

Interval Hessian-based Optimization Algorithm for Unconstrained Non-convex Problems

Ashutosh Sharma^a, Gauransh Dingwani^b, and Ishan Bajaj^{a*}

^a Indian Institute of Technology Kanpur, Department of Chemical Engineering, Kanpur, Uttar Pradesh, India

^b Indian Institute of Technology Roorkee, Department of Chemical Engineering, Roorkee, Uttarakhand, India

* Corresponding author: ibajaj@iitk.ac.in.

ABSTRACT

Second-order optimization algorithms that leverage the exact Hessian or its approximation have been proven to achieve a faster convergence rate than first-order methods. However, their applications on training deep neural networks models, partial differential equations-based optimization problems, and large-scale non-convex problems, are hindered due to high computational cost associated with the Hessian evaluation, Hessian inversion to find the search direction, and ensuring its positive-definiteness. Accordingly, we propose a new search direction based on an interval Hessian and incorporate it into a line-search framework. We apply our algorithm to a set of 210 problems and show that it converges to a local minimum for 70% of the problems. We also compare our algorithm with other approaches. We illustrate that our algorithm is competitive to other methods in finding a local minimum using a smaller number of $O(n^3)$ operations.

Keywords: Second-order optimization, Interval Hessian, Line-search framework, Non-Convex optimization.

1. INTRODUCTION

We consider the unconstrained optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x)$$

where $f: \mathbb{R}^n \rightarrow \mathbb{R}$ is a twice continuously differentiable function, and n is the number of variables. Several first- and second-order methods have been developed to solve the above problem. These algorithms are used in areas like process optimization, communications, machine learning, and signal processing. First-order algorithms, such as steepest descent, stochastic gradient descent, and various adaptive gradient methods, are particularly popular for large-scale problems due to their relatively low computational demands per iteration. Despite their prevalence, first-order methods are limited in that they only guarantee convergence to a first-order stationary point. Thus, these methods can converge to a saddle point rather than a local minimum.

In contrast, second-order optimization algorithms, which utilize the curvature information via the Hessian matrix, provide faster convergence and avoid saddle points. Newton's method, a classic second-order

approach, adjusts the gradient direction with a preconditioning matrix derived from the Hessian, achieving convergence to points that meet second-order optimality conditions.

Although Newton's method exhibits impressive convergence characteristics compared to first-order approaches, their broad application to large-scale optimization problems poses significant challenges. First, the memory requirement increases with the square of the number of variables. Second, the Hessian calculation can be expensive. Third, at each iteration, the search direction requires computing the inverse of the Hessian, which requires $O(n^3)$ operations and can be computationally expensive for large n . Fourth, Newton's direction for non-convex problems may not be descent at each iteration. However, nonconvexity is ubiquitous in several applications, including, training neural network models, reactor and separation network synthesis, process design, and computational chemistry.

To address these limitations, quasi-Newton methods have been developed as computationally feasible alternatives. Quasi-Newton methods, such as the BFGS algorithm [2], approximate the Hessian using gradients of the previous iteration and thus reduce computational

demands. Sub-sampled Newton methods [5] decrease costs by approximating the Hessian using a subset of data points. Despite these advances, balancing Hessian approximation accuracy with computational tractability remains a fundamental challenge.

Accordingly, we propose a new search direction based on an interval Hessian and incorporate it into a line-search framework. Our approach is inspired by α BB method proposed by Floudas and co-workers [1] to construct a convex underestimator of a twice-differentiable function over an interval $[x^L, x^U]$. Our approach involves using the Hessian of the α BB underestimator. There are three advantages of this approach. First, since the underestimator is convex, its Hessian will always be positive definite. Thus, the search direction is guaranteed to be descent. Second, the same Hessian matrix can be used if the iterates lie in the interval for which the interval Hessian is valid. This results in (1) avoiding computing Hessian at each iteration and (2) the search direction is computed by taking matrix-vector product instead of $O(n^3)$ operations that are typically needed in Newton and Quasi-Newton methods.

