Electronic Research Archive

https://www.aimspress.com/journal/era

ERA, 33(7): 4091–4118.
DOI: 10.3934/era.2025183
Received: 23 December 2024
Revised: 30 May 2025
Accepted: 10 June 2025
Published: 02 July 2025

Research article

A matrix-free DFP-like optimization method for problems arising from compressive sensing

Aliyu Muhammad Awwal[1,2,3], **Sulaiman M. Ibrahim**[4,5,*], **Issam A. R. Moghrabi**[6,*], **Mahmoud M. Yahaya**[2], **Aishatu I. Ahmad**[1,2], **Semiu Oladipupo Oladejo**[1,2], **Karimov Javlon Kuzievich**[7] **and Aceng Sambas**[3]

1 Department of Mathematical Sciences, Faculty of Science, Gombe State University, Nigeria
2 GSU-Mathematics for Innovative Research (GSU-MIR) Group, Gombe State University, Nigeria
3 Faculty of Informatics and Computing, Universiti Sultan Zainal Abidin, Besut 22200, Malaysia
4 School of Quantitative Sciences, Universiti Utara Malaysia, Sintok 06010, Kedah, Malaysia
5 Faculty of Education and Arts, Sohar University, Sohar 311, Oman
6 Department of Information Systems and Technology, Kuwait Technical College, Kuwait, Kuwait
7 Department of Higher and Applied Mathematics Mathematics, Tashkent State University of Economics, Uzbekistan

* **Correspondence:** Email: sulaimancga@gmail.com, i.moghrabi@ktech.edu.kw.

Abstract: This paper introduces a matrix-free variant of the Davidon-Fletcher-Powell (DFP) method for unconstrained optimization problems with applications in compressive sensing and image restoration. The main contribution lies in the new search direction incorporating a scaling parameter that ensures the satisfaction of the sufficient descent condition, independent of the line search conditions. A rigorous convergence analysis guarantees the boundedness and theoretical validity of the proposed method. Comprehensive numerical experiments on benchmark unconstrained optimization test problems and compressive sensing problems demonstrate the efficiency and robustness of the algorithm. Specifically, in image restoration tasks, our method outperforms CG-DESCENT, MDL, and NSMA, achieving a 100% success rate compared to 95.8%, 84.5%, and 53.5%, respectively. Additionally, results on computational time, relative error, and PSNR confirm the superior performance of the proposed approach. These findings establish the proposed method as a competitive alternative for large-scale optimization problems.

Keywords: three-term conjugate gradient algorithms; optimization models; global convergence; line search procedure; signal processing

1. Introduction

One of the classes of iterative methods for solving general unconstrained optimization problems given by min $h(x)$, where $x \in \mathbb{R}^n$ is a family of quasi-Newton methods. These methods were designed to overcome some of the limitations associated with Newton's method, such as the computation of second derivatives of the objective function in each iteration. Let $\gamma(x)$ denote the gradient of the real-valued function $h(x)$. The working rule of quasi-Newton methods is to take in an initial guess, say x_0, and continue updating it via the following relation

$$x^{(k+1)} = x^{(k)} + e^{(k)}d^{(k)}, \quad k = 0, 1, 2, \dots, \tag{1.1}$$

where $e^{(k)} \in (0, \infty)$ is known as the step size. The quasi-Newton search direction $d^{(k)}$ takes the following form

$$d^{(k)} = -(P^{(k)})^{-1}\gamma^{(k)}, \tag{1.2}$$

where the matrix $P^{(k)} \in \mathbb{R}^{n\times n}$ is an approximation of the Hessian of the objective function at $x^{(k)}$, and it is structured in such a way that it satisfies the following secant equation:

$$P^{(k+1)}s^{(k)} = \mho^{(k)}, \quad \text{with } s^{(k)} = x^{(k+1)} - x^{(k)} \text{ and } \mho^{(k)} = \gamma^{(k+1)} - \gamma^{(k)}. \tag{1.3}$$

The optimization problem frequently occurs in engineering, management science, economics, and other industrial applications [1–3]. The function h is a real-valued function whose domain $\mathbb{R}^n$ is usually assumed to be continuous and bounded from below, and its first derivative is available [4].

On the other hand, the inverse of the Hessian could be approximated with some quasi-Newton updating formulas, thereby avoiding the need to compute the inverse of matrices in every iteration [5,6]. By setting $V^{(k)} := (P^{(k)})^{-1}$, one of the formulas for updating the matrix $V^{(k)}$ is given by

$$V^{(k)} := V^{(k-1)} + \frac{s^{(k-1)}s^{(k-1)T}}{s^{(k-1)T}\mho^{(k-1)}} - \frac{V^{(k-1)}\mho^{(k-1)}\mho^{(k-1)T}V^{(k-1)}}{\mho^{(k-1)T}V^{(k-1)}\mho^{(k-1)}}. \tag{1.4}$$

The formula (1.4) is named after Davidon, Fletcher, and Powell and is often referred to as the DFP update formula. One of the shortcomings of many quasi-Newton updating formulas, including (1.4), inherited from Newton's method, is the computation and storage of matrices in every iteration. In several works, it has been discussed that the inner product $s^{(k-1)T}\mho^{(k-1)}$ may assume the value zero, which in turn will make the ratio in the second term of (1.4) to be undefined. These are two major limitations associated with the update formula (1.4).

