subdivided into the cases in which the memory parameters are \( m = 8 \) and \( m = 50 \). These values represent a typical value \((m = 8)\) and a relatively large value \((m = 50)\), cf. e.g., [2]. Experiment II.B. tests the limited memory structured BFGS-Plus versions corresponding to Algorithm 2. As before, Experiment II.B. is further subdivided into the cases in which the memory parameters are \( m = 8 \) and \( m = 50 \). For all the solvers, we set \( \varepsilon = 1 \times 10^{-6} (\|g_k\|_{\infty} \leq \varepsilon) \) and maximum iterations to 1,000.

4.2.1 Experiment II.A: L-S-BFGS-M

In Experiment II.A we compare the limited memory implementations of Algorithm 1 with initialization strategies in (20) with the full-recursive S-BFGS-M method from (6). The solvers are tested on all 62 CUTEst problems from [22]. Figure 5 contains the results for the number of iterations.

We observe that the full-memory S-BFGS-M (black) does well in terms of number of iterations and execution time. However, L-S-BFGS-M1 (Init. 1, blue), a limited memory version with memory of only \( m = 8 \), does comparatively well. In particular, this strategy is able to solve one more problem.

Figure 6 shows the results for the computational time. Typically, a larger limited memory parameter makes using limited memory structured matrices more computationally expensive but is also expected to increase the accuracy of the quasi-Newton approximations.

Note that the outcomes of S-BFGS-M (black) in each of Figs. 5 and 6 across both plots are the same, because S-BFGS-M does not depend on the memory parameter.

---

**Fig. 5** Comparison of four initialization strategies of L-S-BFGS-M from (20) to the full-recursive method S-BFGS-M (corresponding to (6)) on all 62 CUTEst problems from [22]. The number of iterations are measured for each respective solver. Left: \( m = 8 \), right: \( m = 50 \)
For the limited memory versions we observe that the outcomes of L-S-BFGS-M2 (green) improve notably, whereas the other limited memory versions remain roughly unchanged. Using the initialization strategies (Init. 1 or Init. 2), limited memory solvers are able to solve one more problem than the full-memory method can, as indicated by the highest ending lines in the plot. We suggest that Init. 1 and Init. 2 (see Sect. 4.1.2) generate initialization parameters $\sigma_k$ that are on average larger than those generated by Init. 3 or Init. 4. These larger values in turn result in shorter average step sizes, which appears advantageous on general nonlinear problems. Note that the solvers do not reach 100% of all problems. The reason for stopping prior to the convergence criterion is that the line search ceases to make progress. Yet, the number of solved problems is similar between the full memory method (S-BFGS-M) and the proposed compact representations, indicating that the use of limited memory techniques does preserve the convergence properties of the full memory algorithm.

4.2.2 Experiment II.B: L-S-BFGS-P

In Experiment II.B we compare the versions of Algorithm 2 using the initialization strategies from (20) with the full memory recursive S-BFGS-P method (7). The solvers are run on 55 of the 62 CUTEst problems from [22] for which $n \leq 2500$.

Figure 7 contains the results for the number of iterations:

![Figure 6](image)

Fig. 6 Comparison of four initialization strategies of L-S-BFGS-M from (20) with the full-recursive method S-BFGS-M (corresponding to (6)) on all 62 CUTEst problems from [22]. The computational times are measured for each respective solver. Left: $m = 8$, right: $m = 50$

We observe that for a relatively small memory parameter $m = 8$, L-S-BFGS-M3 (Init. 3, purple) solves the most problems. L-S-BFGS-M4 (Init. 4, red) requires the fewest iterations, as indicated by the highest circle on the y-axis in the left panel of Fig. 7.

Figure 8 shows the results for the computational times. A larger limited memory parameter makes using limited memory structured matrices more computationally expensive but is also expected to increase the accuracy of the quasi-Newton approximations.