2. METHODOLOGY

2.1. Search Direction

In this article, we propose a variant of the Newton direction to ensure descent at each iteration for non-convex problems. We incorporate the search direction into a line-search framework:

$$x_{k+1} = x_k + \theta_k p_k$$

where x_k is the current iterate, x_{k+1} is the next iterate, θ_k is the step length, and p_k is the search direction. Our search direction is inspired by the α BB method developed for global optimization of non-convex problems. This method constructs a convex underestimator of a twice continuously differentiable function with the following form:

$$\mathcal{L}(x) = f(x) + \sum_{i=1}^n \alpha_i (x_i^L - x_i)(x_i^U - x_i)$$

where $\alpha_i \geq 0$. This underestimator is constructed by subtracting a positive term from $f(x)$, making $\mathcal{L}(x)$ a valid underestimator of $f(x)$ within $[x^L, x^U]$. The parameter α_i is calculated to ensure that $\mathcal{L}(x)$ is convex over this interval, which implies that the Hessian of $\mathcal{L}(x)$ is positive semi-definite. Thus, we choose p_k as follows:

$$p_k = -(B_k)^{-1} \nabla f_k \quad (1)$$

where $\nabla f_k = \nabla f(x_k)$ and $B_k = B_{k-1}$ if x_k lies in the interval for which the convex underestimator is valid; otherwise, new variable bounds are constructed and $B_k = \nabla^2 \mathcal{L}_k$. The Hessian of the α BB underestimator provides a localized

approximation of the objective function's curvature. The relationship between the Hessians of $\mathcal{L}(x)$ and $f(x)$ at $x = x_k$ is:

$$\nabla^2 \mathcal{L}_k = \nabla^2 f_k + 2\Delta$$

where Δ is a diagonal matrix with elements α_i . The matrix Δ is referred to as the diagonal shift matrix. For simplicity, we assume a uniform diagonal shift matrix, such that $\alpha_1 = \alpha_2 = \alpha_3 = \dots = \alpha_n = \alpha$, leading to:

$$\nabla^2 \mathcal{L}_k = \nabla^2 f_k + 2\alpha I$$

where I denotes the identity matrix. Based on the above equation, we conclude that the difference between the Hessian of the function and the α BB underestimator depends on α . Consequently, the accuracy of computing α determines the search direction quality, which in turn affects the performance of our algorithm.

Maranas and Floudas demonstrated [1] that the underestimator, $\mathcal{L}(x)$, is convex if and only if:

$$\alpha \geq \max \left\{ 0, -\frac{1}{2} \min_{i, x^L \leq x \leq x^U} \lambda_i (\nabla^2 f(x)) \right\}$$

where λ_i are the eigenvalues of $\nabla^2 f(x)$. Finding α by using the above equation is equivalent to solving a non-convex optimization problem to global optimality, which can require significant computational effort.

To overcome this difficulty, a lower bound on the minimum eigenvalue of the interval matrix, $[\nabla^2 f(x)] \supseteq \nabla^2 f(x), \forall x \in [x^L, x^U]$, is computed. This interval Hessian is then used in further α calculations. In subsequent sections, we explore various approaches to compute α by employing methods with $O(n^2)$ and $O(n^3)$ complexities.

2.2. α Calculation Methods

We use three methods to compute the value of lower bound on eigenvalues (λ_{min}) for a real interval matrix $[A] = ([\underline{a}_{ij}, \overline{a}_{ij}])$. We define following notations useful in computation of λ_{min} .

$$\underline{A} = (\underline{a}_{ij}) \text{ and } \overline{A} = (\overline{a}_{ij}), \Delta a_{ij} = (\overline{a}_{ij} - \underline{a}_{ij})/2$$

$$\widetilde{\Delta A} = (\widetilde{\Delta a}_{ij}), \widetilde{\Delta a}_{ij} = \begin{cases} 0 & \text{if } i = j, \\ \Delta a_{ij} & \text{otherwise} \end{cases}$$

$$A_M = (a_{M,i,j}), a_{M,i,j} = (\overline{a}_{ij} + \underline{a}_{ij})/2$$

$$\widetilde{A}_M = (\widetilde{a}_{M,i,j}), \widetilde{a}_{M,i,j} = \begin{cases} a_{ij} & \text{if } i = j, \\ a_{M,i,j} & \text{otherwise} \end{cases}$$

where, ΔA is the radius matrix, $\widetilde{\Delta A}$ is the modified radius matrix, A_M is the midpoint matrix, \widetilde{A}_M is the modified midpoint matrix, \underline{A} is the lower vertex matrix and \overline{A} is the upper vertex matrix. The various methods of $O(n^2)$ and $O(n^3)$ complexities to calculate the minimum eigenvalue are given below.