Using the idea of [7], the formula (1.4) can be modified and incorporated into the search direction (1.2) such that it mimics the popular conjugate gradient (CG) method. This means that the computation and storage of matrices can be avoided.

The CG method is also an iterative method for solving unconstrained optimization problems using the relation (1.1) where the computation and storage of matrices, as well as the computing of second derivatives, are completely avoided in every iteration [8,9].

The CG search direction is usually structured to take the form of two terms or three terms as the case may be [10]. The three terms take the following form

$$d^{(k)} = \begin{cases} -\gamma^{(k)}, & k = 0, \\ -\gamma^{(k)} + \beta^{(k)}d^{(k-1)} + \theta^{(k)}a^{(k)}, & k \geq 1, \end{cases} \tag{1.5}$$

where $\beta^{(k)}$ and $\theta^{(k)}$ are suitable parameters and $a^{(k)}$ is a suitable vector.

As we have stated earlier, the quasi-Newton search direction (1.2) can be restructured to incorporate desirable features of the conjugate gradient (CG) search direction. A notable example is the work by Andrei [11], where a scaled memoryless BFGS method was designed to generate directions similar to those of CG, aiming to enhance global convergence for large-scale optimization. Similarly, Zhang et al. [7] modified the classical BFGS update by integrating a mechanism that mimics the PRP (Polak-Ribière-Polyak) CG direction, leading to improved global convergence properties and satisfactory numerical performance.

However, these approaches are not without limitations. The method of Andrei [11] focuses primarily on scaling techniques without a flexible mechanism to dynamically adapt curvature information, while the modification by Zhang et al. [7] relies on a fixed structural form, which may not fully capture the benefits of CG strategies in all scenarios, particularly for ill-conditioned or highly nonlinear problems.

To address these issues, our approach introduces a scaling parameter, $q^{(k)} > 0$, directly into the DFP (Davidon–Fletcher–Powell) update formula, as shown in (1.6):

$$V^{(k)} := q^{(k)}V^{(k-1)} + \frac{s^{(k-1)}s^{(k-1)T}}{s^{(k-1)T}\mho^{(k-1)}} - \frac{V^{(k-1)}\mho^{(k-1)}\mho^{(k-1)T}V^{(k-1)}}{\mho^{(k-1)T}V^{(k-1)}\mho^{(k-1)}}. \tag{1.6}$$

This generalization retains the structure of the classical DFP (1.4) formula, recovered when $q^{(k)} = 1$ for all k, but introduces a flexible parameter that allows the search direction to adapt dynamically based on spectral information. This feature distinguishes our method from existing approaches, enabling it to inherit global convergence guarantees while potentially improving practical performance on challenging problems.

Next is to determine the scaling parameter $q^{(k)}$, in such a way that the search direction $d^{(k)}$ in (1.6) satisfies the very important sufficient descent condition

$$\gamma^{(k)T}d^{(k)} \leq -c\|\gamma^{(k)}\|^2, \quad c > 0. \tag{1.7}$$

Now, following the approach in [11], and setting $V^{(k-1)} \equiv I$ in (1.6), where I is an identity matrix, we have

$$V^{(k)} = q^{(k)} \times I + \frac{s^{(k-1)}s^{(k-1)T}}{s^{(k-1)T}\mho^{(k-1)}} - \frac{\mho^{(k-1)}\mho^{(k-1)T}}{\mho^{(k-1)T}\mho^{(k-1)}}, \tag{1.8}$$

From (1.2) and (1.8), we have

$$d^{(k)} = -q^{(k)}\gamma^{(k)} - \frac{s^{(k-1)T}\gamma^{(k)}}{s^{(k-1)T}\mho^{(k-1)}}s^{(k-1)} + \frac{\mho^{(k-1)T}\gamma^{(k)}}{\|\mho^{(k-1)}\|^2}\mho^{(k-1)}. \tag{1.9}$$

Comparing the two search directions (1.5) and (1.9), it is clear that the latter has the same characteristic as the former with $\beta^{(k)}$ and $\theta^{(k)}$ taking the following structures: $\beta^{(k)} = \frac{s^{(k-1)T}\gamma^{(k)}}{s^{(k-1)T}\mho^{(k-1)}}$ and $\theta^{(k)} = \frac{\mho^{(k-1)T}\gamma^{(k)}}{\|\mho^{(k-1)}\|^2}$.

In this paper, our objective is to explore the computational advantages of the search direction (1.9), as well as to determine the value of the scaling parameter $q^{(k)}$ such that (1.7) is satisfied. Furthermore, the direction will be restructured to be well-defined for all k. Moreover, a set of experiments will be carried out to explore the numerical features of the proposed search direction. The experiment will be divided into three as follows:

- Examine the impact of the scaling parameter on the search direction's numerical performance.
- Determine the numerical performance of the search direction on a collection of some test problems compared to selected state-of-the-art CG algorithms.
- Explore the applicability of the search direction on image restoration problems.

