frontiers in Applied Mathematics and Statistics

ORIGINAL RESEARCH
published: 24 May 2021
doi: 10.3389/fams.2021.673412

Performance Analysis of Trust Region Subproblem Solvers for Limited-Memory Distributed BFGS Optimization Method

Guohua Gao[1]*, *Horacio Florez*[1], *Jeroen C. Vink*[2], *Terence J. Wells*[2], *Fredrik Saaf*[1] and *Carl P. A. Blom*[2]

[1]Shell Global Solutions (United States) Inc., Houston, TX, United States, [2]Shell Global Solutions International B.V., Amsterdam, Netherlands

OPEN ACCESS

Edited by:
Olwijn Leeuwenburgh,
Netherlands Organisation for Applied Scientific Research, Netherlands

Reviewed by:
Andreas Størksen Stordal,
Norwegian Research Institute (NORCE), Norway
Sarfraz Ahmad,
COMSATS University Islamabad, Pakistan

***Correspondence:**
Guohua Gao
guohua.gao@shell.com

Specialty section:
This article was submitted to Optimization, a section of the journal Frontiers in Applied Mathematics and Statistics

Received: 27 February 2021
Accepted: 29 April 2021
Published: 24 May 2021

Citation:
Gao G, Florez H, Vink JC, Wells TJ, Saaf F and Blom CPA (2021) Performance Analysis of Trust Region Subproblem Solvers for Limited-Memory Distributed BFGS Optimization Method. Front. Appl. Math. Stat. 7:673412. doi: 10.3389/fams.2021.673412

The limited-memory Broyden–Fletcher–Goldfarb–Shanno (L-BFGS) optimization method performs very efficiently for large-scale problems. A trust region search method generally performs more efficiently and robustly than a line search method, especially when the gradient of the objective function cannot be accurately evaluated. The computational cost of an L-BFGS trust region subproblem (TRS) solver depend mainly on the number of unknown variables (***n***) and the number of variable shift vectors and gradient change vectors (***m***) used for Hessian updating, with **m** << **n** for large-scale problems. In this paper, we analyze the performances of different methods to solve the L-BFGS TRS. The first method is the direct method using the Newton-Raphson (DNR) method and Cholesky factorization of a dense $n \times n$ matrix, the second one is the direct method based on an inverse quadratic (DIQ) interpolation, and the third one is a new method that combines the matrix inversion lemma (MIL) with an approach to update associated matrices and vectors. The MIL approach is applied to reduce the dimension of the original problem with ***n*** variables to a new problem with ***m*** variables. Instead of directly using expensive matrix-matrix and matrix-vector multiplications to solve the L-BFGS TRS, a more efficient approach is employed to update matrices and vectors iteratively. The L-BFGS TRS solver using the MIL method performs more efficiently than using the DNR method or DIQ method. Testing on a representative suite of problems indicates that the new method can converge to optimal solutions comparable to those obtained using the DNR or DIQ method. Its computational cost represents only a modest overhead over the well-known L-BFGS line-search method but delivers improved stability in the presence of inaccurate gradients. When compared to the solver using the DNR or DIQ method, the new TRS solver can reduce computational cost by a factor proportional to n^2/m for large-scale problems.

Keywords: limited-memory BFGS method, trust region search optimization method, trust region subproblem, matrix inversion lemma, low rank matrix update

INTRODUCTION

Decision-making tools based on optimization procedures have been successfully applied to solve practical problems in a wide range of areas. An optimization problem is generally defined as minimizing (or maximizing) an objective function $\boldsymbol{f}(\boldsymbol{x})$ within a user defined search domain $\boldsymbol{x} \in \boldsymbol{\Omega}$, and subject to some linear or nonlinear constraints, where $\boldsymbol{x}$ is an $\boldsymbol{n}$-dimensional vector that contains all controllable variables.

In the oil and gas industry, an optimal business development plan requires robust production optimization because of the considerable uncertainty of subsurface reservoir properties and volatile oil prices. Many papers have been published on the topic of robust optimization and their applications to cyclic CO_2 flooding through the Gas-Assisted Gravity Drainage process [1], well placement optimization in geologically complex reservoirs [2], optimal production schedules of smart wells for water flooding [3] and in naturally fractured reservoirs [4], just mentioning a few of them as examples.

Some researchers formulated the optimization problem under uncertainty as a single objective optimization problem, e.g., only maximizing the mean of net present value (NPV) for simplicity by neglecting the associated risk. However, it is recommended to formulate robust optimization as a bi-objective optimization problem for consistency and completeness. A bi-objective optimization problem is generally defined as minimizing (or maximizing) two different objective functions, $\boldsymbol{f}_1(\boldsymbol{x})$ and $\boldsymbol{f}_2(\boldsymbol{x})$, within a user-defined search domain $\boldsymbol{x} \in \boldsymbol{\Omega}$, and subject to some linear or nonlinear constraints. For example, we may maximize the mean value, denoted by $\boldsymbol{f}_1(\boldsymbol{x})$, and minimize the standard deviation, denoted by $\boldsymbol{f}_2(\boldsymbol{x})$, of NPV. For a bi-objective optimization problem, the optimal solutions are defined as the Pareto optimal solutions (or Pareto front). It is a very challenging task to find multiple optimal solutions on the Pareto front [5, 6].

It is also a very challenging task to properly characterize the uncertainty of reservoir properties (e.g., porosity and permeability in each grid-block) and reliably quantify the ensuing uncertainty of production forecasts (e.g., production rates of oil, gas, and water phases) by conditioning to historical production data and 4D seismic data [7, 8], which requires generating multiple conditional realizations by minimizing a properly defined objective function, e.g., using the randomized maximum likelihood (RML) method [9].

When an adjoint-based gradient of the objective function is available [10–12], we may apply a gradient-based optimization method, e.g., the Broyden–Fletcher–Goldfarb–Shanno (BFGS) optimization method [13, 14]. However, the adjoint-based gradient is not available for many commercial simulators and not available for integer type or discrete variables (e.g., well locations defined by grid-block indices). In such a case, we must apply a derivative-free optimization (DFO) method [15–18]. For problems with smooth objective functions and continuous variables, model-based DFO methods using either radial-basis function [19] or quadratic model [16, 20, 21] performs more efficiently than other DFO methods such as direct pattern search methods [22] and stochastic search methods [23].

Traditional optimization methods only locate a single optimal solution, and they are referred to as single-thread optimization methods in this paper. It is unacceptably expensive to use a single-thread optimization method to locate multiple optimal solutions on the Pareto front or generate multiple RML samples [24]. To overcome the limitations of single-thread optimization methods, Gao et al. [17] developed a local-search distributed Gauss-Newton (DGN) DFO method to find multiple best matches concurrently. Later, Chen et al. [25] modified the local-search DGN optimization method and generalized the DGN optimizer for global search. They also integrated the global-search DGN optimization method with the RML method to generate multiple RML samples in parallel.

Because multiple search points are generated in each iteration, finally, multiple optimal solutions can be found in parallel. These distributed optimization methods are referred to as multiple-thread optimization methods. Both the local- and global-search DGN optimization methods are only applicable to a specific optimization problem, i.e., history matching problem or least-squares optimization problem, of which the objective function can be expressed as a form of least-squares, $\boldsymbol{f}(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{\alpha}\boldsymbol{x}^T\boldsymbol{x} + \frac{1}{2}\boldsymbol{y}^T(\boldsymbol{x})\boldsymbol{y}(\boldsymbol{x})$, but they cannot be applied to other type or generic optimization problems where the objective function cannot be expressed as a form of least-squares. Furthermore, the DGN optimization methods may become less efficient for history matching problems when both the number of variables ($\boldsymbol{n}$) and the number of observed data ($\boldsymbol{m}$) are large (e.g., in the order of thousands or more).

Recently, Gao et al. [18] developed a well-paralleled distributed quasi-Newton (DQN) DFO method for generic optimization problems by introducing a generalized form of the objective function, $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{f}(\boldsymbol{x})$. Here, $\boldsymbol{x}$ represents the vector of controllable variables or model parameters to be optimized (explicit variables) and $\boldsymbol{y}(\boldsymbol{x})$ denotes the vector of simulated responses (implicit variables) of a reservoir using explicit variables $\boldsymbol{x}$. Using the generalized form of the objective function, the gradient of the objective function can be evaluated analytically by,

$$\boldsymbol{g}(\boldsymbol{x}) = \nabla_{\boldsymbol{x}}\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y}) + \boldsymbol{J}^T(\boldsymbol{x})\nabla_{\boldsymbol{y}}\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y}) \tag{1}$$

In **Eq. 1**, $\boldsymbol{J}^T(\boldsymbol{x}) = \nabla_{\boldsymbol{x}}\boldsymbol{y}^T(\boldsymbol{x})$ is the transpose of the sensitivity matrix. Using the generalized expression of the objective function $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y})$, different objective functions can be evaluated using the same $\boldsymbol{y}(\boldsymbol{x})$ that is simulated from the same reservoir model, e.g., for multi-objective optimization problems. A specific case of the generalized expression is $\boldsymbol{F}(\boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$, where the transpose of the sensitivity matrix becomes the gradient of the objective function.

The DQN optimization algorithm runs $\boldsymbol{N_e}$ optimization threads in parallel. In the $\boldsymbol{k}$-th iteration, there are $\boldsymbol{N_{LC}^{(k)}} \leq \boldsymbol{N_e}$ non-converged threads. The DQN optimizer generates a new search points $\boldsymbol{x_T^{(i,k)}} = \boldsymbol{x^{(i,k)}} + \boldsymbol{s^{(i,k)}}$ for each non-converged thread, where the search step $\boldsymbol{s^{(i,k)}}$ is solved from a trust region subproblem (TRS). The $\boldsymbol{N_{LC}^{(k)}}$ simulation cases are submitted to a high-performance computing (HPC) cluster in parallel, to generate corresponding simulated responses $\boldsymbol{y_T^{(i,k)}} = \boldsymbol{y}(\boldsymbol{x_T^{(i,k)}})$.

Then, we evaluate the objective function $f_T^{(i,k)} = F(x_T^{(i,k)}, y_T^{(i,k)})$ and its associated partial derivatives $\nabla_x F^{(i,k)}$ and $\nabla_y F^{(i,k)}$. If the new search point $x_T^{(i,k)}$ improves the objective function, i.e., $f_T^{(i,k)} < f^{(i,k)} = f(x^{(i,k)})$, then we update $x^{(i,k+1)} = x_T^{(i,k)}$ and $f^{(i,k+1)} = f_T^{(i,k)}$.

All simulation results generated by all DQN optimization threads in previous and current iterations are recorded in a training data set, and they are shared among all DQN optimization threads. The sensitivity matrix $J^{(i,k+1)} = J(x^{(i,k+1)})$ is approximated using a modified QR-method proposed by Gao, et al. [18] by linear interpolation of training data points that are closest to $x^{(i,k+1)}$. Then, we approximate the gradient $g^{(i,k+1)} = g(x^{(i,k+1)})$ using **Eq. 1**. Finally, the Hessian $H^{(i,k+1)}$ for each thread is updated using either the BFGS method or the symmetric rank-1 (SR1) method [14]. For simplicity, we will drop the superscript "i" (the optimization thread index) in the following discussions.

Both DGN and DQN optimization methods approximate the objective function by a quadratic model, and they are designed for problems with smooth objective function and continuous variables. Although their convergence is not guaranteed for problems with integer type variables (e.g., well location optimization), if those integers can be treated as truncated continuous variables, our numerical tests indicate that these distributed optimization methods can improve the objective function significantly and locate multiple suboptimal solutions for problems with integer type variables in only a few iterations.

The number of variables for real-world problems may vary from a few to thousands or even more, depending on the problem and the parameterization techniques employed. For example, the number of variables could be in the order of millions if we do not apply any parameter reduction techniques to reduce the dimension of some history matching problems (e.g., to tune permeability and porosity in each grid block). Because reservoir properties are generally correlated with each other with long correlation lengths, we may reduce the number of variables to be tuned to only a few hundred, e.g., using the principal component analysis (PCA) or other parameter reduction techniques [26].