2.2.1 Approximate Gerschgorin: Complexity $O(n^2)$

$$\lambda_{\min}([A]) \geq \min_i \left[a_{ii} - \sum_{j \neq i} \max(|a_{ij}|, |\bar{a}_{ij}|) \right]$$

2.2.2. E-Matrix Method: Complexity $O(n^3)$

$$\lambda_{\min}([A]) \geq \lambda_{\min}(\widetilde{A}_M + E) - \rho(\widetilde{\Delta A} + |E|)$$

where E be an arbitrary symmetric matrix, whose value is chosen to be $\text{diag}(\Delta A)$ here, $\rho(M)$ denotes the spectral radius of a matrix M, i.e., the maximal absolute eigenvalue and $|E|$ denotes the absolute value taken elementwise.

2.2.3. Mori-Kokame Method: Complexity $O(n^3)$

$$\lambda_{\min}([A]) \geq \lambda_{\min}(\underline{A}) - \rho(\overline{A} - \underline{A})$$

2.3. Algorithm

Our algorithm employs a line search technique with the search direction details given in the Section 2.1. The overall framework is outlined in Algorithm 1. We choose two iteration counters, t and k . The former serves as the outer iteration counter, and the latter as the inner iteration counter.

We begin by randomly choosing an initial point x_0 such that it lies within the interval $[x_0^L, x_0^U]$. The search direction p_k (Eq. (1)) satisfies $\nabla f_k^T p_k < 0$, guaranteeing that it leads to a decrease in the function value. Thus, at each iteration, $f(x_{k+1}) < f(x_k)$.

Once the search direction is determined, we need to decide how far to move in the given direction by choosing an appropriate value of θ_k . We select θ_k that satisfies the Armijo condition:

$$f(x_k + \theta_k p_k) \leq f(x_k) + c_1 \theta_k \nabla f(x_k)^T p_k$$

where $c_1 \in (0, 1)$. This condition ensures that a sufficiently large decrease in the objective function is achieved. We apply a backtracking line search procedure to find θ_k that satisfies the Armijo condition.

Our approach leverages interval-based calculations to reduce the need to compute Hessian and its inverse at every iteration. At iteration k , if the iterate x_k lies outside the current interval $[x_t^L, x_t^U]$, we update the iteration counter t to $t + 1$ and construct an interval of size Δ around the iterate x_k by setting $x_{t+1}^L = x_k - \frac{\Delta}{2}$ and $x_{t+1}^U = x_k + \frac{\Delta}{2}$. After updating the interval, the interval Hessian is computed using interval arithmetic, the minimum eigenvalue of the interval Hessian is estimated using one of the methods mentioned in section 2.2, and $\nabla^2 L_t$ and its inverse are computed. Thus, at any iteration t , depending on the method used to estimate the minimum eigenvalue of the interval Hessian, either one or two $O(n^3)$ operations are performed. On the other hand, at iteration k , if the iterate x_k lies inside the current interval $[x_t^L, x_t^U]$, the

search direction is updated by taking the product of $(\nabla^2 L_t)^{-1}$ and ∇f_k . Conversely, in Newton's method, Hessian evaluation and its inverse is computed at every iteration. Note that $\nabla^2 L_t$ is a positive-definite matrix, and thus, the search direction is always guaranteed to be descent.