The rest of the paper will be structured as follows: In the next section, the proposed method and the algorithm are discussed. Detailed numerical experiments are presented in Section 4. Sections 5 and 6 demonstrate the applicability of the proposed method in signal and image processing, respectively. Finally, the concluding remark is presented in the last Section.

2. Proposed algorithm and analysis

Consider the proposed search direction (1.9) whose scaling parameter $q^{(k)} > 0$ is to be determined. By the definition of the vector $\mho^{(k-1)}$, its inner product with $s^{(k-1)}$ could yield any real number, including zero, which will invalidate (1.9) in its current form. To rectify this abnormality, we redefine the search direction as follows:

$$d^{(k)} := -q^{(k)}\gamma^{(k)} - \frac{s^{(k-1)T}\gamma^{(k)}}{\max\{s^{(k-1)T}\mho^{(k-1)}, \vartheta\}}s^{(k-1)} + \frac{\mho^{(k-1)T}\gamma^{(k)}}{\max\{\|\mho^{(k-1)}\|^2, \vartheta\}}\mho^{(k-1)}, \tag{2.1}$$

where $s^{(k-1)} = x^{(k)} - x^{(k-1)}$, $\mho^{(k-1)} = \gamma^{(k)} - \gamma^{(k-1)}$, and ϑ is a very small positive real number. To ensure (2.1) is sufficiently descending, we take its inner product with the gradient of the objective function at $x^{(k)}$. This yields

$$\begin{aligned}
\gamma^{(k)T}d^{(k)} &= -q^{(k)}\|\gamma^{(k)}\|^2 - \frac{(s^{(k-1)T}\gamma^{(k)})^2}{\max\{s^{(k-1)T}\mho^{(k-1)}, \vartheta\}} + \frac{(\mho^{(k-1)T}\gamma^{(k)})^2}{\max\{\|\mho^{(k-1)}\|^2, \vartheta\}} \\
&\leq -q^{(k)}\|\gamma^{(k)}\|^2 + \frac{(\mho^{(k-1)T}\gamma^{(k)})^2}{\|\mho^{(k-1)}\|^2} \\
&\leq -q^{(k)}\|\gamma^{(k)}\|^2 + \frac{\|\mho^{(k-1)}\|^2\|\gamma^{(k)}\|^2}{\|\mho^{(k-1)}\|^2} \\
&\leq -(q^{(k)} - 1)\|\gamma^{(k)}\|^2.
\end{aligned}$$

This means that we must take $q^{(k)}$ to be at least $r + 1$, where $r > 0$. Without loss of generality, for all k, we set $q^{(k)} = r + 1$, which implies that (2.1) satisfies the descent condition (1.7). That is, for all k,

$$\gamma^{(k)T}d^{(k)} \leq -r\|\gamma^{(k)}\|^2. \tag{2.2}$$

Remark 2.1. *It should be noted that the proposed search direction* (2.1) *satisfies* (1.7) *independently of any line search strategy. This suggests that* (2.1) *can be implemented using any available well-defined line search strategy.*

Based on the above Remark 2.1, we adopt the following line search techniques for the implementation of the search direction (2.1):

- Wolfe line search: Determine $e^{(k)}$ that satisfies the following inequalities

$$h(x^{(k)} + e^{(k)}d^{(k)}) \leq h(x^{(k)}) + \delta e^{(k)}\gamma^{(k)T}d^{(k)}, \tag{2.3}$$

$$\gamma(x^{(k)} + e^{(k)}d^{(k)})^T d^{(k)} \geq \sigma\gamma^{(k)T}d^{(k)}, \tag{2.4}$$

where $0 < \delta < \sigma < 1$ [5,12,13].

- Strong Wolfe line search [11]: Determine $e^{(k)}$ that satisfies the following inequalities:

$$h(x^{(k)} + e^{(k)}d^{(k)}) \leq h(x^{(k)}) + \delta e^{(k)}\gamma^{(k)T}d^{(k)}, \tag{2.5}$$

$$|\gamma(x^{(k)} + e^{(k)}d^{(k)})^T d^{(k)}| \leq -\sigma\gamma^{(k)T}d^{(k)}. \tag{2.6}$$

Next, we give the details of the algorithm named **M**odified **D**avidon **F**letcher **P**owell **A**lgorithm (mDFP)

Algorithm 1 (mDFP).

Stage 0. Given $x^{(0)} \in \mathbb{R}^n$, choose suitable $r, \vartheta > 0$. Set $d^{(0)} = -\gamma^{(0)} = -\nabla h^{(0)}$, and $k := 0$.
Stage 1. Check if $||\gamma^{(k)}|| \leq \epsilon$, then stop.
Stage 2. Compute $e^{(k)}$ using either (2.3) and (2.4), or (2.5) and (2.6).
Stage 3. Update the new point based on (1.1), if $||\gamma^{(k)}|| \leq \epsilon$, terminate the program.
Stage 4. Update the search direction

$$d^{(k)} := -(r+1)\gamma^{(k)} - \frac{s^{(k-1)T}\gamma^{(k)}}{\max\{s^{(k-1)T}\mho^{(k-1)},\ \vartheta\}}s^{(k-1)} + \frac{\mho^{(k-1)T}\gamma^{(k)}}{\max\{||\mho^{(k-1)}||^2,\ \vartheta\}}\mho^{(k-1)}, \tag{2.7}$$

where $\mho^{(k-1)} = \gamma^{(k)} - \gamma^{(k-1)}$ and $s^{(k-1)} = x^{(k)} - x^{(k-1)}$.
Stage 5. Go to stage 1 with $k := k + 1$.