In this paper, our focus is on performance analysis of different methods to solve the TRS formulated with the limited-memory BFGS (L-BFGS) Hessian updating method for unconstrained optimization problems. In the future, we will further integrate the new TRS solver with our newly proposed limited-memory distributed BFGS (L-DBFGS) optimization algorithm and then apply the new optimizer to some realistic oil/gas field optimization cases and benchmark its overall performance against other distributed optimization methods such as those only using the gradient [27]. We will also continue our investigation in the future for constrained optimization problems (e.g., variables with lower- and/or upper-bounds and problems with nonlinear constraints). Gao et al. [18] applied the popular Newton-Raphson method [28] to directly solve the TRS using matrix factorization, the DNR method in short. The DNR method is quite expensive when it is applied to solve N_e TRSs of a distributed optimization method, especially for large-scale problems with thousands or more variables [29]. This paper follows the ideas and concepts presented in the book "Matrix Computation" [30]. Flops are used to quantify the volume of work associated with a computation, a count for floating-point operations of add, subtract, multiply, or divide. Computational cost (flops) of some commonly used algebraic operations and numerical methods are summarized in **Table 1** for reference.

For completeness, we discuss the compact representation of the L-BFGS Hessian updating formulation [31] and the algorithm to directly update the Hessian in the next section directly. In the third section, we present three different methods to solve the L-BFGS TRS: the DNR method, the direct method using inverse quadratic (DIQ) interpolation approach proposed by Gao et al. [32], and the technique using matrix inversion lemma (MIL) together with an efficient matrix updating algorithm. Some numerical tests and performance comparisons are discussed in the fourth section. We finally draw some conclusions in the last section.

THE LIMITED-MEMORY HESSIAN UPDATING FORMULATION

Let $x^{(k)}$ be the best solution obtained in the current (k-th) iteration, and $f^{(k)}$, $g^{(k)}$, and $H^{(k)}$, respectively, the objective function, its gradient and Hessian evaluated at $x^{(k)}$. The Hessian $H^{(k+1)}$ is updated using the BFGS formulation as follows,

$$H^{(k+1)} = H^{(k)} + \frac{z^{(k)}\left[z^{(k)}\right]^T}{\left[z^{(k)}\right]^T s^{(k)}} - \frac{H^{(k)} s^{(k)}\left[s^{(k)}\right]^T H^{(k)}}{\left[s^{(k)}\right]^T H^{(k)} s^{(k)}} \tag{2}$$

In **Eq. 2**, $s^{(k)} = x^{(k+1)} - x^{(k)}$ and $z^{(k)} = g^{(k+1)} - g^{(k)}$. The Hessian $H^{(k+1)}$ updated using **Eq. 2** is guaranteed positive definite if the condition $\varepsilon^{(k)} = [z^{(k)}]^T s^{(k)} > c_3 z^{(k)} s^{(k)}$ is satisfied where $c_3 > 0$ is a small positive number.

Compact Representation

To save memory usage and computational cost, the L-BFGS Hessian updating method limits the maximum history size or the maximum number of pairs ($s^{(k)}$, $z^{(k)}$) used for the BFGS Hessian updating to $L_M > 1$. The recommended value of L_M ranges from 5 to 20, using smaller number for problems with more variables.

Let $1 \le l_k \le L_M$ denote the number of pairs of variable shift vectors and gradient change vectors, $(s^{(j)}, z^{(j)})$ for $j = 1,2,...l_k$, used to update the Hessian using the L-BFGS method in the k-th iteration. Let $S^{(k)} = [s^{(1)}, s^{(2)}, \ldots s^{(l_k)}]$ be the $n \times l_k$ variable shift matrix and $Z^{(k)} = [z^{(1)}, z^{(2)}, \ldots z^{(l_k)}]$ the $n \times l_k$ gradient change matrix. Both matrices $S^{(k)}$ and $Z^{(k)}$ are updated iteration by iteration. Let $m = 2l_k$ and $V^{(k)} = [S^{(k)}, Z^{(k)}]$ is an $n \times m$ matrix. Let $A^{(k)} = [S^{(k)}]^T S^{(k)}$ and $B^{(k)} = [S^{(k)}]^T Z^{(k)}$. We decompose $B^{(k)}$ into three parts: the strictly lower triangular part $L^{(k)}$, the strictly upper triangular part $U^{(k)}$, and the diagonal part $E^{(k)}$ that contains the main diagonals of $B^{(k)}$, i.e.,

$$B^{(k)} = L^{(k)} + E^{(k)} + U^{(k)} \tag{3}$$

The Hessian can be updated using the compact form of the L-BFGS Hessian updating formulation [31, 33–35] as follows,

$$H^{(k+1)} = \alpha^{(k)} I_n - V^{(k)} W^{(k)} \left[V^{(k)}\right]^T \tag{4}$$

TABLE 1 | A summary of computational costs of commonly used algebraic operations.

Operation	Dimension	Computational cost (flops)
$a = x^T y$	$x, y \epsilon R^n$	$2n$
$y = y + axy$	$a \epsilon R$; $x, y \epsilon R^n$	$2n$
$y = y + Ax$	$A \epsilon R^{m\times n}$; $x \epsilon R^n$; $y \epsilon R^m$	$2mn$
$A = A + yx^T$	$A \epsilon R^{m\times n}$; $x \epsilon R^n$; $y \epsilon R^m$	$2mn$
$C = C + AB$	$A \epsilon R^{m\times r}$; $B \epsilon R^{r\times n}$; $C \epsilon R^{m\times n}$	$2mnr$
$A = LL^T$	Symmetric $A \epsilon R^{n\times n}$; $L \epsilon R^{n\times n}$	$n^3/3$
Solve x from $Lx=y$	Lower triangle $L \epsilon R^{n\times n}$; $x \epsilon R^n$; $y \epsilon R^n$	n^2

In **Eq. 4**, I_n is an $n \times n$ identity matrix and $W^{(k)}$ is an $m\times m$ symmetric matrix defined as,

$$\left[W^{(k)}\right]^{-1} = \frac{1}{\alpha^{(k)}} \begin{Bmatrix} A^{(k)} & L^{(k)} \\ \left[L^{(k)}\right]^T & -\alpha^{(k)} E^{(k)} \end{Bmatrix} \tag{5}$$

In **Eq. 4** and **Eq. 5**, $\alpha^{(k)} = \frac{\|z^{(l_k)}\|^2}{[z^{(l_k)}]^T s^{(l_k)}}$ is a scaling factor. It is required that $a^{(k)} > a_{cr} > 0$ using **Eq. 4** and **Eq. 5** to update the Hessian, where a_{cr} is a small positive number.

The Algorithm to Directly Update the Hessian Using the L-BFGS Compact Representation

Given $L_M \geq 1$, $k \geq 1$, $l_{k-1} \geq 1$, the thresholds $c_3 > 0$ and $a_{cr} > 0$, search step $s = x^{(k+1)} - x^{(k)}$ and gradient change $z = g^{(k+1)} - g^{(k)}$, the two $n \times l_{k-1}$ matrices, $S^{(k-1)}$ and $Z^{(k-1)}$, we can update $S^{(k)}$, $Z^{(k)}$ and directly compute the Hessian $H^{(k+1)}$ using the compact representation of the L-BFGS Hessian updating **Eq. 4**, as summarized in Algorithm-1.

Algorithm-1: Updating the Hessian directly using the L-BFGS compact representation

1. *Compute* $||s||, ||z||, \varepsilon^{(k)} = z^T s$ *and* $\alpha^{(k)} = \frac{\|z^2\|}{\varepsilon^{(k)}}$ if $\varepsilon^{(k)} > c_3 \|s\| \|z\|$;
2. *If* $\varepsilon^{(k)} \leq c_3 \|s\| \|z\|$ *or* $\alpha^{(k)} \leq \alpha_{cr}$, *set* $l_k = l_{k-1}$, $S^{(k)} = S^{(k-1)}$, $Z^{(k)} = Z^{(k-1)}$, *and goto step 4;*
3. *Otherwise*,
   a. *Set* $l_k = L_M$ *and remove the first column from* $S^{(k-1)}$ *and* $Z^{(k-1)}$ *if* $l_{k-1} = L_M$; *else set* $l_k = l_{k-1} + 1$;
   b. *Update* $S^{(k)} = [S^{(k-1)}\ s]$ *and* $Z^{(k)} = [Z^{(k-1)}\ z]$;
4. *Form* $V^{(k)} = [S^{(k)}\ \ Z^{(k)}]$;
5. *Compute* $A^{(k)} = [S^{(k)}]^T S^{(k)}$ *and* $B^{(k)} = [S^{(k)}]^T Z^{(k)}$;
6. *Form the matrix* $W_I^{(k)} = [W^{(k)}]^{-1}$ *using* ***Eq. 5***;
7. *Compute* $W^{(k)} = [W_I^{(k)}]^{-1}$ *using Cholesky factorization;*
8. *Compute* $U^{(k)} = W^{(k)} [V^{(k)}]^T$;
9. *Compute* $T^{(k)} = V^{(k)} U^{(k)}$;
10. *Update* $H^{(k+1)} = \alpha^{(k)} I_n - T^{(k)}$.
11. *Stop*.

The computational cost to directly update the Hessian using the L-BFGS method as described in Algorithm-1 mainly depends on the number of variables (n) and the number of vectors used to update the Hessian ($m = 2l_k$). We should reemphasize that m is updated iteratively but limited to $m \leq 2L_k$. The computational cost is $c_1 = 2mn^2 + (7 + 3m^2)n + O(m^3)$ (flops), which includes the following seven operations: 1) computing $\|s\|$, $\|z\|$, and $\varepsilon^{(k)} = z^T s$ in step 1 ($6n$ flops); 2) computing $A^{(k)}$ and $B^{(k)}$ in step 5 ($m^2 n$ flops); 3) forming the matrix $W_I^{(k)}$ in step 6 (m^2 flops); 4) computing $W^{(k)}$ in step 7 ($O(m^3)$ flops); 5) compute $U^{(k)}$ in step 8 ($2m^2 n$ flops); 6) compute $T^{(k)}$ in step 9 ($2mn^2$ flops); and 7) updating $H^{(k+1)}$ in step 10 (n flops).

SOLVING THE L-BFGS TRUST REGION SUBPROBLEM

The Trust Region Subproblem

In the neighborhood of the best solution $x^{(k)}$, the objective function $f(x)$ can be approximated by a quadratic function of $s = x - x^{(k)}$,

$$q^{(k)}(s) = f^{(k)} + \left[g^{(k)}\right]^T s + \frac{1}{2} s^T H^{(k)} s \tag{6}$$

If the Hessian $H^{(k)}$ is positive definite, then $q^{(k)}(s)$ has a unique minimum $s^{*(k)}$, which is the solution of $H^{(k)} s = -g^{(k)}$,

$$s^{*(k)} = -\left[H^{(k)}\right]^{-1} g^{(k)} \tag{7}$$

When a line search strategy is applied, we accept the full search step $s^{*(k)}$ by setting $x^{(k+1)} = x^{(k)} + s^{*(k)}$ if it improves the objective function sufficiently (e.g., satisfying the two Wolfe conditions). Otherwise, we can find a better search step size $0 < \gamma^{(k)} \leq 1$ such that $x^{(k+1)} = x^{(k)} + \gamma^{(k)} s^{*(k)}$ improves the objective function sufficiently. If the gradient $g^{(k)}$ can be accurately evaluated, it has theoretically proved that a line search strategy can converge [14]. However, for most real-world optimization problems, the gradient cannot be accurately evaluated, and as discussed by Gao et al. [16], a trust region search strategy performs more robustly and more efficiently than a line search strategy.

The trust region search step $s^{*(k)}$ for the next iteration is the global minimum of the quadratic model $q^{(k)}(s)$ within a ball-shaped trust region with radius $\Delta^{(k)} \geq \Delta_{min} > 0$ which is solved from the following trust region subproblem (TRS) [28],

$$\min_s q^{(k)}(s) = f^{(k)} + \left[g^{(k)}\right]^T s + \frac{1}{2} s^T H^{(k)} s \text{ subject to } ||s||_2 \leq \Delta^{(k)} \tag{8}$$

If the solution $s^{*(k)}$ given by **Eq. 7** satisfies $\left\|s^{*(k)}\right\|_2 \leq \Delta^{(k)}$, then $s^{*(k)}$ is the solution of the TRS defined in **Eq. 8**. Otherwise, the solution

of the TRS must lie on the boundary of $\|s\|_2 = \Delta^{(k)}$, and we should solve the Lagrange multiplier $\lambda^{*(k)}$ together with the search step $s^{*(k)}$ from the following nonlinear TRS equations,

$$\left[H^{(k)} + \lambda I_n\right] s(\lambda) = -g^{(k)} \tag{9}$$

$$\pi(\lambda) = [s(\lambda)]^T s(\lambda) = \left[\Delta^{(k)}\right]^2 \tag{10}$$

For the trivial case with $H^{(k)} = \alpha I_n$, we have $s(\lambda) = -\frac{g^{(k)}}{\alpha+\lambda}$ and $\pi(\lambda) = \frac{\|g^{(k)}\|^2}{(\alpha+\lambda)^2}$. The solution of the TRS defined in **Eq. 8** is $\lambda^* = 0$ and $s^{*(k)} = -g^{(k)}/\alpha$ if $\|g^{(k)}\|^2 \le \alpha\Delta^{(k)}$ or $\lambda^* = \frac{\|g^{(k)}\|^2}{\Delta^{(k)}} - \alpha$ and $s^{*(k)} = -g^{(k)}\frac{\Delta^{(k)}}{\|g^{(k)}\|^2}$ otherwise.