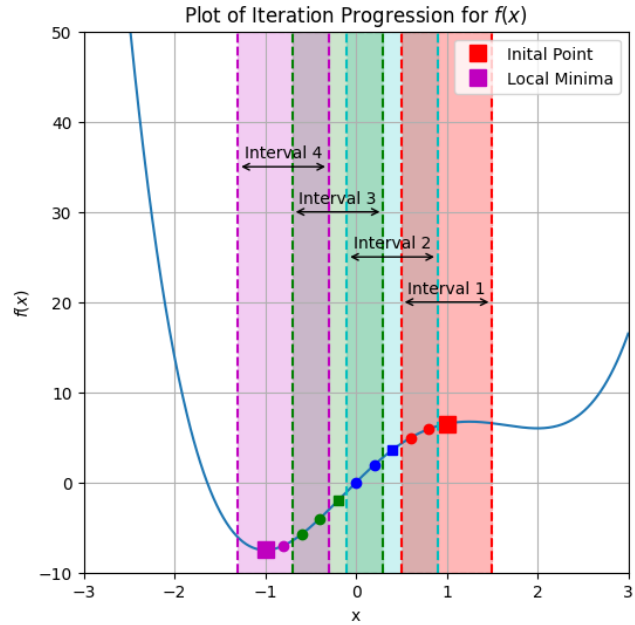


Figure 1. Graphical illustration of our proposed algorithm.

Figure 1 demonstrates the iterative mechanics of our approach using a one-dimensional example. The figure illustrates that if the iterates remain within an interval (denoted by a circle marker), the Hessian remains the same, and the search direction is obtained by the matrix-vector product. When an iterate exits the current interval, a new interval is formed around this iterate (denoted by a small square marker), and all the relatively more expensive computations occur.

The algorithm is implemented in C++. We use IBEX for interval Hessian and gradient evaluations and Eigen to compute the matrix inverse.

3. COMPETING ALGORITHMS

We compare the performance of our algorithm with the steepest descent, quasi-Newton and the Newton method. The steepest descent and quasi-Newton are popular optimization algorithms; their details can be found in a standard optimization textbook [2]. In Newton's method, the search direction, denoted by p_k^N , is given by:

$$p_k^N = -(\nabla^2 f_k)^{-1} \nabla f_k$$

At every iteration k , finding the search direction requires $O(n^3)$ operations. If the Hessian matrix is positive-definite at the current iterate, x_k , then the search

Algorithm 1 Interval Hessian based line-search method

Init. Choose $x_0 \in \mathbb{R}^n, c_1 \in (0,1), \rho \in (0,1), \epsilon_g \in (0,1), \epsilon_H \in (0,1), t = 0$;

for $k = 0, 1, 2, \dots$ **do**

Set $g_k = \nabla f(x_k)$;

if $\|g_k\| < \epsilon_g$ and $\lambda_{\min}(\nabla^2 f(x_k)) > \epsilon_H$ **then**

Terminate – Solution found;

end if

if $k > iter_{max}$ **then**

Terminate

if $k = 0$ **then** Set $H_k = \nabla^2 f(x_k)$

Hessian Modification (Compute B_t, x_t^L, x_t^U)

for $i = 0, 1, 2, \dots$ **do**

$x_{t,i}^L = x_{k,i} - \Delta/2$

$x_{t,i}^U = x_{k,i} + \Delta/2$

end for

Compute interval Hessian $[H_t]$ for the interval $[x_t^L, x_t^U]$ and calculate λ_{\min} using methods suggested above

Set $B_k = \nabla^2 L_t + \lambda_{\min} I_n$

$t \leftarrow t + 1$

else

if $x_k \in [x_t^L, x_t^U]$ **then**

Use the pre-computed Hessian $B_k = B_{k-1}$

else

Hessian Modification (Compute B_t, x_t^L, x_t^U)

$t \leftarrow t + 1$

end if

end if

Find the search direction by solving $p_k = -B_t^{-1} g_k$

Compute step length θ_k satisfying Armijo condition and set $x_{k+1} = x_k + \theta_k p_k$

end for

direction is guaranteed to be descent. On the other hand, if the Hessian is indefinite, then it needs to be modified, requiring additional computations. A simple idea to modify the Hessian is to find a parameter $\tau > 0$ such that $\nabla^2 f_k + \tau I$ is positive definite. Algorithm 2 describes a method to iteratively increase the value of τ until the matrix becomes positive definite, verified by performing Cholesky decomposition.