Remark 2.2. *It is clear that the search direction* (2.1) *is well-defined. Now, based on the fact that the two (2) line search techniques considered are well-defined, we conclude that Algorithm 1 is well-defined.*

Assumption 2.3. *In the convergence analysis of most CG algorithms, the following presumptions are essential and always necessary.*

Ai. The objective $h(x)$ is bounded on the level set $\mathscr{B} = \{x \in \mathbb{R}^n | h(x) \leq h(x^{(0)})\}$.
Aii. Let $\widehat{\mathscr{B}}$ be a convex neighborhood of $\mathscr{B}$. The gradient $\gamma(x)$ is Lipschitzian on $\widehat{\mathscr{B}}$. This means we can always have a positive number L for which

$$||\gamma(x) - \gamma(y)|| \leq L||x - y||, \ \forall x, y \in \widehat{\mathscr{B}}. \tag{2.8}$$

Remark 2.4. *It is not difficult to deduce from Assumption 2.3 that there exist some constants ω and $\widehat{\omega}$ such that*

$$||\gamma(x^{(k)})|| \leq \omega, \quad \forall x^{(k)} \in \widehat{\mathscr{B}}, \tag{2.9}$$

$$||x^{(k)} - x^{(k-1)}|| \leq \widehat{\omega}, \quad \forall x^{(k)},\ x^{(k-1)} \in \widehat{\mathscr{B}}. \tag{2.10}$$

Furthermore, since the objective function is decreasing as $k \to +\infty$, then from the Assumption 2.3, we deduce that the sequence of iterates generated by Algorithm 1 is contained in a bounded region and hence, $\{x^{(k)}\}$ is convergent.

Lemma 2.5. *Let $\{d^{(k)}\}$ be the sequence of search directions generated by Algorithm 1. Suppose that the gradient γ is Lipschitzian at $x^{(k)}$ for each k, then there exists $\overline{\omega}$ such that*

$$||d^{(k)}|| \leq \overline{\omega}, \quad \forall\ k = 0, 1, 2, \cdots. \tag{2.11}$$

Proof. For $k = 0$, we have $\|d^{(0)}\| = \|\gamma^{(0)}\| \leq \omega$. On the other hand, since $s^{(k-1)} = x^{(k)} - x^{(k-1)}$, then for $k = 1, 2, 3, \cdots$, we have

$$\begin{aligned}
\|d^{(k)}\| &\leq (r+1)\|\gamma^{(k)}\| + \left|\frac{s^{(k-1)T}\gamma^{(k)}}{\max\{s^{(k-1)T}\mho^{(k-1)}, \vartheta\}}\right|\|s^{(k-1)}\| + \left|\frac{\mho^{(k-1)T}\gamma^{(k)}}{\max\{\|\mho^{(k-1)}\|^2,\ \vartheta\}}\right|\|\mho^{(k-1)}\| \\
&\leq (r+1)\|\gamma^{(k)}\| + \frac{\|s^{(k-1)}\|\|\gamma^{(k)}\|}{\vartheta}\|s^{(k-1)}\| + \frac{\|\mho^{(k-1)}\|\|\gamma^{(k)}\|}{\|\mho^{(k-1)}\|^2}\|\mho^{(k-1)}\| \\
&= (r+1)\|\gamma^{(k)}\| + \frac{\|s^{(k-1)}\|^2\|\gamma^{(k)}\|}{\vartheta} + \|\gamma^{(k)}\| \\
&\leq (r+1)\omega + \frac{\widehat{\omega}^2\omega}{\vartheta} + \omega := \overline{\omega},
\end{aligned}$$

where the last inequality follows from (2.9) and (2.10).

Lemma 2.6. *Let $\{d^{(k)}\}$ be the sequence of sufficient descent search direction generated by Algorithm 1, then*

$$\sum_{k=0}^{\infty} \frac{(\gamma^{(k)T}d^{(k)})^2}{\|d^{(k)}\|^2} < +\infty. \tag{2.12}$$

Proof. Zoutendijk's work [14] provides an easy way to deduce the proof.