We may apply the popular DNR method to solve the TRS when $H^{(k)}$ is a dense matrix. However, the DNR method is quite expensive for large-scale problems and we should seek and apply a more efficient method to solve the TRS.

Solving the L-BFGS TRS Directly Using the Newton-Raphson Method

The DNR method solves the TRS defined in **Eq. 8** using the Cholesky decomposition. It applies the Newton-Raphson method to directly solve the nonlinear TRS equation [28], $\phi(\lambda) = \frac{1}{s(\lambda)} - \frac{1}{\Delta} = 0$, iteratively, which requires computing the first order derivative $\phi'(\lambda) = -\frac{v(\lambda)^2}{s(\lambda)^3}$ where v is solved from $L^T v = s$ together with the Cholesky factorization of $D = H^{(k)} + \lambda I_n = LL^T$ or LU-decomposition.

Given the trust region size $\Delta = \Delta^{(k)} > 0$, threshold of convergence $\delta_{cr} > 0$, maximum number of iterations allowed $N_{TRS,max} > 0$, Hessian $H = H^{(k)}$ and gradient $g = g^{(k)}$ evaluated at the current best solution $x^{(k)}$, both the Lagrange multiplier λ^* and the trust region search step $s^* = s(\lambda^*)$ can be solved from the TRS defined in **Eq. 8**, using the DNR method as summarized in Algorithm-2.

Algorithm-2: Solving the L-BFGS TRS Using the DNR Method

1. *Initialize* $l = 0$, $\lambda_0 = 0$;
2. *Compute* $D = H + \lambda_0 I_n$ *and Cholesky decomposition* $D = LL^T$;
3. *Solve* u^* *from* $Lu = -g$, s^* *from* $L^T s = u^*$, *and* v^* *from* $L^T v = s^*$;
4. *Compute* $\|s^*\|$ *and* $\|v^*\|$;
5. *If* $\|s^*\| \le \Delta$, *then accept* $\lambda^* = \lambda_0$ *and go to step 8;*
6. *Set* $\delta = \left|1 - \frac{\|s^*\|}{\Delta}\right|$;
7. *Repeat steps (a) through (f) below, until convergence* ($\delta \le \delta_{cr}$ *or* $l > N_{TRS,max}$):
   a. *Update* $\lambda_{l+1} = \lambda_l + \left(\frac{\|s^*\|}{\|v^*\|}\right)^2 \frac{\|s^*\|-\Delta}{\Delta}$;
   b. *Compute* $D = H + \lambda_{l+1} I_n$ *and Cholesky decomposition* $D = LL^T$;
   c. *Solve* u^* *from* $Lu = -g$, s^* *from* $L^T s = u^*$, *and* v^* *from* $L^T v = s^*$;
   d. *Compute* $\|s^*\|$ *and* $\|v^*\|$;
   e. *Update* $\delta = \left|1 - \frac{\|s^*\|}{\Delta}\right|$;
   f. *Set* $l = l + 1$.
8. *Stop.*

Let N_{DNR} denote the number of iterations required for the DNR TRS solver to converge. The total computational cost (flops) to solve the TRS using Algorithm-2 is, $c_2 = \left(5n + 3n^2 + \frac{n^3}{3}\right)(N_{DNR} + 1)$, including the following three operations: (1) computing $D = H + \lambda_{k+1} I_n$ and Cholesky decomposition $D = LL^T$ in step 2 and step 7(b) ($n + \frac{n^3}{3}$flops); (2) solving u^* from $Lu = -g$, s^* from $L^T s = u^*$, and v^* from $L^T v = s^*$ in step 3 and step 7(c) ($3n^2$ flops); (3) computing $\|s^*\|$ and $\|v^*\|$ in step 4 and step 7(d) ($4n$ flops).

The total computational cost (flops) used to solve the L-BFGS TRS using the DNR method (Algorithm-2) together with the L-BFGS Hessian updating method (Algorithm-1) is,

$$c_{DNR} = c_1 + c_2 = 2mn^2 + (7 + 3m^2)n + \left(5n + 3n^2 + \frac{n^3}{3}\right)(N_{DNR} + 1) + \mathbf{O}(m^3) \tag{11}$$

Our numerical results indicate that the DNR TRS solver may fail when tested on the well-known Rosenbrock function, especially for problems with large n. The root cause for failure of convergence using the DNR method is the same as in the GNTRS solver using the traditional Newton-Raphson method as discussed by Gao et al. [36]. Very small value of $|\phi'(\lambda)|$ may result in a very large search step and thus result in failure of converging. Gao et al. [36] proposed integrating the DNR method with a bisection line search to overcome this issue.

The Inverse Quadratic Model Interpolation Method to Directly Solve the L-BFGS TRS

Instead of applying the Newton-Raphson method which requires evaluating $\phi'(\lambda)$, Gao et al. [32] proposed a method to directly solve the TRS using inverse quadratic model interpolation (called the DIQ method), i.e., approximating $\pi(\lambda) = [s(\lambda)]^T s(\lambda)$ by the following inverse quadratic function,

$$q_{INV}(\lambda) = \frac{b}{(\lambda + a)^2} \tag{12}$$

Both coefficients of a and b in **Eq. 12** can be determined by interpolating the values of $\pi(\lambda)$ evaluated at two different points $\pi_1 = \pi(\lambda_1)$ and $\pi_2 = \pi(\lambda_2)$ with $0 \le \lambda_1 < \lambda_2$.

In the first iteration, we set $\lambda_1 = \lambda_{min} = 0$ and accept $\lambda_{min} = 0$ as the solution if $\pi_1 \le \Delta^2$. Otherwise, $\pi_1 > \Delta^2$ holds and we set $\lambda_2 = \lambda_{max} = max\left[0, \frac{\|g^{(k)}\|}{\Delta} - \alpha\right]$ and $\pi_2 < \Delta^2$ holds if it is not accepted as the solution. In the following iteration, either λ_1 or λ_2 will be updated accordingly such that the two conditions $\pi_1 > \Delta^2$ and $\pi_2 < \Delta^2$ always hold.

Let $\rho = \frac{\sqrt{\pi_1}}{\sqrt{\pi_1} - \sqrt{\pi_2}} \ge 1$ and the following solution of $q_{INV}(\lambda) = \Delta^2$ will be used as the trial search point for the next iteration,

$$\lambda^* = \lambda_2 - \rho\left(1 - \frac{\sqrt{\pi_2}}{\Delta}\right)(\lambda_2 - \lambda_1) \tag{13}$$

We either update $\lambda_1 = \lambda^*$ if $\pi(\lambda^*) > \Delta^2$ or update $\lambda_2 = \lambda^*$ if $\pi(\lambda^*) < \Delta^2$ iteratively until convergence. We accept λ^* as the desired solution and terminate the iterative process either when $\left|\frac{\sqrt{\pi(\lambda^*)}}{\Delta} - 1\right| < \delta_{cr}$, the tolerance for convergence, or when the number of iterations used to solve the TRS (N_{DIQ}) reaches the user specified maximum iteration number ($N_{TRS,max}$), i.e., $N_{DIQ} \geq N_{TRS,max}$.

Given the scaling factor $\alpha > 0$, trust region size $\Delta > 0$, threshold of convergence $\delta_{cr} > 0$, maximum number of iterations allowed $N_{TRS,max} > 0$, Hessian H and gradient g, both the Lagrange multiplier λ^* and the trust region search step $s^* = s(\lambda^*)$ can be solved from the TRS defined in **Eq. 8**, using the DIQ method as summarized in Algorithm-3.

Algorithm-3: Solving the L-BFGS TRS Using the DIQ Method

(1) *Compute* $\|g\|$;
(2) *Calculate* $\lambda_{max} = max\{0, \frac{\|g\|}{\Delta} - \alpha\}$;
(3) *Initialize* $k = 0$, $\lambda_1 = \lambda_{min} = 0$ *and* $\lambda_2 = \lambda_{max}$;
(4) *Compute* $D = H + \lambda_1 I_n$ *and Cholesky decomposition* $D = LL^T$;
(5) *Solve* u_1 *from* $Lu = -g$ *and* s_1 *from* $L^T s = u_1$;
(6) *Compute* $\pi_1 = \pi(\lambda_1) = s_1^T s_1$;
(7) *If* $\pi_1 \leq \Delta^2$, *then accept* $\lambda^* = \lambda_1$ *and go to step 12;*
(8) *If* $\pi_1 > \Delta^2$, *then*

a. *Compute* $D = H + \lambda_2 I_n$ *and Cholesky decomposition* $D = LL^T$;
b. *Solve* u_2 *from* $Lu = -g$ *and* s_2 *from* $L^T s = u_2$;
c. *Compute* $\pi_2 = \pi(\lambda_2) = s_2^T s_2$;

(9) *If* $\left|1 - \frac{\sqrt{\pi_1}}{\Delta}\right| < \left|1 - \frac{\sqrt{\pi_2}}{\Delta}\right|$, *then* $\delta = \left|1 - \frac{\sqrt{\pi_1}}{\Delta}\right|$, $\lambda^* = \lambda_1$;
(10) *Otherwise,* $\delta = \left|1 - \frac{\sqrt{\pi_2}}{\Delta}\right|$, $\lambda^* = \lambda_2$;
(11) *Repeat steps (a) through (i) below, until convergence* ($\delta \leq \delta_{cr}$ or $k > N_{TRS,max}$):

a. *Calculate* $\rho = \frac{\sqrt{\pi_1}}{\sqrt{\pi_1} - \sqrt{\pi_2}}$;
b. *Calculate* $\lambda^* = \lambda_2 - \rho\left(1 - \frac{\sqrt{\pi_2}}{\Delta}\right)(\lambda_2 - \lambda_1)$;
c. *Compute* $D = H + \lambda^* I_n$*and Cholesky decomposition* $D = LL^T$;
d. *Solve* u^* *from* $Lu = -g$ *and* s^* *from* $L^T s = u^*$;
e. Compute $\pi^* = \pi(\lambda^*) = [s^*]^T s^*$;
f. *Update* $\delta = \left|1 - \frac{\sqrt{\pi^*}}{\Delta}\right|$;
g. *If* $\sqrt{\pi^*} > \Delta$, *update* $\lambda_1 = \lambda^*$ *and* $\pi_1 = \pi^*$;
h. *Otherwise, update* $\lambda_2 = \lambda^*$ *and* $\pi_2 = \pi^*$;
i. $k \leftarrow k + 1$

(12) *Stop.*

Let N_{DIQ} denote the number of iterations required for the DIQ TRS solver to converge. The total computational cost to solve the L-BFGS TRS using Algorithm-3 together with Algorithm-1 is,

$$c_{IQ} = 2mn^2 + (7 + 3m^2)n + \left(5n + 2n^2 + \frac{n^3}{3}\right)(N_{DIQ} + 1) + O(m^3) \tag{14}$$

Generally, the DIQ method converges faster than the DNR method and it performs more efficiently and robustly than the DNR method, especially for large scale problems. Both the DNR method and the DIQ method become quite expensive for large-scale problems.

Using Matrix Inversion Lemma (MIL) to Solve the L-BFGS TRS

To save both memory usage and computational cost, Gao, et al. [32, 36, 37] proposed an efficient algorithm to solve the Gauss-Newton TRS (GNTRS) for large-scale history matching problems using the matrix inversion lemma (or the Woodbury matrix identity). With appropriate normalization of both parameters and residuals, the Hessian of the objective function for a history matching problem can be approximated by the well-known Gauss-Newton equation as follows,

$$H^{(k+1)} = \alpha I_n + V^{(k)}\left[V^{(k)}\right]^T \tag{15}$$

In **Eq. 15**, $V^{(k)} = [J^{(k+1)}]^T$ is an $n \times m$ matrix, the transpose of the sensitivity matrix $J^{(k+1)}$ that is evaluated at the current best solution $x^{(k+1)}$. Here, m denotes the number of observed data to be matched and it is not the same m as used in the L-BFGS Hessian updating **Eq. 4**.