Note that the outcomes of S-BFGS-P in both plots in each figure are the same, because the full-memory solver does not depend on the memory parameter. For a
Compact representations of structured BFGS matrices

larger memory \( m = 50 \), the outcomes of L-S-BFGS-P2 (green) and L-S-BFGS-P4 (red) improve notably. Overall, L-S-BFGS-P4 solves the most problems.

From the experiments in this section, we find that initialization strategies Init.1 and Init. 2 appear most desirable for L-S-BFGS-M, whereas Init. 4 and Init. 2 appear most desirable for L-S-BFGS-P.

4.3 Experiment III

This section describes one application of the methods in the context of machine learning. A 2nd similar application to PDE constrained optimization is included, too. For all solvers we set \( m = 8 \) (memory parameter), \( \epsilon = 1 \times 10^{-6} \) (\( \|g_k\|_{\infty} \leq \epsilon \)) and maximum iterations to 10,000. Since some of the problems in this section are large we use the techniques described in Sect. 3.4 throughout the experiments. Because some of the problems in this experiment are very large, the recursive formulas from (6) and (7) (with \( m = \infty \)) cannot be directly used on these problems. However, the
limited memory compact representations use the memory parameter $m$ to threshold the computational and memory cost and are therefore applicable.

### 4.3.1 Experiment III.A: logistic regressions

The problems in this section are defined by smooth-structured objective functions from machine learning, as described, for example, in [26]. In particular, logistic regression problems use smooth objective functions for classification tasks (for instance, [6]), which often depend on a large number of data points and many variables. The classification problems are defined by the data pairs $\{d_i, y_i\}_{i=1}^D$, where the so-called feature vectors $d_i \in \mathbb{R}^n$ may be large, and the so-called labels $y_i \in \{-1, 1\}$ are scalars. In [27] regularized logistic regression problems are described in which the objective function is composed of two terms. The optimization problems are formulated as

$$
\text{minimize } \frac{\lambda}{2} \| x \|^2_2 + \sum_{i=1}^{D} \log \left( 1 + \exp(-y_i x^T d_i) \right),
$$

where $\lambda > 0$. The regularization term, $\frac{\lambda}{2} \| x \|^2_2$, has a second derivative, $\lambda I$, that is readily available. Therefore, we define the known and unknown components for this problem as

$$
\hat{k}(x) = \frac{\lambda}{2} \| x \|^2_2, \quad \hat{u}(x) = \sum_{i=1}^{D} \log \left( 1 + \exp(-y_i x^T d_i) \right). \quad (27)
$$

This data was obtained from [www.csie.ntu.edu.tw/~cjlin/libsvm/](www.csie.ntu.edu.tw/~cjlin/libsvm/) (retrieved on 10/03/19). Ten problems were used, with problem dimensions listed in Table 2.

| Problem     | $D$   | $n$   |
|-------------|-------|-------|
| rcv1        | 20242 | 47236 |
| duke        | 34    | 7129  |
| gisette     | 6000  | 5000  |
| colon_cancer| 62    | 2000  |
| leukemia    | 38    | 7129  |
| real_sim    | 72309 | 20958 |
| madelon     | 2000  | 500   |
| w8a         | 49749 | 300   |
| mushrooms   | 2000  | 500   |
| a9a         | 32561 | 123   |

Here $D$ denotes the number of training pairs $\{d_i, y_i\}_{i=1}^D$, and $n$ denotes the number of variables/feature weights (the size of the problem).
Some of the problems are large, with \( n \geq 5000 \) and thus we focus on the computations as described in Sect. 3.4.1. The regularization parameter is set as \( \lambda = 10^{-3} \). For comparison, we include IPOPT [28] with a L-BFGS quasi-Newton matrix (we use a precompiled Mex file with IPOPT 3.12.12, MUMPS and MA57). We specify the limited memory BFGS option for IPOPT using the setting `hessian_approximation= 'limited memory'` and tolerances by `tol=9.5e-10` and `acceptable_tol = 9.5e-10`. The results of the experiments are shown in Fig. 9. We observe that all solvers, except for L-S-BFGS-M2, solve the same total number of problems. Moreover, the structured L-BFGS solvers tend to use fewer iterations and overall less computational time than IPOPT’s L-BFGS method.