Theorem 2.7. *Suppose that Assumption 2.3 is true. Let $\gamma^{(k)}$ represent the objective function's gradient at $x^{(k)}$, and let Algorithm 1 generate the sequence of iterates $\{x^{(k)}\}$. Then*

$$\liminf_{k\to\infty} \|\gamma^{(k)}\| = 0. \tag{2.13}$$

Proof. In contrast, if (2.13) is not true, then there must be a constant $c > 0$ such that

$$\|\gamma^{(k)}\| \geq c, \quad k \geq 0. \tag{2.14}$$

Next, taking the square of both sides of (2.2) yields

$$r^2\|\gamma^{(k)}\|^4 \leq (\gamma^{(k)T}d^{(k)})^2. \tag{2.15}$$

Now by dividing both sides of (2.15) by $\|d^{(k)}\|^2$, we get

$$r^2\frac{\|\gamma^{(k)}\|^4}{\|d^{(k)}\|^2} \leq \frac{(\gamma^{(k)T}d^{(k)})^2}{\|d^{(k)}\|^2}. \tag{2.16}$$

Finally, by (2.11) and (2.14), and taking the summation of (2.16) over $k = 0, 1, 2, \ldots$, yields

$$\sum_{k=0}^{\infty} \frac{(\gamma^{(k)T}d^{(k)})^2}{\|d^{(k)}\|^2} \geq r^2 \sum_{k=0}^{\infty} \frac{\|\gamma^{(k)}\|^4}{\|d^{(k)}\|^2} \geq r^2 \sum_{k=0}^{\infty} \frac{c^4}{\overline{\omega}^2} = +\infty.$$

This shows a contradiction with (2.12) and hence, the conclusion (2.13) holds.

3. Numerical experiments

In this section, we present the numerical results obtained by implementing the mDFP direction to a set of standard unconstrained optimization benchmark problems. The objective is to evaluate the effectiveness and efficiency of the proposed direction compared to other classical directions, including:

- CG_DESCENT: Hager and Zhang algorithm [15].
- MDL: Algorithm of Zhang et al. [16].
- NSMA: Self-scaling quasi-Newton algorithm by [17].

The proposed mDFP algorithm is implemented with $r = 0.1, \quad \vartheta = 10^{-20}$, and Wolfe parameters $\sigma = 10^{-3}$ and $\delta = 10^{-4}$.

To comprehensively assess the performance of the proposed mDFP algorithm, the study considered a diverse set of benchmark functions from Andrie [18] and CUTEr library, including:

- FLETCHER: A function with a challenging minimization landscape.
- DENSCHNF: A function known to be computationally intensive because of its dense Hessian matrices, whose formula is defined as:

$$h(x) = \sum_{i=1}^{n-1} (x_i^2 + x_{i+1}^2 - 1)^2 + (x_i - x_{i+1})^2.$$

The function is non-convex and has multiple local minima because of its quadratic terms. The gradient of this function is computed as:

$$\nabla h(x) = \begin{pmatrix} 4x_1(x_1^2 + x_2^2 - 1) + 2(x_1 - x_2) \\ 4x_2(x_1^2 + x_2^2 - 1) + 2(x_2 - x_3) - 2(x_1 - x_2) \\ 4x_3(x_2^2 + x_3^2 - 1) + 2(x_3 - x_4) - 2(x_2 - x_3) \\ \vdots \\ 4x_{n-1}(x_{n-2}^2 + x_{n-1}^2 - 1) + 2(x_{n-1} - x_n) - 2(x_{n-2} - x_{n-1}) \\ 4x_n(x_{n-1}^2 + x_n^2 - 1) - 2(x_{n-1} - x_n). \end{pmatrix}.$$

These benchmark problems represent a variety of characteristics, ensuring a thorough evaluation. All experiments were conducted on an Intel Core i7-10750H machine whose specifications include a 2.60 GHz processor, 16 GB RAM, and Windows 10 Pro operating system; the software used was MATLAB R2023a. The performance of each algorithm was evaluated based on three metrics: Number of iterations (NOI) required by an algorithm to converge to the solution, number of function evaluations (NOF), the value of the objective function at the solution; and computational Time (CPU time), the total time taken by an algorithm to reach convergence, which is measured in seconds. The stopping criteria parameters for all algorithms, including the proposed mDFP algorithm, were set as follows:

- Tolerance: 1×10^{-6},
- Maximum iterations: 2000,

or if the code fails to execute due to low memory.

3.1. *Effect of line search methods*

The line search methods are crucial in determining the step length at every iteration of the optimization procedure, directly influencing the convergence and overall performance. We conducted the initial set of experiments using two popular line search methods, the Strong Wolfe and Weak Wolfe conditions, to evaluate the efficiency of the proposed direction. The comparison was performed on the selected problems that cover various problem sizes and complexities. Performance was assessed based on the NOI, NOF, and CPU time required to attain predefined termination criteria. To ensure a comprehensive and fair comparison, the performance profile tool introduced by Dolan and More [19] was employed. This tool graphs the proportion of functions solved within a given tolerance, which allows for a visual comparison of the efficiency of each line search across all test cases, as presented in Figures 1–3 below:

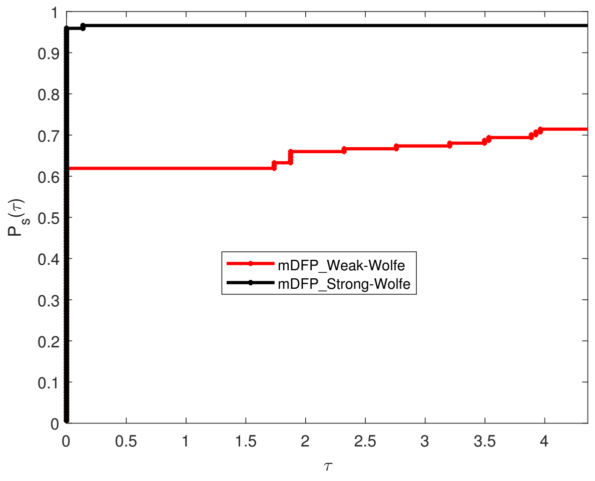

Figure 1. Performance profiles for NOI based on Strong Wolfe and Weak Wolfe conditions.