The GNTRS solver proposed by Gao, et al. [32, 36, 37] using the matrix inversion lemma (MIL) has been implemented and integrated with the distributed Gauss-Newton (DGN) optimizer. Because the compact representation of the L-BFGS Hessian updating formula **Eq. 4** is similar to **Eq. 15**, we follow the similar idea as proposed by Gao, et al. [32, 36, 37] to compute $[H^{(k+1)} + \lambda I_n]^{-1}$ by applying the matrix inversion lemma and then solve the L-BFGS trust region search step $s(\lambda)$.

$$\left[H^{(k+1)} + \lambda I_n\right]^{-1} = \frac{1}{\alpha^{(k)} + \lambda} I_n - \frac{1}{[\alpha^{(k)} + \lambda]^2} V^{(k)} \left\{\frac{1}{\alpha^{(k)} + \lambda}\left[V^{(k)}\right]^T V^{(k)} - \left[W^{(k)}\right]^{-1}\right\}^{-1} \left[V^{(k)}\right]^T \tag{16}$$

Let $P^{(k)} = [V^{(k)}]^T V^{(k)}$ and $u^{(k)} = [V^{(k)}]^T g^{(k)}$. We first solve $v(\lambda)$ from,

$$\left\{\frac{1}{\alpha^{(k)} + \lambda} P^{(k)} - \left[W^{(k)}\right]^{-1}\right\} v(\lambda) = u^{(k)}. \tag{17}$$

Then, we compute the trust region search step $s(\lambda)$ and $\pi(\lambda) = \|s(\lambda)\|^2$ as,

$$s(\lambda) = -\left[H^{(k+1)} + \lambda I_n\right]^{-1} g^{(k)} = -\frac{1}{\alpha^{(k)} + \lambda} g^{(k)} + \frac{1}{[\alpha^{(k)} + \lambda]^2} V^{(k)} v(\lambda), \quad (18)$$

$$\pi(\lambda) = \frac{\left\|g^{(k)}\right\|^2}{[\alpha^{(k)} + \lambda]^2} + \frac{[v(\lambda)]^T P^{(k)} v(\lambda)}{[\alpha^{(k)} + \lambda]^4} - \frac{2[u^{(k)}]^T v(\lambda)}{[\alpha^{(k)} + \lambda]^3}. \quad (19)$$

From **Eq. 15**, we have $P^{(k)} v(\lambda) = [\alpha^{(k)} + \lambda]\{u^{(k)} + [W^{(k)}]^{-1} v(\lambda)\}$ and **Eq. 19** can be rewritten as,

$$\pi(\lambda) = \frac{\left\|g^{(k)}\right\|^2}{[\alpha^{(k)} + \lambda]^2} + \frac{[v(\lambda)]^T [W^{(k)}]^{-1} v(\lambda) - [u^{(k)}]^T v(\lambda)}{[\alpha^{(k)} + \lambda]^3}. \quad (20)$$

We first try $\lambda = \lambda_{min} = 0$. If $\pi(0) \le \Delta^2$ holds, we accept $\lambda^* = 0$ and $s^* = s(0)$ as the solution of the TRS defined in **Eq. 8**. Otherwise, the solution is on the boundary defined in **Eq. 10**, which can be solved by the Newton-Raphson (NR) method iteratively.

Let $\phi(\lambda) = \frac{1}{\sqrt{\pi(\lambda)}} - \frac{1}{\Delta}$ and we have $\phi'(\lambda) = -\frac{1}{2} \frac{\pi'(\lambda)}{[\pi(\lambda)]^{3/2}}$ where $\pi'(\lambda)$ is computed by,

$$\pi'(\lambda) = \frac{\left\|g^{(k)}\right\|^2}{[\alpha^{(k)} + \lambda]^3} - \frac{3\pi(\lambda)}{\alpha^{(k)} + \lambda} + \frac{2[v(\lambda)]^T [W^{(k)}]^{-1} w(\lambda) - [u^{(k)}]^T w(\lambda)}{[\alpha^{(k)} + \lambda]^3}. \quad (21)$$

In **Eq. 21**, $w(\lambda) = v'(\lambda)$ is solved from,

$$\left\{\frac{1}{\alpha^{(k)} + \lambda} P^{(k)} - \left[W^{(k)}\right]^{-1}\right\} w(\lambda) = \frac{1}{[\alpha^{(k)} + \lambda]^2} P^{(k)} v(\lambda). \quad (22)$$

Because $\left\{\frac{1}{\alpha^{(k)}+\lambda} P^{(k)} - [W^{(k)}]^{-1}\right\}$ is an $m \times m$ symmetric and positive definite matrix, it only requires. $\frac{m^3}{3} + 6m^2$ flops to solve $v(\lambda)$ and $w(\lambda)$ from **Eq. 17** and **Eq. 22** using the Cholesky factorization.

Given the $n \times l_k$ variable shift matrix $S^{(k)}$ and gradient change matrix $Z^{(k)}$, we may directly compute the following three $l_k \times l_k$ matrices, $A^{(k)} = [S^{(k)}]^T S^{(k)}$, $B^{(k)} = [S^{(k)}]^T Z^{(k)}$ and $C^{(k)} = [Z^{(k)}]^T Z^{(k)}$, and then form the $m \times m$ matrix $P^{(k)} = [V^{(k)}]^T V^{(k)}$ as,

$$P^{(k)} = \begin{Bmatrix} A^{(k)} & B^{(k)} \\ [B^{(k)}]^T & C^{(k)} \end{Bmatrix} \quad (23)$$

Directly computing the three $l_k \times l_k$ matrices, $A^{(k)}$, $B^{(k)}$ and $C^{(k)}$, requires $6l_k^2 n = 1.5m^2 n$ flops. We also need to compute the vector $u^{(k)} = [V^{(k)}]^T g^{(k)}$ ($2mn$ flops) and $\left\|g^{(k)}\right\|^2$ ($2n$ flops). To further reduce the computational cost, the three matrices $A^{(k)}$, $B^{(k)}$ and $C^{(k)}$ and the vector $u^{(k)}$ should be updated iteratively instead.

Updating Matrices and Vectors Used for Solving the L-BFGS TRS

We first initialize $k = 0$ and $H^{(0)} = I_n$. If the trial search point $x_T^{(0)} = x^{(0)} + s^{*(0)}$ does not improve the objective function, i.e., $f(x_T^{(0)}) \ge f(x^{(0)})$, we set $x^{(k+1)} = x^{(0)}$ and $H^{(k+1)} = H^{(0)}$, recompute the sensitivity matrix $J^{(k+1)} = J(x^{(0)})$ and the gradient $g^{(k+1)} = g(x^{(0)})$ when needed (e.g., using updated training data points), and shrink the trust region size $\Delta^{(k+1)} = \gamma_d \Delta^{(k)}$ with $0 < \gamma_d < 1$. We repeatedly generate trial search point $x_T^{(k)} = x^{(k)} + s^{*(k)}$ until $x_T^{(k)}$ improves the objective function, i.e., $f(x_T^{(k)}) < f(x^{(0)})$.

If $f(x_T^{(k)}) < f(x^{(0)})$, we set $l_k = 1$, $x^{(k+1)} = x_T^{(k)}$, evaluate $g^{(k+1)} = g(x^{(k+1)})$, and compute $s = x^{(k+1)} - x^{(k)}$ and $z = g^{(k+1)} - g^{(k)}$. In the following iterations, we update $l_k \ge 1$ and associated matrices and vectors used for solving the L-BFGS TRS for different cases accordingly. We use $Sflag^{(k)}$ to indicate whether the new search point $x_T^{(k)}$ improves the objective function ($Sflag^{(k)} = "True"$) or not ($Sflag^{(k)} = "False"$).

The L-BFGS Hessian updating formulation of **Eq. 4** requires $\varepsilon^{(k)} = z^T s > c_3 \|z\| \|s\|$ and $\alpha^{(k)} = \frac{z^T z}{z^T s} > \alpha_{cr} > 0$ where c_3 and α_{cr} are small positive numbers. If $\varepsilon^{(k)} \le c_3 \|z\| \|s\|$ or $\alpha^{(k)} \le \alpha_{cr}$, we simply set $l_k = l_{k-1}$, $s^{(l_k)} = s^{(l_{k-1})}$, $z^{(l_k)} = z^{(l_{k-1})}$, $\alpha^{(k)} = \alpha^{(k-1)}$, $S^{(k)} = S^{(k-1)}$, $Z^{(k)} = Z^{(k-1)}$, $A^{(k)} = A^{(k-1)}$, $B^{(k)} = B^{(k-1)}$, $C^{(k)} = C^{(k-1)}$, and $P^{(k)} = P^{(k-1)}$, with no updating. However, we need to compute $u^{(k)} = [S^{(k)}, Z^{(k)}]^T g^{(k+1)}$ using the new gradient $g^{(k+1)}$. In the following cases, we assume that both conditions $\varepsilon^{(k)} > c_3 \|z\| \|s\|$ and $\alpha^{(k)} > \alpha_{cr}$ are satisfied.

Case-1: $Sflag^{(k)}$ = "False"

Although the best solution $x^{(k+1)}$ does not change when $Sflag^{(k)}$ = "*False*", the sensitivity matrix $J^{(k+1)}$ and thus the gradient of the objective function $g^{(k+1)}$ evaluated at $x^{(k+1)}$ using simulation results and training data points updated in the $k+1$ (current) iteration may be different from those evaluated at the same point but using simulation results and training data points obtained in the k (previous) iteration. To use the right gradient, we should replace the last column in $Z^{(k)}$ with z.

Let $\tilde{Z}^{(k-1)}$ be the submatrix of $Z^{(k-1)}$ by removing its last column $\tilde{z}^{(l_{k-1})}$, $\tilde{B}^{(k-1)} = [S^{(k-1)}]^T \tilde{Z}^{(k-1)}$ the submatrix of $B^{(k-1)} = [S^{(k-1)}]^T Z^{(k-1)}$ by removing its last column, and $\tilde{C}^{(k-1)} = [\tilde{Z}^{(k-1)}]^T \tilde{Z}^{(k-1)}$ the submatrix of $C^{(k-1)} = [Z^{(k-1)}]^T Z^{(k-1)}$ by removing its last row and last column. We first compute the following two vectors $p_2^{(k)} = [S^{(k-1)}]^T z$ and $\tilde{p}_4^{(k)} = [\tilde{Z}^{(k)}]^T z$ and two scalars $\mu^{(k)} = z^T z$ and $\tau^{(k)} = z^T g^{(k+1)} - [\tilde{z}^{(l_{k-1})}]^T g^{(k)}$, with computational cost $2mn$ flops.

We set $l_k = l_{k-1}$, $s^{(l_k)} = s^{(l_{k-1})}$, $z^{(l_k)} = z$, $S^{(k)} = S^{(k-1)}$, and $A^{(k)} = A^{(k-1)}$, and Update $Z^{(k)}$ as,

$$Z^{(k)} = \left[\tilde{Z}^{(k-1)} \quad z\right]. \quad (24)$$

It is straightforward to derive the following equations to update $B^{(k)}$ and $C^{(k)}$ accordingly,

$$B^{(k)} = \left[S^{(k-1)}\right]^T \left[\tilde{Z}^{(k-1)} \quad z\right] = \left[\tilde{B}^{(k-1)} \quad p_2^{(k)}\right], \quad (25)$$

$$C^{(k)} = \begin{Bmatrix} \left[\tilde{Z}^{(k-1)}\right]^T \\ z^T \end{Bmatrix} \left[\tilde{Z}^{(k-1)} \quad z\right] = \begin{Bmatrix} \tilde{C}^{(k-1)} & \tilde{p}_4^{(k)} \\ \left[\tilde{p}_4^{(k)}\right]^T & \mu^{(k)} \end{Bmatrix}. \quad (26)$$

By definition,

$$u^{(k-1)} = \left[S^{(k-1)}, \ Z^{(k-1)}\right]^T g^{(k)} = \left[S^{(k-1)}, \ \tilde{Z}^{(k-1)}, \tilde{z}^{(l_{k-1})}\right]^T g^{(k)}$$

$$\boldsymbol{u}^{(k)} = \left[\boldsymbol{S}^{(k)},\ \boldsymbol{Z}^{(k)}\right]\boldsymbol{g}^{(k+1)} = \left[\boldsymbol{S}^{(k-1)},\ \tilde{\boldsymbol{Z}}^{(k-1)}, \boldsymbol{z}\right]^T\boldsymbol{g}^{(k+1)}$$