Next, we describe experiments for optimal control problems with similar structures.

### 4.3.2 Experiment III.B: optimal control problems

This experiment describes a typical situation in PDE constrained optimization. In particular, if the PDE is nonlinear, then we can compute gradients efficiently using the adjoint equation, but Hessians of the unknown part cannot be computed efficiently. Denoting \( u \) as the horizontal axis and \( v \) as the vertical axis, then 2D Poisson problems, with an unknown control \( x(u, v) \), are defined by the differential equation: \( y_{uu} + y_{vv} = x \). The solution \( y(u, v) \) has known boundary values on a box \( (u, v) \in [0, 1]^2 \); in other words, \( y(0, v), y(1, v), y(u, 0), \) and \( y(u, 1) \) are known. Discretizing the domain and splitting it into an interior and boundary part, we get for the optimal control problem

\[
\minimize_{x \in \mathbb{R}^n} \frac{1}{2} \left\{ \lambda \|x\|_2^2 + \|y(x) - y^*\|_2^2 \right\} \quad \text{subject to} \quad Ax = x + g,
\]

where \( g \in \mathbb{R}^n \) represents a vector with boundary information, \( A \in \mathbb{R}^{n \times n} \) is a matrix resulting from a 5-point stencil finite difference discretization of the partial derivatives, and \( y^* \) are fixed data values. Because the Hessian of the regularization term, \( \frac{\lambda}{2} \|x\|_2^2 \), is straightforward to compute, we define the structured objective function by

\[
\minimize_{x \in \mathbb{R}^n} \frac{1}{2} \|x\|_2^2 + \|y(x) - y^*\|_2^2
\]
\[ \hat{k}(x) = \frac{\lambda}{2}||x||^2_2, \quad \hat{u}(x) = \frac{1}{2}||y(x) - y^*||^2_2, \] (28)

using \( y(x) = A^{-1}(x + g) \). The number of variables is defined by the formula \( n = (10 \times j - 2)^2 \), where \( j = 2, 3, \ldots, 10 \), which corresponds to discretizations with 20, 30, \ldots, 100 mesh points in one direction. The largest problem has \( n = 9604 \) variables. We set \( \lambda = 1 \times 10^{-1} \). For comparison we also include the implementation of a “standard” BFGS method from [22], and an efficient implementation of Algorithm 1 with the constant initialization \( \sigma_k = \lambda \) (cf. Sect. 3.4.1 how to take advantage of a constant initialization) which uses the same line search as do the limited memory structured methods and IPOPT’s L-BFGS method (Fig. 10).

### 4.4 Experiment IV

In this experiment the structured solvers are compared to IPOPT [28] with an L-BFGS quasi-Newton matrix (we use a precompiled Mex file with IPOPT 3.12.12 that includes MUMPS and MA57 libraries). The objective function is a structured quartic function

\[ f(x) = \hat{k}(x) + \hat{u}(x), \quad \hat{k}(x) = \frac{1}{12} \sum_{i=1}^{n} (a_i x_i^4 + 12x_i g_i), \quad \hat{u}(x) = \frac{1}{2} \sum_{i=1}^{n} q_i x_i^2, \] (29)

where the data \( a_i, g_i \) and \( q_i \) are random normal variables with \( n = j \times 100, 1 \leq j \leq 7 \). The starting values are all ones, i.e., \( x_0 = 1 \). We specify the limited memory BFGS option for IPOPT using the setting hessian_approximation= ‘limited memory’ and tolerances by tol=9.5e-10 and acceptable_tol = 9.5e-10. For all solvers we set \( m = 8 \) (memory parameter), and maximum iterations to 10,000. A solver is regarded to have converged when \( ||g_k||_\infty \leq 9.5 \times 10^{-5} \). The average outcomes of 5 runs of the experiments are in Fig. 11.