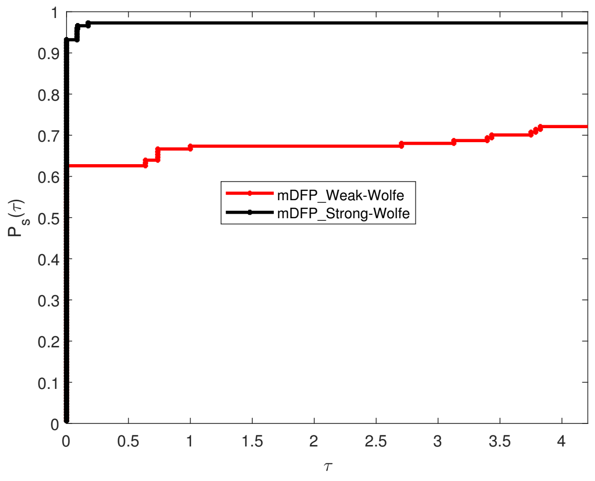

Figure 2. Performance profiles for NOF based on Strong Wolfe and Weak Wolfe conditions.

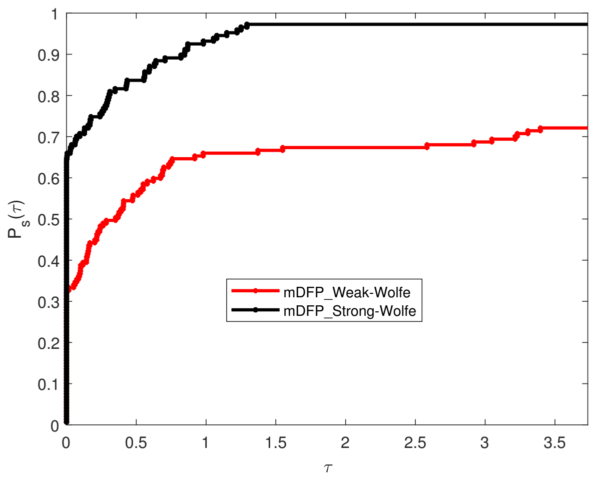

Figure 3. Performance profiles for CPU time based on Strong Wolfe and Weak Wolfe conditions.

The results, as illustrated in Figures 1–3, indicate that the Strong Wolfe line search demonstrates superior performance across the majority of the test functions. Specifically, it consistently required fewer function evaluations and iterations compared to the Weak Wolfe condition, resulting in significantly faster convergence. This improvement is particularly evident in large-scale problems and those characterized by ill-conditioning or non-convexity. One of the key advantages of the Strong Wolfe condition is its ability to maintain a precise balance between the curvature condition and the sufficient descent condition (Armijo condition). Unlike the Weak Wolfe condition, which permits a looser enforcement of curvature constraints, the Strong Wolfe condition imposes a stricter requirement on the search direction, ensuring more stable and controlled updates. This characteristic is particularly beneficial for problems where gradient information fluctuates sharply or where traditional methods struggle to maintain descent, as observed in several of the test cases.

Additionally, the impact of the Strong Wolfe condition on computational efficiency is evident in the reported CPU times. In problems where function evaluations are expensive, a reduction in unnecessary evaluations translates directly to lower computational cost. The figures demonstrate that the Strong Wolfe condition achieves a favorable balance by reducing function calls while maintaining rapid convergence, making it more suitable for large-scale applications. Another notable observation is the performance difference in problems with highly nonlinear structures. In such cases, the Weak Wolfe condition sometimes led to inefficient steps, requiring additional iterations to correct direction choices. The Strong Wolfe condition, on the other hand, ensured a more reliable update mechanism, preventing unnecessary oscillations and reducing the total number of iterations. This robustness makes it a practical choice for real-world optimization problems where stability and efficiency are critical.

Based on these findings, the Strong Wolfe strategy was selected as the preferred approach for further experimentation with the proposed search direction. Its ability to consistently yield lower iteration counts, fewer function evaluations, and reduced computational time establishes it as a highly effective strategy in the context of unconstrained optimization.

3.2. Effect of the parameter value (r)

The parameter r introduced in the direction plays an important role in influencing the balance between the correction terms and gradient direction. Adjusting the value of r, the study can control the weight given to the direction of the gradient versus the correction term that involves the size of the previous step and the gradient differences.

Given the importance of the parameter r in influencing the direction performance, a range of values for r was evaluated to choose the most efficient value for our experiments. In particular, the study considered the following values: $r = 0.1$, $r = 0.5$, $r = 1$, $r = 2, r = 5$, $r = 10$, and $r = 100$. These range values were selected to cover a broad spectrum of search behaviors, ranging from strong emphasis on correction terms ($r = 100$) to moderate influence of correction terms ($r = 0.5$). The selection of the optimal value for r was based on performance metrics, including NOI, NOF, and overall convergence behavior between the selected functions. Performance profiles were once again used to compare the results for each parameter value r. The value that led to the best balance between the accuracy of the solution and fast convergence will be selected for further experimentation with the proposed search direction. Figures 4–6 present the visual illustration of the performance of the proposed direction using the different values of r.