Thus, we have,

$$\boldsymbol{u}^{(k)} = \boldsymbol{u}^{(k-1)} + \begin{Bmatrix} \left[\boldsymbol{S}^{(k-1)},\ \tilde{\boldsymbol{Z}}^{(k)}\right]^T z \\ \boldsymbol{z}^T\boldsymbol{g}^{(k+1)} - \left[\tilde{\boldsymbol{z}}^{(l_{k-1})}\right]^T\boldsymbol{g}^{(k)} \end{Bmatrix} = \boldsymbol{u}^{(k-1)} + \begin{bmatrix} \boldsymbol{p}_2^{(k)} \\ \tilde{\boldsymbol{p}}_4^{(k)} \\ \boldsymbol{\tau}^{(k)} \end{bmatrix}. \tag{27}$$

Case-2: $\mathrm{Sflag}^{(k)}$ = "True" *and* $\mathrm{l_{k-1}} < \mathrm{L_M}$

We set $\boldsymbol{l_k} = \boldsymbol{l_{k-1}} + 1$ and $\boldsymbol{s}^{(l_k)} = \boldsymbol{s}$ and $\boldsymbol{z}^{(l_k)} = \boldsymbol{z}$, compute four $\boldsymbol{l_{k-1}}$-dimensional vectors $\boldsymbol{p}_1^{(k)} = [\boldsymbol{S}^{(k-1)}]^T\boldsymbol{s}$ and $\boldsymbol{p}_2^{(k)} = [\boldsymbol{S}^{(k-1)}]^T\boldsymbol{z}$ and $[\boldsymbol{p}_3^{(k)}]^T = \boldsymbol{s}^T\boldsymbol{Z}^{(k-1)}$ and $\boldsymbol{p}_4^{(k)} = [\boldsymbol{Z}^{(k-1)}]^T\boldsymbol{z}$ and five scalars, $\boldsymbol{\beta}^{(k)} = \boldsymbol{s}^T\boldsymbol{s}$ and $\boldsymbol{\varepsilon}^{(k)} = \boldsymbol{s}^T\boldsymbol{z}$ and $\mu^{(k)} = \boldsymbol{z}^T\boldsymbol{z}$ and $\boldsymbol{\gamma}_s^{(k)} = \boldsymbol{s}^T\boldsymbol{g}^{(k+1)}$ and $\boldsymbol{\gamma}_z^{(k)} = \boldsymbol{z}^T\boldsymbol{g}^{(k+1)}$, with computational cost $4\boldsymbol{mn}$ flops. Both $\boldsymbol{S}^{(k)}$ and $\boldsymbol{Z}^{(k)}$ are updated by,

$$\boldsymbol{S}^{(k)} = \left[\boldsymbol{S}^{(k-1)}\ \ \boldsymbol{s}\right] \tag{28}$$

$$\boldsymbol{Z}^{(k)} = \left[\boldsymbol{Z}^{(k-1)}\ \ \boldsymbol{z}\right] \tag{29}$$

It is straightforward to derive the following equations to update $\boldsymbol{A}^{(k)}$, $\boldsymbol{B}^{(k)}$, and $\boldsymbol{C}^{(k)}$,

$$\boldsymbol{A}^{(k)} = \left[\boldsymbol{S}^{(k-1)}\ \boldsymbol{s}\right]^T\left[\boldsymbol{S}^{(k-1)}\ \boldsymbol{s}\right] = \begin{Bmatrix} \boldsymbol{A}^{(k-1)} & \boldsymbol{p}_1^{(k)} \\ \left[\boldsymbol{p}_1^{(k)}\right]^T & \boldsymbol{\beta}^{(k)} \end{Bmatrix} \tag{30}$$

$$\boldsymbol{B}^{(k)} = \left[\boldsymbol{S}^{(k-1)}\ \boldsymbol{s}\right]^T\left[\boldsymbol{Z}^{(k-1)}\ \boldsymbol{z}\right] = \begin{Bmatrix} \boldsymbol{B}^{(k-1)} & \boldsymbol{p}_2^{(k)} \\ \left[\boldsymbol{p}_3^{(k)}\right]^T & \boldsymbol{\varepsilon}^{(k)} \end{Bmatrix} \tag{31}$$

$$\boldsymbol{C}^{(k)} = \left[\boldsymbol{Z}^{(k-1)}\ \boldsymbol{Z}\right]^T\left[\boldsymbol{Z}^{(k-1)}\ \boldsymbol{z}\right] = \begin{Bmatrix} \boldsymbol{C}^{(k-1)} & \boldsymbol{p}_4^{(k)} \\ \left[\boldsymbol{p}_4^{(k)}\right]^T & \mu^{(k)} \end{Bmatrix} \tag{32}$$

We first split the $2\boldsymbol{l_{k-1}}$-dimensional vector $\boldsymbol{u}^{(k-1)}$ into two $\boldsymbol{l_{k-1}}$-dimensional sub-vectors $\boldsymbol{u}_S^{(k-1)}$ and $\boldsymbol{u}_Z^{(k-1)}$, i.e.,

$$\boldsymbol{u}^{(k-1)} = \left[\boldsymbol{S}^{(k-1)},\ \boldsymbol{Z}^{(k-1)}\right]^T\boldsymbol{g}^{(k)} = \begin{bmatrix} \boldsymbol{u}_S^{(k-1)} \\ \boldsymbol{u}_Z^{(k-1)} \end{bmatrix}.$$

where $\boldsymbol{u}_S^{(k-1)} = [\boldsymbol{S}^{(k-1)}]^T\boldsymbol{g}^{(k)}$ is composed of the first l_{k-1} rows in $\boldsymbol{u}^{(k-1)}$ and $\boldsymbol{u}_Z^{(k-1)} = [\boldsymbol{Z}^{(k-1)}]^T\boldsymbol{g}^{(k)}$ is composed of the last $\boldsymbol{l_{k-1}}$ rows in $\boldsymbol{u}^{(k-1)}$.

Because $\boldsymbol{u}^{(k)} = [\boldsymbol{S}^{(k)},\ \boldsymbol{Z}^{(k)}]\boldsymbol{g}^{(k+1)} = [\boldsymbol{S}^{(k-1)}, \boldsymbol{s}\ \ \boldsymbol{Z}^{(k-1)}, \boldsymbol{z}]^T\boldsymbol{g}^{(k+1)}$ and $\boldsymbol{g}^{(k+1)} = \boldsymbol{g}^{(k)} + \boldsymbol{z}$, thus we have,

$$\boldsymbol{u}^{(k)} = \begin{bmatrix} \boldsymbol{u}_S^{(k-1)} + \boldsymbol{p}_2^{(k)} \\ \boldsymbol{\gamma}_s^{(k)} \\ \boldsymbol{u}_Z^{(k-1)} + \boldsymbol{p}_4^{(k)} \\ \boldsymbol{\gamma}_z^{(k)} \end{bmatrix}. \tag{33}$$

Case-3: $\mathrm{Sflag}^{(k)}$ = "True" *and* $\mathrm{l_{k-1}} = \mathrm{L_M}$

We set $\boldsymbol{l_k} = \boldsymbol{l_{k-1}} = \boldsymbol{L_M}$. Let $\widehat{\boldsymbol{S}}^{(k-1)}$ and $\widehat{\boldsymbol{Z}}^{(k-1)}$ denote the submatrices of $\boldsymbol{S}^{(k-1)}$ and $\boldsymbol{Z}^{(k-1)}$ by removing the first column $\boldsymbol{s}^{(1)}$ and $\boldsymbol{z}^{(1)}$ from $\boldsymbol{S}^{(k-1)}$ and $\boldsymbol{Z}^{(k-1)}$, respectively. Let $\widehat{\boldsymbol{A}}^{(k-1)}$ and $\widehat{\boldsymbol{B}}^{(k-1)}$ and $\widehat{\boldsymbol{C}}^{(k-1)}$ denote the submatrices of $\boldsymbol{A}^{(k)}$, $\boldsymbol{B}^{(k)}$, and $\boldsymbol{C}^{(k)}$ by removing their first row and first column.

We apply **Eq. 28** through **Eq. 32** to update $\boldsymbol{S}^{(k)}$, $\boldsymbol{Z}^{(k)}$, $\boldsymbol{A}^{(k)}$, $\boldsymbol{B}^{(k)}$, and $\boldsymbol{C}^{(k)}$ by simply replacing $\boldsymbol{S}^{(k-1)}$, $\boldsymbol{Z}^{(k-1)}$, $\boldsymbol{A}^{(k-1)}$, $\boldsymbol{B}^{(k-1)}$, and $\boldsymbol{C}^{(k-1)}$ with $\widehat{\boldsymbol{S}}^{(k-1)}$ and $\widehat{\boldsymbol{Z}}^{(k-1)}$ and $\widehat{\boldsymbol{A}}^{(k-1)}$ and $\widehat{\boldsymbol{B}}^{(k-1)}$ and $\widehat{\boldsymbol{C}}^{(k-1)}$, respectively. We can also apply **Eq. 33** to update $\boldsymbol{u}^{(k)}$ by replacing $\boldsymbol{u}_S^{(k-1)}$ with $\widehat{\boldsymbol{u}}_S^{(k-1)} = [\widehat{\boldsymbol{S}}^{(k-1)}]^T\boldsymbol{g}^{(k)}$ that is composed of the second row through the $\boldsymbol{l_{k-1}}$-th rows in $\boldsymbol{u}^{(k-1)}$ and replacing $\boldsymbol{u}_Z^{(k-1)}$ with $\widehat{\boldsymbol{u}}_Z^{(k-1)} = [\widehat{\boldsymbol{Z}}^{(k-1)}]^T\boldsymbol{g}^{(k)}$ that is composed of the last $\boldsymbol{l_{k-1}} - 1$ rows in $\boldsymbol{u}^{(k-1)}$.

The Algorithm to Update Matrices and Vectors Used for Solving the L-BFGS TRS

Given $\boldsymbol{L_M} \geq 1$, $\boldsymbol{k} > 1$, $\boldsymbol{l_{k-1}} \geq 1$, the thresholds $\boldsymbol{c}_3 > 0$ and $\boldsymbol{a}_{\mathrm{cr}} > 0$, $\boldsymbol{n}$-dimensional vectors $\boldsymbol{g}^{(k+1)}$, $\boldsymbol{s}$ and $\boldsymbol{z}$, $\boldsymbol{l_{k-1}}$-dimensional vector $\boldsymbol{u}^{(k-1)}$, $\boldsymbol{n} \times \boldsymbol{l_{k-1}}$ matrices $\boldsymbol{S}^{(k-1)}$ and $\boldsymbol{Z}^{(k-1)}$, $\boldsymbol{l_{k-1}} \times \boldsymbol{l_{k-1}}$ matrices $\boldsymbol{A}^{(k-1)}$, $\boldsymbol{B}^{(k-1)}$ and $\boldsymbol{C}^{(k-1)}$, $2\boldsymbol{l_{k-1}} \times 2\boldsymbol{l_{k-1}}$ matrices $\boldsymbol{P}^{(k-1)}$ and $\boldsymbol{W}_I^{(k-1)} = [\boldsymbol{W}^{(k-1)}]^{-1}$, we can update $\boldsymbol{l_k}$ and the $\boldsymbol{l_k}$-dimensional vector $\boldsymbol{u}^{(k)}$, $\boldsymbol{n} \times \boldsymbol{l_k}$ matrices $\boldsymbol{S}^{(k)}$ and $\boldsymbol{Z}^{(k)}$, $\boldsymbol{l_k} \times \boldsymbol{l_k}$ matrices $\boldsymbol{A}^{(k)}$, $\boldsymbol{B}^{(k)}$, and $\boldsymbol{C}^{(k)}$, and $2\boldsymbol{l_k} \times 2\boldsymbol{l_k}$ matrices $\boldsymbol{P}^{(k)}$ and $\boldsymbol{W}_I^{(k)} = [\boldsymbol{W}^{(k)}]^{-1}$, using the algorithm as summarized in Algorithm-4.