Algorithm 2 Brute-force Diagonal Shift

Init. Choose $\tau_0 = 0$;

for $i = 0, 1, 2, \dots$ **do**

Cholesky Factorization $LL^T = \nabla^2 f_k + \tau_i I_n$

if factorization is completed successfully

$\lambda_{\min} \leftarrow \tau_i$ and **terminate**

else $\tau_{i+1} \leftarrow \tau_i + 1$

end if

end for

4. RESULTS

We illustrate the performance of our algorithm on two illustrative examples and conduct extensive numerical experiments over a comprehensive set of 210 problems [3]. Specifically, we compare the performance based on the number of Hessian evaluations and $O(n^3)$ operations. Since we estimate the interval Hessian at an iteration t , we count equivalent to two Hessian evaluations are performed.

4.1. Illustrative Examples

4.1.1. Example 1

The objective function is of the following form:

$$f(x_1, x_2) = \sum_{j=1}^{17} \sum_{k=1}^{m_j} (t_{j,k} - [\exp(-2jx_2)(0.49 - x_1) + x_1])^2$$

where m_j denotes the number of groups at index j and $t_{j,k}$ is a constant. Figure 2(a) shows the paths taken by

the iterates generated by the steepest descent, quasi-newton BFGS and three variants of our algorithm to calculate α based on the methods described in Section 2.2. For this problem, we used $\Delta = 0.1$. It can be observed that the steepest descent can get stuck at regions with small gradients and is unable to reach a local minimum in 10000 iterations. The BFGS method followed a path similar to the steepest descent but failed to find a descent direction after 9 iterations. This occurs because the quasi-Newton method can fail to find a descent direction for non-convex problems if the Armijo condition is used to find the step size θ (refer chapter 6 of [2]). On the other hand, all our algorithm variants take almost the same path and converge to the same solution. The Gerschgorin-based method requires 8 Hessian evaluations, 8 $O(n^3)$ operations, and 466 iterations. E-matrix based method requires 317, and the Mori-Kokame method converges in 1125 iterations, requiring 7 and 9 Hessian evaluations and 14 and 18 $O(n^3)$ operations respectively.

4.1.2. Example 2

We minimize the function of the form:

$$f(x_1, x_2) = C_A x_1 + C_B x_2 + C_C - \left\{ \sum_{(i,j) \in S} C_{ij} x_1^i x_2^j + \frac{C_D}{1+x_2} + C_E e^{C_F x_1 x_2} \right\}$$

where C_A, \dots, C_F and C_{ij} are constants. Figure 2(b) shows the paths taken by the iterates generated by brute force method and three variants of our algorithm corresponding to α calculation methods described in Section 2.2. To illustrate the effect of Hessian evaluations, we use the sleep function with 2 seconds when the Hessian is evaluated. Figure 2(c) shows that all the algorithm variants converge to a better solution in less time than the brute force method.

4.2. Benchmarking with Data Profiles

We construct data profiles [4] to systematically benchmark the algorithms. A data profile assesses the absolute performance of an algorithm. It determines the ability of an algorithm to solve an optimization problem within some desirable performance metric. For each problem-algorithm pair (l, a) , the data profile $d_a(\delta)$ indicates the proportion of problems solved within δ metric:

$$d_a(\delta) = \frac{1}{|L|} \text{size}\{l \in L : m_{l,a} \leq \delta\}$$

where $|L|$ represents the total number of problems and $m_{l,a}$ is the performance measure. We evaluate the performance of our algorithm based on Hessian evaluations and the number of $O(n^3)$ operations, as these are the most computationally expensive steps in a second-order optimization method.