![Fig. 10 Comparison of L-S-BFGS-M solvers on PDE constrained optimal control problems. The dimensions of the problems are \( n = (10 \times j - 2)^2 \) for \( j = 2, 3, \ldots, 10 \). Left: number of iterations, right: time. L-S-BFGS-M0 represents an efficient implementation of Algorithm 1 in which the initialization is the constant \( \sigma_k = \lambda \).](image-url)
IPOPT and the L-S-BFGS-P solvers converge to the specified tolerances on all problems. The outcomes of the number of iterations (left plot) and computational times (right plot) in Fig. 11 are consistent. In particular, we observe that the differences in the number of iterations are roughly reflected in the difference in the computational times. In this problem the known Hessian is sensitive to changes in \( \mathbf{x} \), and including second-order information in the quasi-Newton approximations yields outcomes with fewer iterations.

4.5 Experiment V

This experiment describes the application of the structured compact BFGS methods on an imaging problem. Large-scale imaging problems are challenging, because they involve large amounts of data and high-dimensional parameter space. Typically, image reconstruction problems are formulated as optimization problems. In [14], efficient gradient-based quasi-Newton techniques for large-scale ptychographic phase retrieval are described. However, even if the objective function is not directly formulated as in problem (4) it may still be possible to exploit known 2nd derivatives. Let \( \mathbf{z} = \mathbf{x} + yi \in \mathbb{C}^{\tilde{n}} \) be the object of interest, and \( \mathbf{d}_j \in \mathbb{R}^{\tilde{m}} \) be the observed data (or intensities) measured from the \( j \)th probe, where \( \tilde{n} \) and \( \tilde{m} \) are the dimensions of the vectorized object and data resolution images, respectively. A ptychography experiment is modeled by

\[
\mathbf{d}_j = |\mathcal{F}(\mathbf{Q}_j \mathbf{z})|^2 + \epsilon_j, \quad j = 1, \ldots, N,
\]

where \( N \) is the total number of probes (or scanning positions), \( \mathcal{F} : \mathbb{C}^{\tilde{m}} \rightarrow \mathbb{C}^{\tilde{n}} \) is the two-dimensional discrete Fourier operator, \( \mathbf{Q}_j \in \mathbb{C}^{\tilde{m} \times \tilde{n}} \) is the \( k \)th probe (a constant diagonal illumination matrix), and \( \epsilon_j \in \mathbb{R}^{\tilde{m}} \) is the noise. The diagonal elements of \( \mathbf{Q}_j \) are nonzero in the columns corresponding to the pixels being illuminated in the object at scanning step \( j \). There are different ways for formulating the reconstruction problem. One such formulation is the amplitude-based error metric

![Fig. 11 Comparison of L-S-BFGS-P on structured objective functions to IPOPT and L-BFGS-B. Left: number of iterations, right: time](image-url)
\[
\min_{z} f(z) = \frac{1}{2} \sum_{j=1}^{N} \left\| \mathcal{F}(Q_j z) - \sqrt{d_j} \right\|_2^2 = \frac{1}{2} \sum_{j=1}^{N} r_j^T r_j, \tag{31}
\]

where \( r_j = |\mathcal{F}(Q_j z)| - \sqrt{d_j} \) and for later use \( d_j = \sqrt{d_j} \). Even though \( f \) does not immediately separate into the form \( f = \hat{k} + \hat{u} \), the problem’s Hessian matrix has a structured separation \( \nabla^2 f = \nabla^2 \hat{k}_C + \nabla^2 \hat{u} \), in which \( \nabla^2 \hat{k}_C \) is a known constant. This will enable the definition of the vectors e.g., \( u_k \) from (5) in order to define a structured BFGS matrix. Given a known constant Hessian component, say \( \nabla^2 \hat{k}_C \), a definition of the functions \( \hat{k} \) and \( \hat{u} \) is

\[
\hat{k}(z) = \frac{1}{2} z^T \nabla^2 \hat{k}_C z, \quad \hat{u}(z) = f(z) - \hat{k}(z). \tag{32}
\]