Figures 4–6 clearly indicated that $r = 0.5$ was the most suitable choice, followed by $r = 100$. In particular, $r = 0.5$ outperformed the other values of r in terms of efficiency and robustness. In most cases, it demonstrated faster convergence and required fewer iterations on average compared to larger values of r, including $r = 10$, $r = 2$, or $r = 5$, which tended to slow down the convergence due to an over reliance on the correction terms. In addition, $r = 0.5$ struck a better balance between the incorporation of correction terms and the gradient information, leading to more efficient and stable search directions. The superior performance of $r = 0.5$ indicated that it provides an adequate balance between adjusting the search path and exploring the gradient direction based on information from previous iterations. As a result, $r = 0.5$ was selected and considered for the next stage of the experiment.

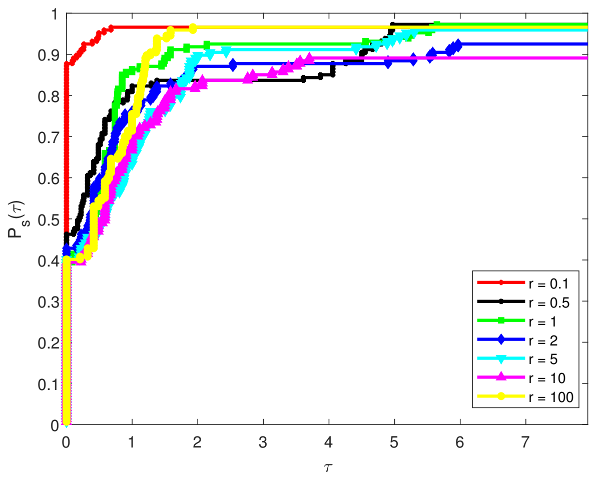

Figure 4. Performance profiles for NOI based on different parameter values (r)

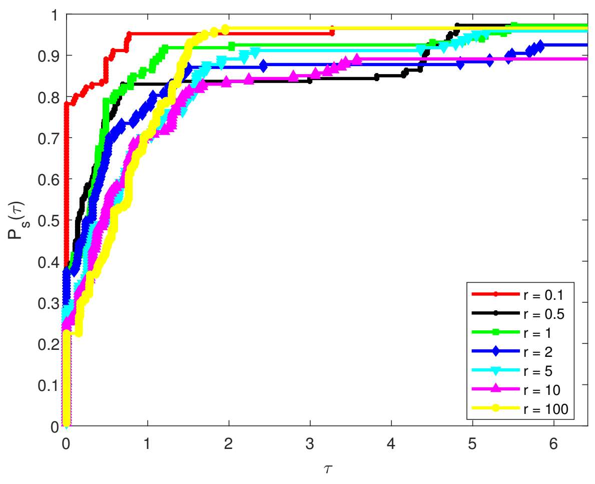

Figure 5. Performance profiles for NOF based on different parameter values (r).

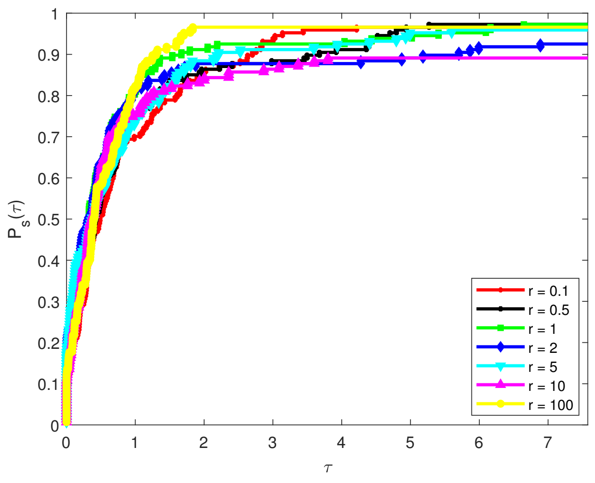

Figure 6. Performance profiles for CPU time based on different parameter values (r).

3.3. Computational efficiency

This part demonstrates the computational efficiency of the proposed direction compared to other existing directions based on the NOI, NOF, and the CPU time needed by the algorithms to minimize a function by varying the initial points and dimensions. Tables 1–4 present detailed performance results to demonstrate the scalability of all algorithms, with consistent performance across functions of different dimensions and sizes.