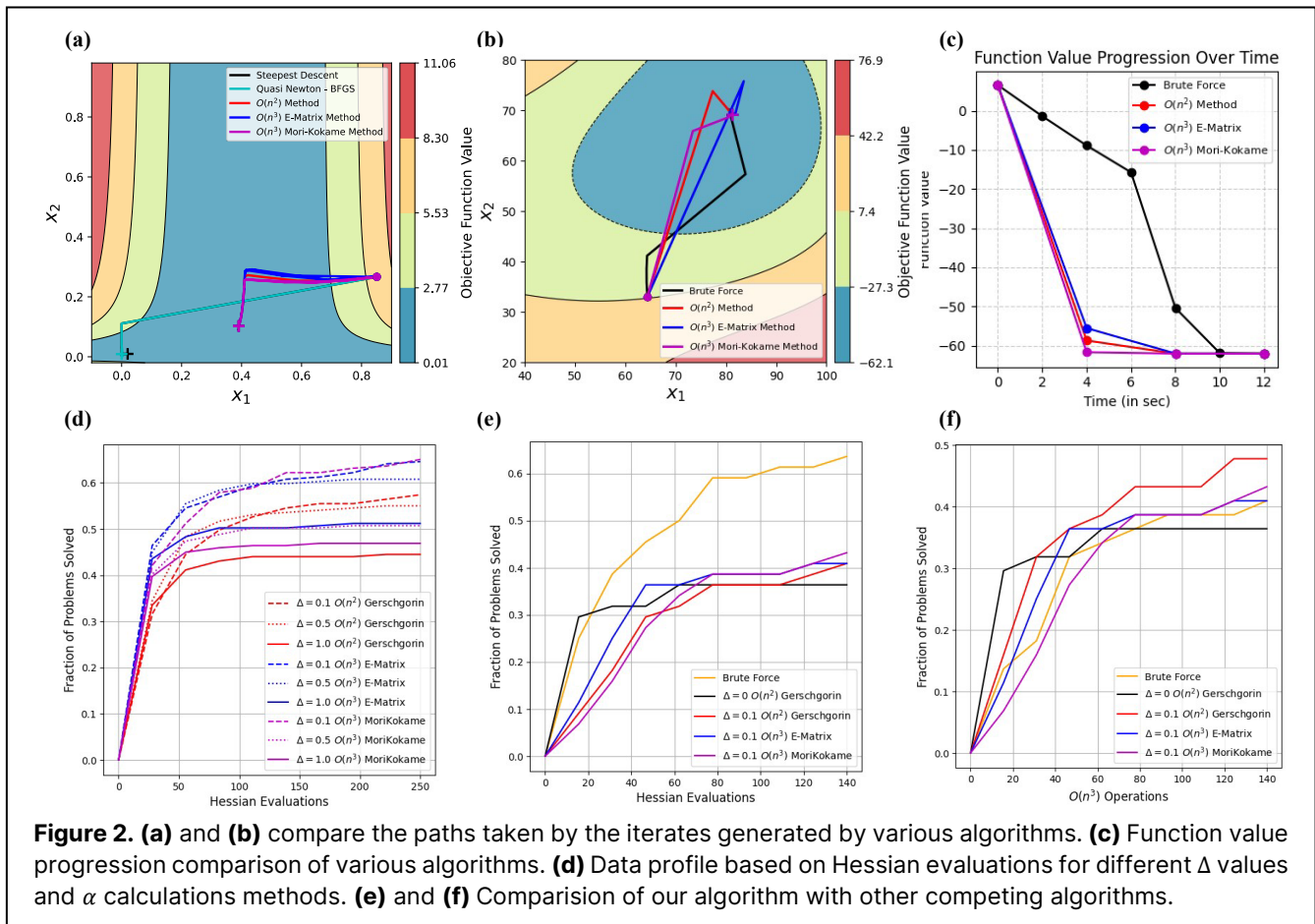
A key trade-off in the proposed approach lies in the choice of interval size (Δ). Larger intervals reduce the

computations associated with Hessian evaluations and inversion but potentially sacrifice accuracy in α estimates. Conversely, smaller intervals enhance precision and robustness in capturing curvature information but at the cost of increased computational overhead. Additionally, the accuracy of α depends on the method employed. We observed that the Mori-Kokame and E-matrix methods provide more accurate α estimates than the Gerschgorin method. While the former two approaches require $O(n^3)$ operations, the latter requires $O(n^2)$ operations.