Algorithm-4: Updating Matrices and Vectors Used for Solving the L-BFGS TRS

1. *Compute* $\boldsymbol{\varepsilon}^{(k)} = \boldsymbol{z}^T\boldsymbol{s}$, $\mu^{(k)} = \boldsymbol{z}^T\boldsymbol{z}$, $\boldsymbol{\beta}^{(k)} = \boldsymbol{s}^T\boldsymbol{s}$;
2. *Compute the scaling factor* $\boldsymbol{\alpha}^{(k)} = \mu^{(k)}/\boldsymbol{\varepsilon}^{(k)}$ *if* $\boldsymbol{\varepsilon}^{(k)} > \boldsymbol{c}_3\sqrt{\boldsymbol{\beta}^{(k)}\mu^{(k)}}$, *or set* $\boldsymbol{\alpha}^{(k)} = 0$ *otherwise;*
3. *If* $\boldsymbol{\alpha}^{(k)} \leq \boldsymbol{\alpha}_{cr}$ *or* $\boldsymbol{\varepsilon}^{(k)} \leq \boldsymbol{c}_3\sqrt{\boldsymbol{\beta}^{(k)}\mu^{(k)}}$, *then*
   a. *Set* $\boldsymbol{\alpha}^{(k)} = \boldsymbol{\alpha}^{(k-1)}$ *and* $\boldsymbol{l_k} = \boldsymbol{l_{k-1}}$;
   b. *Update* $\boldsymbol{S}^{(k)} = \boldsymbol{S}^{(k-1)}$ *and* $\boldsymbol{Z}^{(k)} = \boldsymbol{Z}^{(k-1)}$;
   c. *Update* $\boldsymbol{A}^{(k)} = \boldsymbol{A}^{(k-1)}$, $\boldsymbol{B}^{(k)} = \boldsymbol{B}^{(k-1)}$, *and* $\boldsymbol{C}^{(k)} = \boldsymbol{C}^{(k-1)}$;
   d. *Update* $\boldsymbol{P}^{(k)} = \boldsymbol{P}^{(k-1)}$ *and* $\boldsymbol{WI}^{(k)} = \boldsymbol{WI}^{(k-1)}$;
   e. *Compute* $\boldsymbol{u}^{(k)} = [\boldsymbol{S}^{(k)},\ \boldsymbol{Z}^{(k)}]^T\boldsymbol{g}^{(k+1)}$;
   f. *Goto step 10.*
4. *Else if* $\boldsymbol{Sflag}^{(k)}$ = "**False**" *(for Case-1), then*
   a. *Set* $\boldsymbol{l_k} = \boldsymbol{l_{k-1}}$;
   b. *Update* $\boldsymbol{S}^{(k)} = \boldsymbol{S}^{(k-1)}$;
   c. *Set* $\tilde{\boldsymbol{z}}^{(l_{k-1})}$ *to the last column of* $\boldsymbol{Z}^{(k-1)}$;
   d. *Set* $\tilde{\boldsymbol{Z}}^{(k)}$ *by removing the last column* $\tilde{\boldsymbol{z}}^{(l_{k-1})}$ *from* $\boldsymbol{Z}^{(k-1)}$;
   e. *Update* $\boldsymbol{Z}^{(k)} = [\tilde{\boldsymbol{Z}}^{(k)}\ \ \boldsymbol{z}]$;
   f. *Compute* $\boldsymbol{\tau}^{(k)} = \boldsymbol{z}^T\boldsymbol{g}^{(k+1)} - [\tilde{\boldsymbol{z}}^{(l_{k-1})}]^T\boldsymbol{g}^{(k)}$;
   g. *Compute* $\boldsymbol{p}_2^{(k)} = [\boldsymbol{S}^{(k-1)}]^T\boldsymbol{z}$ *and* $\tilde{\boldsymbol{p}}_4^{(k)} = [\tilde{\boldsymbol{Z}}^{(k)}]^T\boldsymbol{z}$;
   h. *Compute* $\boldsymbol{u}^{(k)}$ *using* **Eq. 27**;
   i. *Update* $\boldsymbol{A}^{(k)} = \boldsymbol{A}^{(k-1)}$;
   j. *Set* $\tilde{\boldsymbol{B}}^{(k-1)}$ *by removing the first column from* $\boldsymbol{B}^{(k-1)}$;
   k. *Set* $\tilde{\boldsymbol{C}}^{(k-1)}$ *by removing the first row and the first column from* $\boldsymbol{C}^{(k-1)}$;
   l. *Update* $\boldsymbol{B}^{(k)}$ *and* $\boldsymbol{C}^{(k)}$ *using* **Eq. 25** *and* **Eq. 26**;
5. *Else if* $\boldsymbol{l_{k-1}} < \boldsymbol{L_M}$ *(for Case-2), then*
   a. *Set* $\boldsymbol{l_k} = \boldsymbol{l_{k-1}} + 1$;
   b. *Compute* $\boldsymbol{p}_1^{(k)} = [\boldsymbol{S}^{(k-1)}]^T\boldsymbol{s}$, $\boldsymbol{p}_2^{(k)} = [\boldsymbol{S}^{(k-1)}]^T\boldsymbol{z}$, $\boldsymbol{p}_3^{(k)} = [\boldsymbol{Z}^{(k-1)}]^T\boldsymbol{s}$, *and* $\boldsymbol{p}_4^{(k)} = [\boldsymbol{Z}^{(k-1)}]^T\boldsymbol{z}$;

c. *Compute* $\gamma_s = s^T g^{(k+1)}$ *and* $\gamma_z = z^T g^{(k+1)}$;
d. *Update* $S^{(k)}$, $Z^{(k)}$, $A^{(k)}$, $B^{(k)}$, *and* $C^{(k)}$ *using* ***Eq. 28*** *through* ***Eq. 32****, respectively;*
e. *Set* $u_S^{(k-1)}$ *to the first* l_{k-1} *rows in* $u^{(k-1)}$ *and* $u_Z^{(k-1)}$ *to the last* l_{k-1} *rows in* $u^{(k-1)}$;
f. *Update* $u^{(k)}$ *using* ***Eq. 31***.

6. *Else (for Case-3)*
a. *Set* $l_k = L_M$;
b. *Remove the first column from* $S^{(k-1)}$ *and* $Z^{(k-1)}$;
c. *Remove the first row and first column from* $A^{(k-1)}$, $B^{(k-1)}$, *and* $C^{(k-1)}$;
d. *Compute* $p_1^{(k)} = [S^{(k-1)}]^T s$, $p_2^{(k)} = [S^{(k-1)}]^T z$, $p_3^{(k)} = [Z^{(k-1)}]^T s$, *and* $p_4^{(k)} = [Z^{(k-1)}]^T z$;
e. *Compute* $\gamma_s = s^T g^{(k+1)}$ *and* $\gamma_z = z^T g^{(k+1)}$;
f. *Update* $S^{(k)}$, $Z^{(k)}$, $A^{(k)}$, $B^{(k)}$, *and* $C^{(k)}$ *using* ***Eq. 28*** *through* ***Eq. 32*** *present, respectively;*
g. *Set* $u_S^{(k-1)}$ *to the second row through the* l_{k-1}*-th rows in* $u^{(k-1)}$;
h. *Set* $u_Z^{(k-1)}$ *to the last* $l_{k-1} - 1$ *rows in* $u^{(k-1)}$;
i. *Update* $u^{(k)}$ *using* ***Eq. 33***.

7. *Form the matrix* $P^{(k)}$ *using* ***Eq. 23***;
8. *Decompose* $B^{(k)} = L^{(k)} + E^{(k)} + U^{(k)}$;
9. *Form the matrix* $W_I^{(k)}$ *using* ***Eq. 5***;
10. *End*

The computational cost to update matrices and vectors used for solving the L-BFGS TRS as described in Algorithm-4 is at most $c_4 = 2n + 4mn$ (taking Case-2 in step 5 as an example), including the following 4 operations: (1) computing $\varepsilon^{(k)} = z^T s$, $\mu^{(k)} = z^T z$, and $\beta^{(k)} = s^T s$ in step 1 ($6n$ flops); (2) computing $p_1^{(k)} = [S^{(k-1)}]^T s$, $p_2^{(k)} = [S^{(k-1)}]^T z$, $p_3^{(k)} = [Z^{(k-1)}]^T s$, and $p_4^{(k)} = [Z^{(k-1)}]^T z$ in step 5(b) ($8l_{k-1}n = 4mn - 8n$ flops),; (3) computing $\gamma_s = s^T g^{(k+1)}$ and $\gamma_z = z^T g^{(k+1)}$in step 5(c) ($4n$ flops).

In this paper, we apply the same idea of reducing the dimension from n to m using the matrix inversion lemma as presented in the three papers [32, 36, 37]. However, the implementation is quite different. For the DGN optimization method, the $n \times m$ matrix $V^{(k)}$ (or the transpose of the sensitivity matrix) in **Eq. 15** is directly computed. Its dimensions (both the number of variables n and the number of observed data m) are fixed, i.e., they remain the same for all iterations. Therefore, we have to directly compute the $m \times m$ matrix $P^{(k)} = [V^{(k)}]^T V^{(k)}$ and the m-dimensional vector $u^{(k)} = [V^{(k)}]^T g^{(k)}$, with the computational cost $2m^2n + 2mn$ flops. Using the algorithm as summarized in Algorithm-4, the computational cost can be further reduced to $2n + 4mn$, by a factor of $l_k = m/2$, 5 to 20 roughly.

The Algorithm to Solve the L-BFGS TRS Using the Matrix Inversion Lemma

Given the trust region size $\Delta = \Delta^{(k)} > 0$, threshold of convergence $\delta_{cr} > 0$, maximum number of iterations allowed $N_{TRS,max} > 0$, the scaling factor $\alpha > \alpha_{cr} > 0$, $m \geq 0$, $n \times m$ matrix $V = [S^{(k)}, Z^{(k)}]$, $m \times m$ matrices $P = P^{(k)}$ and $W_I = [W^{(k)}]^{-1}$, m-dimensional vector $u = u^{(k)}$ and n-dimensional gradient vector $g = g^{(k)}$ evaluated at the current best solution $x^{(k)}$, both the Lagrange multiplier $\lambda^*$and the trust region search step $s^* = s(\lambda^*)$ can be solved from the TRS defined in **Eq. 8** using the matrix inversion lemma (MIL) as summarized in Algorithm-5.

Algorithm-5: Solving the L-BFGS TRS Using the Matrix Inversion Lemma

1. *Compute* $\|g\|$;
2. *Initialize* $l = 0$, $\lambda_0 = 0$;
3. *If* $m = 0$, *then*
a. *Set* $\lambda^* = 0$ *and* $s^* = -g/\alpha$ *if* $\|g\| \leq \alpha\Delta$;
b. *Set* $\lambda^* = \frac{\|g\|}{\Delta} - \alpha$ *and* $s^* = -g\frac{\Delta}{\|g\|}$ *if* $\|g\| > \alpha\Delta$;
c. *Goto step 11.*
4. *Compute* $D = \frac{1}{\alpha+\lambda_0}P - W_I$ *and Cholesky decomposition* $D = LL^T$;
5. *Solve* p^* *from* $Lp = u$ *and* v^* *from* $L^T v = p^*$;
6. *Compute* $u^T v^*$, $p = W_I v^*$, $p^T v^*$, *and* $\pi_0 = \pi(\lambda_0)$ *using* ***Eq. 20***;
7. *If* $\pi_0 \leq \Delta^2$, *then accept* $\lambda^* = \lambda_0$ *and goto step 10;*
8. *Set* $\delta = \left|1 - \frac{\sqrt{\pi_0}}{\Delta}\right|$;
9. *Repeat steps (a) through (j) below, until convergence* ($\delta \leq \delta_{cr}$ or $l > N_{TRS,max}$):
a. *Compute* $q = \frac{1}{(\alpha+\lambda_l)^2}Pv^*$;
b. *Solve* p^* *from* $Lp = q$ *and* w^* *from* $L^T w = p^*$;
c. *Compute* $u^T w^*$, $p = W_I w^*$, $p^T v^*$, *and* $\pi_l' = \pi'(\lambda_l)$ *using* ***Eq. 21****;*
d. *Compute* $\phi_l = \frac{1}{\sqrt{\pi_l}} - \frac{1}{\Delta}$ *and* $\phi_l' = -\frac{1}{2}\frac{\pi_l'}{(\pi_l)^{3/2}}$;
e. *Update* $\lambda_{l+1} = \lambda_l - \frac{\phi_l}{\phi_l'}$;
f. *Set* $l = l + 1$;
g. *Compute* $D = \frac{1}{\alpha+\lambda_l}P - W_I$ *and Cholesky decomposition* $D = LL^T$;
h. *Solve* p^* *from* $Lp = u$ *and* v^* *from* $L^T v = p^*$;
i. *Compute* $u^T v^*$, $p = W_I v^*$, $p^T v^*$, *and* $\pi_l = \pi(\lambda_l)$ *using* ***Eq. 20***;
j. *Set* $\delta = \left|1 - \frac{\sqrt{\pi_l}}{\Delta}\right|$.
10. *Compute* $p^* = Vv^*$ *and* $s^* = -\frac{1}{\alpha+\lambda^*}g + \frac{1}{(\alpha+\lambda^*)^2}p^*$;
11. *End*