Note that here \( \hat{k}(z) \) is only unique up to constant and linear terms. Moreover, since \( f : \mathbb{C}^n \to \mathbb{R} \) is a real-valued cost function defined on the complex domain, it is thus not complex-differentiable [23]. To overcome the lack of complex-differentiability, it is common to employ the notion of Wirtinger (Wirtinger) Calculus, where the derivatives of the real and imaginary parts of \( z \) are computed independently [23, 25]. For these real-valued functions, the mere existence of these Wirtinger derivatives is necessary and sufficient for the existence of a stationary point [3, 23, 25]. Using Wirtinger calculus (using \( z_j = \mathcal{F}Q_j z \)), the partial gradients for (31) can be computed as

\[
\nabla_z r_j = J_j = \text{diag}(z_j/\sqrt{d_j}) \mathcal{F}Q_j, \quad \nabla_z f = \sum_{j=1}^{N} J_j^* r_j = \sum_{j=1}^{N} Q_j^* \mathcal{F}^* \text{diag}(z_j/\sqrt{d_j}) (\sqrt{d_j} - z_j), \tag{33}
\]

**Hessian.** To compute the Hessian matrix, let

\[
T_{1,j} = Q_j^* \mathcal{F}^* \text{diag}(d_j/\sqrt{z_j}) \mathcal{F}Q_j,
\]

and

\[
T_{2,j} = Q_j^* \mathcal{F}^* \text{diag}(d_j \otimes z_j^2/\sqrt{z_j}) \mathcal{F}Q_j,
\]

then

\[
\mathbf{H} = \sum_{j=1}^{N} \begin{bmatrix} Q_j^* Q_j - \Re(T_{1,j}) + \Re(T_{2,j}) & \Im(T_{1,j}) + \Im(T_{2,j}) \\ \Im(T_{1,j}) + \Im(T_{2,j}) & Q_j^* Q_j - \Re(T_{1,j}) - \Re(T_{2,j}) \end{bmatrix}, \tag{34}
\]

where the known 2nd derivatives are \( \mathbf{K} = \sum_{j=1}^{N} \begin{bmatrix} Q_j^* Q_j \\ Q_j^* Q_j \end{bmatrix} \) and the remaining block elements of \( \mathbf{H} \) are estimated. Since all \( Q_j \)'s are constant, \( \mathbf{K} \) is thus a known constant matrix. Therefore we specify \( \hat{k}(z) \) in (32) by setting \( \nabla^2 \hat{k}_C = \mathbf{K} \).
Defining the vectorization from complex to real variables by \( xx = vecR(z) \equiv vec(\Re(z), \Im(z)) \), where \( vec(x_1, x_2) = [x_1^T, x_2^T]^T \), we define the vectors for the structured BFGS methods by \( y_k = vecR(\nabla f(z_{k+1})) - vecR(\nabla f(z_k)) \), \( s_k = vecR(z_{k+1}) - vecR(z_k) \), \( \widehat{u}_k = y_k - Ks_k \) and

\[
\mathbf{u}_k = \mathbf{u}_k + Ks_k = y_k.
\]

Using these vectors, we can form the compact structured BFGS matrices. In this experiment, we compare a limited memory structured BFGS method (L-S-BFGS) and limited memory BFGS (L-BFGS) method in Fig. 12. The image dimensions are \( n = 50 \) so that the total number of real variables is \( n = 2 \cdot n^2 = 5,000 \). Moreover, \( m = 16 \) so that \( m^2 \times n^2 = 256 \times 5,000 = 1,280,000 \), and \( N = 16 \). Because of the structure of the known Hessian \( l = 1 \) (cf. Table 1), and solves with this matrix are done on the order of \( O(n) \). Because of the size of this problem, and the corresponding computational/memory requirements the recursive update formulas from Eqs. (6) and (7) are not applicable, yet the limited memory techniques threshold the required computational resources.