Accordingly, to study these trade-offs, we consider six variants of our algorithm by choosing three Δ values (0.1, 0.5, 1) and three methods to estimate the minimum eigenvalue (section 2.2). We apply the algorithms to a set of 210 problems. The results of the computational experiments are shown in Figure 2 (d), which shows that our algorithm could solve 70% of the problems in less than 300 Hessian evaluations. For 13% of the instances terminated, the algorithm reached the maximum number of iterations (10000), 1.9% stopped due to the time limit (3600 seconds), 9% stopped due to small step size, and 0.9% of cases converged to an iterate where the norm of the gradient was below the threshold but failed to satisfy the minimum eigenvalue condition of the Hessian. The remaining 4.8% of cases involved scenarios where a local minimum did not exist or the algorithm started from points with excessively large function values exceeding the range of data type double. We observe that the E-matrix and Mori-Kokame methods with $\Delta = 0.1$ outperform the Gerschgorin method and converge in a smaller number of iterations.

Next, we compare the performance of our algorithm with Newton's method, employing Algorithm 2 to ensure a positive-definite Hessian matrix. To illustrate the effectiveness of the interval-based approach, we also implement the Gerschgorin method of finding α with $\Delta = 0$. For this computational study, we selected the problems requiring Hessian modifications. Figures 2(e) and 2(f) show the data profiles based on the number of Hessian evaluations and $O(n^3)$ operations, respectively.

We make three key observations. First, the interval-based method for finite values of Δ performs similarly to the Gerschgorin method with $\Delta = 0$ in terms of Hessian evaluations; however, the former method requires a smaller number of $O(n^3)$ operations. Second, when the computational budget is more than 40 $O(n^3)$ operations, the Gerschgorin-based interval method outperforms the other methods, illustrating the effectiveness of the search direction in navigating the non-convex landscape. Finally, Newton's method with brute force method of modifying the Hessian solves the most problems in least number of Hessian evaluations. Still, it requires a higher number of $O(n^3)$ operations. This can be explained by the fact that the brute-force method adds τ just enough to make the Hessian positive definite, resulting in a better



search direction. However, this increases $O(n^3)$ operations due to the Cholesky decomposition performed for positive-definiteness verification.

5. CONCLUSIONS

In this study, we introduced an interval-based optimization algorithm to find a search direction guaranteed to be descent to find a local minimum of non-convex problems. We show through numerical experiments that our algorithm finds a local minimum for a greater number of problems using less $O(n^3)$ operations compared to Newton's method. The results highlight the potential of our approach to improve the scalability of second-order optimization for complex, real-world applications.

DIGITAL SUPPLEMENTARY MATERIAL

Supplementary material is uploaded in Living Archive for Process Systems Engineering with ID LAPSE:2025.0005.

REFERENCES

1. Adjiman C, Dallwig S, Floudas C, Neumaier A. A global optimization method, α BB, for general twice-differentiable constrained NLPs — I. Theoretical

advances. *Comput Chem Eng* 22:1137–1158 (1998) [https://doi.org/10.1016/S0098-1354\(98\)00003-3](https://doi.org/10.1016/S0098-1354(98)00003-3)

2. Nocedal, J., & Wright, S. J. NUMERICAL OPTIMIZATION. In Springer eBooks (2006).
3. Puranik, Y., & Sahinidis, N. V. Bounds tightening based on optimality conditions for nonconvex box-constrained optimization. *J Glob Opt*, 67(1–2), 59–77 (2016) <https://doi.org/10.1007/s10898-016-0491-8>
4. Moré, J. J., & Wild, S. M. Benchmarking Derivative-Free optimization Algorithms. *SIAM J Opt*, 20(1), 172–191 (2009). <https://doi.org/10.1137/080724083>
5. Xu, P., Roosta, F., & Mahoney, M. W. Second-order Optimization for Non-convex Machine Learning: an Empirical Study. In *SIAM eBooks* (pp. 199–207). (2020) <https://doi.org/10.1137/1.9781611976236.23>

© 2025 by the authors. Licensed to PSEcommunity.org and PSE Press. This is an open access article under the creative commons CC-BY-SA licensing terms. Credit must be given to creator and adaptations must be shared under the same terms. See <https://creativecommons.org/licenses/by-sa/4.0/>