Let N_{MIL} denote the number of iterations required for the L-BFGS TRS solver to converge. The total computational cost (flops) to solve the L-BFGS TRS using Algorithm-5 is, $c_5 = 4n + 2mn + \left(9m + 12m^2 + \frac{m^3}{3}\right)(N_{MIL} + 1)$, including the following eight operations: (1) computing $\|g\|$ in step 1 ($2n$ flops); (2) computing $D = \frac{1}{\alpha+\lambda_l}P - W_I$ and Cholesky

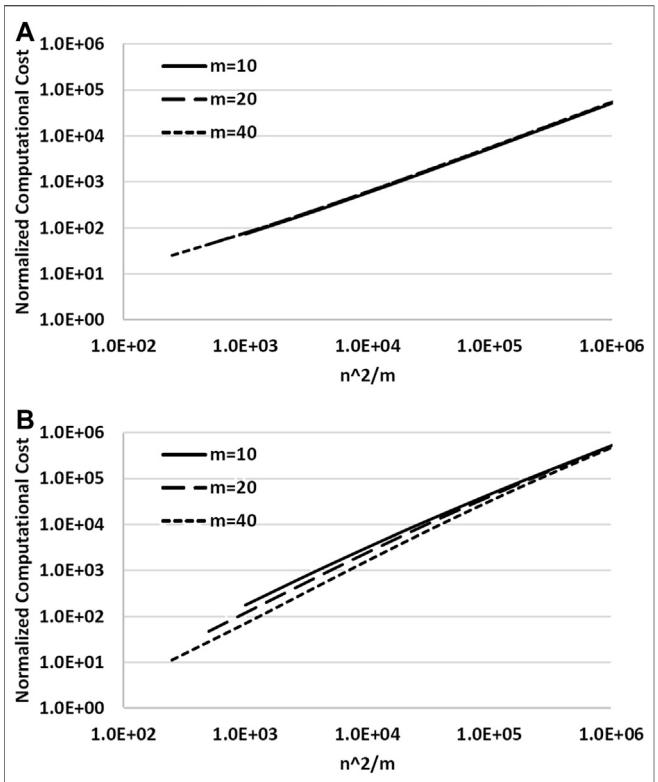

FIGURE 1 | Plots of the ratio of computational cost β vs. n^2/m, computed using ***Eq. 35*** by fixing $N_{DIQ} = N_{MIL} = 0$ in **(A)** and $N_{DIQ} = N_{MIL} = 10$ in **(B)**.

decomposition $D = LL^T$ in step 4 or step 9(g) ($2m^2 + \frac{m^3}{3}$ flops); (3) solving p^* from $Lp = u$, and v^* from $L^T v = p^*$ in step 5 or step 9(h) ($2m^2$ flops); (4) computing $u^T v^*$, $p = W_I v^*$ and $p^T v^*$ in step 6 or step 9(i) ($4m + 2m^2$ flops); (5) computing $q = \frac{1}{(\alpha+\lambda_I)^2} P v^*$ in step 9(a) ($m + 2m^2$ flops); (6) Solving p^* from $Lp = q$ and w^* from $L^T w = p^*$ in step 9(b) ($2m^2$ flops); and (7) computing $u^T w^*$, $p = W_I w^*$, and $p^T v^*$ in step 9(c) ($4m + 2m^2$ flops); and (8) computing $p^* = Vz^*$ and $s^* = -\frac{1}{\alpha+\lambda^*} g + \frac{1}{(\alpha+\lambda^*)^2} p^*$ in step 10 ($2mn + 2n$).

The total computational cost (flops) used to solve the L-BFGS TRS using Algorithm-5 together with Algorithm-4 is,

$$c_{MIL} = c_4 + c_5 = 6n + 6mn + \left(9m + 12m^2 + \frac{m^3}{3}\right)(N_{MIL} + 1) \tag{34}$$

It is straightforward to compute the normalized computational cost, or the ratio of computational cost of the DNR method (Algorithm-2) or the DIQ method (Algorithm-3) together with the algorithm that directly updates Hessian (Algorithm-1) over the MIL TRS solver using the matrix inversion lemma (Algorithm-5) and the efficient matrix updating algorithm (Algorithm-4),

$$\beta = \frac{c_{DIQ}}{c_{MIL}} = \frac{2mn^2 + (7 + 3m^2)n + \left(5n + 2n^2 + \frac{n^3}{3}\right)(N_{DIQ} + 1) + \mathbf{O}(m^3)}{6n + 6mn + \left(9m + 12m^2 + \frac{m^3}{3}\right)(N_{MIL} + 1)} \tag{35}$$

Because $n \gg m$ for large-scale problems, **Eq. 35** can be further simplified as,

$$\beta = \frac{c_{DIQ}}{c_{MIL}} \approx \frac{N_{DIQ} + 1}{18} \frac{n^2}{m} \tag{36}$$

For some problems the solution $s^* = s(0)$ is accepted with $N_{DIQ} = N_{MIL} = 0$. For other problems, it takes roughly $N_{DIQ} \approx N_{MIL} = 5 \sim 15$ iterations for a TRS solver to converge. **Figure 1A,B** illustrate the plots of β vs. n^2/m for different m by fixing $N_{DIQ} = N_{MIL} = 0$ and $N_{DIQ} = N_{MIL} = 10$, respectively. As shown in both **Figure 1A** and **Figure 1B**, the MIL TRS solver may reduce the computational cost by a factor $\beta = 10 \sim 10^5$ for $\frac{n^2}{m} = 100 \sim 10^6$.

NUMERICAL VALIDATION

We benchmarked the MIL TRS solver against the DNR (or DIQ) TRS solver on two well-known analytic optimization problems using analytical gradients: the Rosenbrock function and the Sphere function defined as follows,

$$f(x)_{\text{Rosenbrock}} = \sum_{i=1}^{n-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (1 - x_i)^2\right], \tag{37}$$

$$f(x)_{\text{Sphere}} = \sum_{i=1}^{n} x_i^2 \tag{38}$$

The Rosenbrock function defined in **Eq. 37** has one local minimum located at $x_1 = -1$ and $x_i = 1$ for $i = 2, 3, \ldots n$ with the objective function being 4 (approximately), and one global minimum located at $x_i = 1$ for $i = 1, 2, \ldots n$ with the objective function being 0. Occasionally, a local-search optimizer may converge to the local minimum.

We implemented different algorithms described in this paper in our in-house C++ optimization library (OptLib). We employ "Armadillo" as our foundational template-based C++ library for linear algebra (http://arma.sourceforge.net/). We ran all test cases reported on **Tables 2–5** on a virtual machine computer equipped with an Intel Xeon Platinum Processor at 2.60 MHz and 16.0 GB of RAM. We varied the number of parameters, n as powers of two and fixed twice the maximum number of snapshots, $M = 2L_M = 10$. **Tables 2–5** summarize the preliminary numerical results, including the value of the objective function, the gradient norm, the number of iterations, and the elapsed CPU time (s). The last column displays the resulting speedup, namely, the ratio of CPU times using the DNR (or DIQ) method over the MIL method. We prescribe the tolerance to stop the L-BFGS TRS solver for all the numerical experiments in this subsection as

TABLE 2 | Performance comparison of DNR vs. MIL methods testing on Rosenbrock function.

	DNR				MIL				
n	$f^{(k)}$	$g^{(k)}$	**Iters.**	**CPU**	$f^{(k)}$	$g^{(k)}$	**Iters.**	**CPU**	**Speedup**
8	4.4E-09	2.9E-05	41	2.9E-02	3.4E-09	2.5E-05	41	1.6E-02	1.8
16	1.7E-09	1.0E-05	62	3.8E-02	1.5E-09	1.9E-05	61	4.2E-02	0.9
32	1.3E-09	1.1E-05	75	4.6E-02	3.2E-09	2.2E-05	75	4.8E-02	1.0
48	1.3E-08	2.6E-05	72	9.3E-02	4.4E-09	2.0E-05	72	4.0E-02	2.3

TABLE 3 | Performance comparison of DIQ vs. MIL methods testing on Rosenbrock function.

	DIQ				MIL				
n	$f^{(k)}$	$g^{(k)}$	**Iters.**	**CPU**	$f^{(k)}$	$g^{(k)}$	**Iters.**	**CPU**	**Speedup**
16	1.2E-09	2.2E-05	51	2.4E-02	1.5E-09	1.9E-05	61	3.2E-02	0.8
32	1.0E-09	2.4E-05	61	4.3E-02	3.2E-09	2.2E-05	75	3.2E-02	1.4
64	8.0E-10	2.4E-05	75	1.3E-01	7.8E-09	2.9E-05	79	3.6E-02	3.5
128	4.5E-10	1.8E-05	82	3.0E-01	4.6E-09	2.8E-05	73	3.8E-02	7.9
256	2.2E-10	1.3E-05	87	1.0E+00	5.0E-09	3.0E-05	78	7.2E-02	13.9
512	4.0E+00	1.5E-05	84	5.5E+00	3.1E-10	9.6E-06	90	1.4E-01	39.2
1024	1.5E-09	2.8E-05	98	3.9E+01	7.5E-09	2.7E-05	94	4.2E-01	90.9

TABLE 4 | Performance comparison of DNR vs. MIL methods testing on Sphere function.

	DNR				MIL				
n	$f^{(k)}$	$g^{(k)}$	**Iters.**	**CPU**	$f^{(k)}$	$g^{(k)}$	**Iters.**	**CPU**	**Speedup**
16	5.4E-64	1.8E-33	39	2.2E-02	3.3E-65	3.1E-34	56	3.8E-02	0.6
32	1.1E-63	1.5E-33	66	4.0E-02	1.3E-65	1.6E-34	67	3.7E-02	1.1
64	4.4E-63	2.1E-33	92	1.4E-01	4.0E-67	2.2E-35	86	5.2E-02	2.8
128	8.4E-63	2.1E-33	128	7.3E-01	9.2E-66	6.5E-35	135	6.3E-02	11.6
256	1.4E-62	1.8E-33	188	2.8E+00	9.7E-63	1.5E-33	185	7.0E-02	39.1
512	9.6E-64	3.3E-34	265	1.9E+01	5.7E-64	2.7E-34	256	5.6E-01	32.9
1024	3.6E-62	1.4E-33	383	1.4E+02	1.8E-64	1.0E-34	371	2.7E+00	52.4
2048	2.1E-60	7.8E-33	525	8.4E+02	2.2E-66	7.8E-36	534	1.7E+01	50.2

TABLE 5 | Performance comparison of DIQ vs. MIL methods testing on Sphere function.

	DIQ				MIL				
n	$f^{(k)}$	$g^{(k)}$	**Iters.**	**CPU**	$f^{(k)}$	$g^{(k)}$	**Iters.**	**CPU**	**Speedup**
16	3.6E-67	4.6E-35	39	3.8E-02	1.6E-65	2.1E-34	56	4.0E-02	1.0
32	2.1E-68	6.5E-36	66	3.8E-02	7.4E-64	1.2E-33	67	4.4E-02	0.9
64	2.0E-65	1.4E-34	92	1.0E-01	5.3E-65	2.5E-34	86	4.8E-02	2.1
128	2.0E-65	9.9E-35	128	3.0E-01	4.4E-64	4.5E-34	135	5.8E-02	5.1
256	2.7E-65	8.0E-35	188	1.6E+00	2.6E-64	2.5E-34	185	8.8E-02	18.3
512	1.5E-67	4.2E-36	265	1.3E+01	7.9E-65	9.9E-35	256	6.5E-01	19.7
1024	4.4E-66	1.6E-35	383	6.5E+01	4.3E-65	5.0E-35	371	3.1E+00	21.1
2048	1.4E-65	2.0E-35	525	4.6E+02	1.6E-66	6.6E-36	534	1.7E+01	27.4

$\boldsymbol{\delta_{cr}} = 10^{-4}$ and the maximum number of iterations as $\boldsymbol{N_{TRS,max}} = 16$. We set the initial trust region size at $\Delta = 0.5$.

We noticed that the matrix $\boldsymbol{D}$ may become ill-conditioned for a few cases. Indeed, we could observe that the first entries on the diagonal are nearly zero. The same situation occurs for different optimization steps where $\boldsymbol{m}$ grows from 0, 2, 4, 6, up to ten. The Cholesky decomposition of such a matrix will fail, i.e., Armadillo throws an exception. We advise replacing the former with an LU-decomposition to handle the pivots, namely, entries on the diagonal of the lower triangular matrix, to be nearly zero. The rest of the algorithm remains the same.