5 Conclusions

In this article we develop the compact representations of structured BFGS formulas. Limited memory versions of the compact representations with four non-constant initialization strategies are implemented in two line search algorithms. The proposed limited memory compact representations enable efficient search
direction computations by the Sherman-Morrison-Woodbury formula and the use of efficient initialization strategies. The proposed methods are compared in a collection of experiments, which include the original full-memory methods. The structured methods typically require fewer total iterations than do the unstructured approaches. Among the four proposed initialization strategies, initializations 1 and 2 appear best for the structured minus methods (L-S-BFGS-M), whereas initializations 4 and 2 appear robust for the structured plus (L-S-BFGS-P) methods. In an array of applications, including a large-scale real world imaging problem, the proposed structured limited memory methods obtain better numerical results than conventional unstructured methods.

Appendix A: initialization comparison with $\phi = 1000$

In Sect. 4.1, the four L-S-BFGS-M initializations were compared on structured quadratic objective functions with eigenvalues clustered around 1, whereas in this section the eigenvalues are clustered around 1000. In particular, $\phi = 1000$, and the elements of $\mathbf{D}$ are uniformly distributed in the interval $[-999, 0]$. The results are displayed in Fig. 13. For the large clustered eigenvalues Init. 1 and 3 require the fewest iterations, while Init. 3 appears fastest overall. For L-S-BFGS-P the computations with $\phi = 1000$ are in Fig. 14.

In the comparison of L-S-BFGS-P, Init. 2 and Init. 3 do best in iterations.

Fig. 13 Comparison of initialization strategies for L-S-BFGS-M on problems with eigenvalues clustered around 1,000 with $1 \leq \lambda_r \leq 1000$ and $\lambda_{r+1} = \cdots = \lambda_n = 1000$. Left: number of iterations; right: time.
Fig. 14 Comparison of initialization strategies for L-S-BFGS-P on problems with eigenvalues clustered around 1,000 with $1 \leq \lambda_i \leq 1000$ and $\lambda_{r+1} = \cdots = \lambda_n = 1000$. Left: number of iterations; right: time

Acknowledgements This work was supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357 at Argonne National Laboratory. This work was also performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. We like to thank for the reviewer’s valuable insights, which improved the coherence of the derivations, and the overall article.