Table 2 compares performance of the DNR method vs. the MIL methods for the Rosenbrock problem encompassing 8, 16, 32, and 48 parameters, respectively. Beyond those problem sizes, the DNR method could not converge to the solution. Indeed, the DNR method becomes relatively slow,

i.e., renders very large lambda values that imply tiny shifts. Often, after 2,048 iterations, the objective function is still greater than one. It seems that the DNR method is not robust enough to tackle the Rosenbrock function with more than 48 parameters. It appears that the MIL method slightly outperforms the DNR method for problems with fewer parameters.

Table 3 compares performance of the DIQ method vs. the MIL method. These numerical results confirm that the MIL method outperforms the DIQ method. As expected, the speedup improves with the increase of the problem size $\boldsymbol{n}$.

Table 4 shows similar results for the Sphere function problem. We could benchmark the DNR method against the MIL method with ranks up to 2,048 parameters. Again, the speedup increases with problem size. Finally, **Table 5** benchmarks the DIQ method against the MIL method testing on the Sphere function. The speedup reduces when compared to **Table 4** for the same ranks, but we still clearly see that the performance of the MIL solver exceeds its DIQ counterpart. Results shown in **Table 4** and **Table 5** also confirm that the DIQ method performs better than the DNR method. We believe that the DIQ TRS solver converges faster with smaller number of iterations than the DNR TRS solver (i.e., $\boldsymbol{N_{DIQ} < N_{DNR}}$).

The computational costs (in terms of CPU time) summarized in **Tables 2–5** are the overall costs for all iterations, including the cost of computing both value and gradient of the objective function in addition to the cost of solving the L-BFGS TRS. In contrast, the theoretical analysis results of computational cost for $\boldsymbol{\beta}$ in **Eq. 35** only count for the cost of solving the L-BFGS TRS in just one iteration. Therefore, numerical results listed in **Tables 2–5** are not quantitatively in agreement with the theoretically derived $\boldsymbol{\beta}$.

CONCLUSION

We can draw the following conclusions based on theoretical analysis and numerical tests:

1) Using the compact representation of the L-BFGS formulation in **Eq. 4**, the updated Hessian $\boldsymbol{H^{(k+1)}}$ is composed of a diagonal matrix $\boldsymbol{\lambda I_n}$ and a matrix $\boldsymbol{V^{(k)}W^{(k)}[V^{(k)}]^T}$ with low rank when $\boldsymbol{m < n}$.
2) Using the matrix inversion lemma, we are able to reduce the cost of solving the L-BFGS TRS by transforming the original equation with $\boldsymbol{n}$ variables to a new equation with $\boldsymbol{m}$ variables, by taking advantage of the low rank matrix updating feature.
3) Further reduction in computational cost is achieved by updating the vector $\boldsymbol{u^{(k)}}$ and matrices such as $\boldsymbol{S^{(k)}}$ and $\boldsymbol{Z^{(k)}}$, $\boldsymbol{A^{(k)}}$, $\boldsymbol{B^{(k)}}$, and $\boldsymbol{C^{(k)}}$, $\boldsymbol{P^{(k)}}$, and $\boldsymbol{[W^{(k)}]^{-1}}$ iteratively, which effectively avoids expensive operations such as matrix-matrix and matrix-vector multiplications.
4) Through seamless integration of matrix inversion lemma with the iterative matrix and vector updating approach, the newly proposed TRS solver for the limited-memory distributed BFGS optimization method performs much more efficiently than the DNR TRS solver.
5) The MIL TRS solver can obtain the right solution with acceptable accuracy, without increasing memory usage but reducing the computational cost significantly, as confirmed by numerical results on the Rosenbrock function and Sphere function.

DATA AVAILABILITY STATEMENT

The original contributions presented in the study are included in the article/Supplementary Material, further inquiries can be directed to the corresponding author.

AUTHOR CONTRIBUTIONS

GG, HF, and JV contributed to conception and algorithmic analysis of the study. HF, TW, FS, and CB contributed to design and numerical validations. GG and FH wrote the first draft of the article. All authors contributed to article revision, read, and approved the submitted version.

REFERENCES

1. Ai-Mudhafar WJ, Rao DN, and Srinivasan S. Robust Optimization of Cyclic CO2 Flooding through the Gas-Assisted Gravity Drainage Process under Geological Uncertainties. *J Pet Sci Eng* (2018). 166:490–509. doi:10.1016/j.petrol.2018.03.044
2. Alpak FO, Jin L, and Ramirez BA. Robust Optimization of Well Placement in Geologically Complex Reservoirs. in: Paper SPE-175106-MS Presented at the SPE Annual Technical Conference and Exhibition; September28–30 2015; Houston, TX (2015).
3. van Essen GM, Zandvliet MJ, Den Hof PMJ, Bosgra OH, Jansen JD, et al. Robust Optimization of Oil Reservoir Flooding. in: Paper Presented at the IEEE International Conference on Control Applications; Oct 4-6 2006; Munich, Germany (2006). p. 4–6.
4. Liu Z, and Forouzanfar F. Ensemble Clustering for Efficient Robust Optimization of Naturally Fractured Reservoirs. *Comput Geosci* (2018). 22:283–96. doi:10.1007/s10596-017-9689-1
5. Liu X, and Reynolds AC. Augmented Lagrangian Method for Maximizing Expectation and Minimizing Risk for Optimal Well-Control Problems with Nonlinear Constraints. *SPE J* (2016). 21(5). doi:10.1007/s10596-017-9689-1
6. Liu X, and Reynolds AC. Gradient-based Multi-Objective Optimization with Applications to Waterflooding Optimization. *Comput Geosci* (2016). 20:677–93. doi:10.1007/s10596-015-9523-6
7. Oliver DS, Reynolds AC, and Liu N. *Inverse Theory for Petroleum Reservoir Characterization and History Matching*. Cambridge, United Kingdom: Cambridge University Press (2008). doi:10.1017/cbo9780511535642
8. Tarantola A. *Inverse Problem Theory and Methods for Model Parameter Estimation*. Philadelphia, PA: SIAM (2005). doi:10.1137/1.9780898717921
9. Oliver DS. On Conditional Simulation to Inaccurate Data. *Math Geol* (1996). 28:811–7. doi:10.1007/bf02066348
10. Jansen JD. Adjoint-based Optimization of Multi-phase Flow through Porous Media - A Review. *Comput Fluids* (2011). 46(1):40–51. doi:10.1016/j.compfluid.2010.09.039
11. Li R, Reynolds AC, and Oliver DS. History Matching of Three-phase Flow Production Data. *SPE J* (2003). 8(4):328–40. doi:10.2118/87336-pa

12. Sarma P, Durlofsky L, and Aziz K. Implementation of Adjoint Solution for Optimal Control of Smart Wells. in: Paper SPE 92864 Presented at the SPE Reservoir Simulation Symposium Held in the Woodlands; Jan.-2 Feb; Woodlands, TX (2005). p. 31.
13. Gao G, and Reynolds AC. An Improved Implementation of the LBFGS Algorithm for Automatic History Matching. *SPE J* (2006). 11(1):5–17. doi:10.2118/90058-pa
14. Nocedal J, and Wright SJ. *Numerical Optimization*. New York City: Springer (1999).
15. Chen C, et al. Assisted History Matching Using Three Derivative-free Optimization Algorithms. in: Paper SPE-154112-MS Presented at the SPE Europec/EAGE Annual Conference Held in Copenhagen; June 4–7 2012; Copenhagen, Denmark (2012). p. 4–7.
16. Gao G, Vink JC, Chen C, Alpak FO, and Du K. A Parallelized and Hybrid Data-Integration Algorithm for History Matching of Geologically Complex Reservoirs. *SPE J* (2016). 21(6):2155–74. doi:10.2118/175039-pa
17. Gao G, Vink JC, Chen C, El Khamra Y, Tarrahi M, et al. Distributed Gauss-Newton Optimization Method for History Matching Problems with Multiple Best Matches. *Comput Geosciences* (2017). 21(5-6):1325–42. doi:10.1007/s10596-017-9657-9
18. Gao G, Wang Y, Vink J, Wells T, Saaf F, et al. Distributed Quasi-Newton Derivative-free Optimization Method for Optimization Problems with Multiple Local Optima. in: 17th European Conference on the Mathematics of Oil Recovery ECMOR XVII; September 2020; Edinburgh, United Kingdom (2020). p. 14–7.
19. Wild SM. *Derivative Free Optimization Algorithms for Computationally Expensive Functions*. [PhD thesis]. Ithaca (new york): Cornell University (2009).
20. Powell MJD. Least Frobenius Norm Updating of Quadratic Models that Satisfy Interpolation Conditions. *Math Programming* (2004). 100(1):183–215. doi:10.1007/s10107-003-0490-7
21. Zhao H, Li G, Reynolds AC, and Yao J. Large-scale History Matching with Quadratic Interpolation Models. *Comput Geosci* (2012). 17:117–38. doi:10.1007/s10596-012-9320-4
22. Audet C, and Dennis JE, Jr. Mesh Adaptive Direct Search Algorithms for Constrained Optimization. *SIAM J Optim* (2006). 17:188–217. doi:10.1137/040603371
23. Spall JC. *Introduction to Stochastic Search and Optimization*. Hoboken, NJ: John Wiley & Sons (2003). doi:10.1002/0471722138
24. Chen C, et al. EUR Assessment of Unconventional Assets Using Parallelized History Matching Workflow Together with RML Method. In: Paper Presented the 2016 Unconventional Resources Technology Conference; August 1–3 2016; San Antonio, TX(2016).
25. Chen C, Gao G, Li R, Cao R, Chen T, Vink JC, et al. Global-Search Distributed-Gauss-Newton Optimization Method and its Integration with the Randomized-Maximum-Likelihood Method for Uncertainty Quantification of Reservoir Performance. *SPE J* (2018). 23(05):1496–517. doi:10.2118/182639-pa
26. Chen C, Gao G, Ramirez BA, Vink JC, and Girardi AM. Assisted History Matching of Channelized Models by Use of Pluri-Principal-Component Analysis. *SPE J* (2016). 21(05):1793–812. doi:10.2118/173192-pa
27. Armacki A, Jakovetic D, Krejic N, Krklec Jerinkic N, et al. Distributed Trust-Region Method with First Order Models. in Paper Presented at the IEEE EUROCON-2019, 18th International Conference on Smart Technologies July 1–14 2019 Novi Sad, Serbia (2019). p. 1–4.
28. Moré JJ, and Sorensen DC. Computing a Trust Region Step. *SIAM J Sci Stat Comput* (1983). 4(3):553–72. doi:10.1137/0904038
29. Mohamed AW. Solving Large-Scale Global Optimization Problems Using Enhanced Adaptive Differential Evolution Algorithm. *Complex Intell Syst* (2017). 3:205–31. doi:10.1007/s40747-017-0041-0
30. Golub GH, and Van Loan CF. *Matrix Computations*. 4th ed. The Johns Hopkins University Press (2013).
31. Brust JJ, Leyffer S, and Petra CG. *Compact Representations of BFGS Matrices*. Preprint ANL/MCS-P9279-0120. Lemont, IL: ARGONNE NATIONAL LABORATORY (2020).
32. Gao G, Jiang H, Vink JC, van Hagen PPH, Wells TJ, et al. Performance Enhancement of Gauss-Newton Trust-Region Solver for Distributed Gauss-Newton Optimization Method. *Comput Geosci* (2020). 24(2):837–52. doi:10.1007/s10596-019-09830-x
33. Burdakov O, Gong L, Zikrin S, and Yuan Y. On Efficiently Combining Limited-Memory and Trust-Region Techniques. *Math Prog Comp* (2017). 9:101–34. doi:10.1007/s12532-016-0109-7
34. Burke JV, Wiegmann A, and Xu L. *Limited Memory BFGS Updating in a Trust-Region framework Tech. Rep*. Seattle, WA: University of Washington (2008).
35. Byrd RH, Nocedal J, and Schnabel RB. Representations of Quasi-Newton Matrices and Their Use in Limited Memory Methods. *Math Programming* (1994). 63:129–56. doi:10.1007/bf01582063
36. Gao G, Jiang H, van Hagen P, Vink JC, and Wells T. A Gauss-Newton Trust Region Solver for Large Scale History Matching Problems. *SPE J* (2017b). 22(6):1999–2011. doi:10.2118/182602-pa
37. Gao G, Saaf F, Vink J, Krymskaya M, Wells T, et al. Gauss-Newton Trust Region Search Optimization Method for Ill-Conditioned Least Squares Problem. in: ECMOR XVII 17th European Conference on the Mathematics of Oil Recovery; 14 September 2020; Edinburgh, UK (2020). p. 14–17.

Conflict of Interest: Authors GG, HF, and FS were employed by Shell Global Solutions (United States) Inc. Authors JV, TW and CB were employed by Shell Global Solutions International B.V.