References

1. Barzilai, J., Borwein, J.: Two-point step size gradient methods.IMA J. Numer. Anal. 8(1), 141–148 (1988). https://doi.org/10.1093/imanum/8.1.141
2. Boggs, P., Byrd, R.: Adaptive, limited-memory bfgs algorithms for unconstrained optimization. SIAM J. Optim. 29(2), 1282–1299 (2019). https://doi.org/10.1137/16M1065100
3. Brandwood, D.H.: A complex gradient operator and its application in adaptive array theory. IEEE Proc. F Commun. Radar Signal Process. 130(1), 11–16 (1983)
4. Broyden, C.G.: The convergence of a class of double-rank minimization algorithms 1. General considerations. IMA J. Appl. Math. 6(1), 76–90 (1970). https://doi.org/10.1093/imanum/6.1.76
5. Brust, J.J., Burdakov, O., Erway, J.B., Marcia, R.F.: A dense initialization for limited-memory quasi-newton methods. Comput. Optim. Appl. 74(1), 121–142 (2019). https://doi.org/10.1007/s10589-019-00112-x
6. Byrd, R.H., Chin, G.M., Neveitt, W., Nocedal, J.: On the use of stochastic hessian information in optimization methods for machine learning. SIAM J. Optim. 21, 977–995 (2011). https://doi.org/10.1137/10079923X
7. 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)
8. Chang, C.C., Lin, C.J.: Libsvm: a library for support vector machines. ACM Trans. Intell. Syst. Technol. 2(3), 27:1-27:27 (2011)
9. Dennis, J., Moré, J.: Quasi-newton methods, motivation and theory. SIAM Rev. 19, 46–89 (1977)
10. Dennis, J.E., Jr., Gay, D.M., Walsh, R.E.: An adaptive nonlinear least-squares algorithm. ACM Trans. Math. Softw. 7(3), 348–368 (1981). https://doi.org/10.1145/355958.355965
11. Dennis, J.E., Jr., Martinez, H.J., Tapia, R.A.: Convergence theory for the structured bfgs secant method with an application to nonlinear least squares. J. Optim. Theory Appl. 61(2), 161–178 (1989). https://doi.org/10.1007/BF00962795
12. Dolan, E., Moré, J.: Benchmarking optimization software with performance profiles. Math. Program. 91, 201–213 (2002)
13. Fletcher, R.: A new approach to variable metric algorithms. Comput. J. 13(3), 317–322 (1970). https://doi.org/10.1093/comjnl/13.3.317
14. Fung, S.W., Di Wendy, Z.: Multigrid optimization for large-scale ptychographic phase retrieval. SIAM J. Imaging Sci. 13(1), 214–233 (2020). https://doi.org/10.1137/18M1223915
15. Gill, P.E., Murray, W.: Algorithms for the solution of the nonlinear least-squares problem. SIAM J. Numer. Anal. 11, 311–365 (2010)
16. Goldfarb, D.: A family of variable-metric methods derived by variational means. Math. Comp. 24, 23–26 (1970). https://doi.org/10.1090/S0025-5718-1970-0258249-6
17. Gould, N.I.M., Orban, D., Toint, P.L.: CUTEr and SifDec: a constrained and unconstrained testing environment, revisited. ACM Trans. Math. Softw. 29(4), 373–394 (2003)
18. Mahajan, A., Leyffer, S., Kirches, C.: Solving mixed-integer nonlinear programs by qp diving. Technical Report ANL/MCS-P2071-0312, Mathematics and Computer Science Division, Argonne National Laboratory, Lemont, IL (2012)
19. Moré, J.J., Thuente, D.J.: Line search algorithms with guaranteed sufficient decrease. ACM Trans. Math. Softw. 20(3), 286–307 (1994). https://doi.org/10.1145/192115.192132
20. Nocedal, J.: Updating quasi-Newton matrices with limited storage. Math. Comput. 35, 773–782 (1980)
21. Nocedal, J., Wright, S.J.: Numerical Optimization. Springer-Verlag, New York (1999)
22. Petra, C., Chiang, N., Anitescu, M.: A structured quasi-newton algorithm for optimizing with incomplete hessian information. SIAM J. Optim. 29(2), 1048–1075 (2019). https://doi.org/10.1137/18M1167942
23. Remmert, R.: Theory of Complex Functions. Springer-Verlag, New York (1991)
24. Shanno, D.F.: Conditioning of quasi-Newton methods for function minimization. Math. Comput. 24, 647–656 (1970). https://doi.org/10.1090/S0025-5718-1970-0274029-X
25. Sorber, L., Barel, M.V., Lathauwer, L.D.: Unconstrained optimization of real functions in complex variables. SIAM J. Optim. 22(3), 879–898 (2012). https://doi.org/10.1137/110832124
26. Sra, S., Nowozin, S., Wright, S.J.: Optimization for Machine Learning. The MIT Press, New York (2011)
27. Teo, C.H., Vishwanathan, S., Smola, A.J., Le, Q.V.: Bundle methods for regularized risk minimization. J. Mach. Learn. Res. 11, 311–365 (2010)
28. Wächter, A., Biegler, L.T.: On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. Math. Program. 106, 25–57 (2006)
29. Yabe, H., Takahashi, T.: Factorized quasi-newton methods for nonlinear least squares problems. Math. Program. (1991). https://doi.org/10.1007/BF01586927
30. Zhu, C., Byrd, R., Nocedal, J.: Algorithm 778: L-BFGS-B: fortran subroutines for large-scale bound-constrained optimization. ACM Trans. Math. Softw. 23, 550–560 (1997)

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.