Table 1. Performance comparison based on NOI, NOF, and CPU time for problems 1–36 .

				mDFP			CG_DESCENT			MDL			NSMA		
S/N	**Test Functions**	**DIM**	**Initial Point**	**NI**	**FE**	**CPU**	**NI**	**FE**	**CPU**	**NI**	**FE**	**CPU**	**NI**	**FE**	**CPU**
P1	PRICE 4	2	(2,-2)	91	266	0.0012	51	159	0.0018	***	***	***	***	***	***
P2	BOOTH	2	(3,3)	5	11	0.0011	2	5	0.0016	2	5	0.0021	***	***	***
P3	BOOTH	2	(11,11)	4	9	0.0016	2	5	0.0009	2	5	0.0009	***	***	***
P4	BIGGSBI	2	(3,3)	1	3	0.0208	1	3	0.0056	1	3	0.0037	1	3	0.0494
P5	BIGGSBI	2	(11,11)	1	3	0.0048	1	3	0.0029	1	3	0.0042	1	3	0.0048
P6	El-Attar-Vidyasagar-Dutta	2	(3,3)	11	50	0.0016	12	51	0.0023	***	***	***	71	122	0.0047
P7	El-Attar-Vidyasagar-Dutta	2	(5,5)	12	56	0.001	11	64	0.0005	***	***	***	***	***	***
P8	Aluffi-Pentini	2	(14,14)	5	23	0.0013	9	24	0.0011	10	39	0.0015	***	***	***
P9	Aluffi-Pentini	2	(3,3)	5	18	0.0005	10	19	0.0004	10	26	0.0005	14	29	0.0008
P10	TEST	3	(3,3)	4	36	0.0022	61	263	0.0096	14	80	0.0051	***	***	***
P11	TEST	3	(4,4)	6	39	0.0031	49	103	0.0076	15	97	0.0041	***	***	***
P12	Tridiagonal White and Holst	2	(1.001,1.001)	3	10	0.0008	4	8	0.0013	3	8	0.001	***	***	***
P13	Tridiagonal White and Holst	2	(1.01,1.01)	65	135	0.0007	9	18	0.0004	6	14	0.0013	***	***	***
P14	SIX HUMP	2	(-2,-2)	4	18	0.0007	6	20	0.0003	10	37	0.0004	***	***	***
P15	SIX HUMP	2	(2,2)	4	18	0.0004	6	20	0.0004	10	37	0.0004	***	***	***
P16	MATYAS	2	(1,1)	1	8	0.0014	63	128	0.0005	1	8	0.0004	2	4	0.0009
P17	MATYAS	2	(0,5,0.5)	1	8	0.0006	59	120	0.0004	1	8	0.0005	2	4	0.0012
P18	BRENT	2	(3,3)	1	3	0.0017	1	3	0.0004	1	3	0.0004	1	3	0.0014
P19	BRENT	2	(11,11)	1	3	0.0003	1	3	0.0006	1	3	0.0005	1	3	0.0005
P20	TRECANNI	2	(10,…,10)	3	18	0.0005	8	28	0.0004	11	56	0.0003	***	***	***
P21	TRECANNI	2	(30,…,30)	5	23	0.0009	11	42	0.0005	16	118	0.0004	***	***	***
P22	DIAGONAL 3	2	(2,2)	6	17	0.0027	12	21	0.0063	5	11	0.003	18	26	0.0272
P23	DIAGONAL 3	2	(3,3)	5	17	0.0018	6	12	0.0025	5	10	0.0023	21	32	0.003
P24	Rotated Ellipse 2	2	(0.5,-1)	4	9	0.0004	4	8	0.0006	7	12	0.0004	13	21	0.0005
P25	Rotated Ellipse 2	2	(5,-5)	1	3	0.0005	1	3	0.0004	1	3	0.0005	1	3	0.0005
P26	POWER	2	(1,1)	2	10	0.0004	15	31	0.0005	6	19	0.0004	12	32	0.0004
P27	POWER	2	(15,15)	2	10	0.0004	18	37	0.0007	6	19	0.0006	609	14804	0.0004
P28	ZETTL	2	(1,1)	19	65	0.0006	20	35	0.0004	11	26	0.0004	***	***	***
P29	ZETTL	2	(-1,1)	25	95	0.0004	29	57	0.0009	14	45	0.0003	***	***	***
P30	ZETTL	2	(-1,-1)	25	95	0.0007	29	57	0.0004	14	45	0.0005	***	***	***
P31	DIAGONAL 9	2	(-.5,-.5)	3	9	0.0016	10	27	0.0036	***	***	***	2000	3005	0.1789
P32	DIAGONAL 9	2	(1,1)	3	9	0.0009	13	37	0.005	***	***	***	***	***	***
P33	DIAGONAL 2	2	(1,1)	6	21	0.0022	11	14	0.0028	4	12	0.0021	16	28	0.0219
P34	DIAGONAL 2	4	(1,1,1,1)	11	41	0.0032	23	24	0.002	10	30	0.0034	19	34	0.0055
P35	DIAGONAL 1	2	(3,3)	6	18	0.0015	8	15	0.0024	8	15	0.003	16	34	0.0181
P36	DIAGONAL 1	4	(3,3,3,3)	17	42	0.003	14	29	0.003	13	31	0.0033	30	52	0.0039

Table 2. Performance comparison based on NOI, NOF, and CPU time for problems 37–72 .

				mDFP			CG_DESCENT			MDL			NSMA		
S/N	**Test Functions**	**DIM**	**Initial Point**	**NI**	**FE**	**CPU**	**NI**	**FE**	**CPU**	**NI**	**FE**	**CPU**	**NI**	**FE**	**CPU**
P37	Quadratic QF1	2	(-0.011,0.011)	2	5	0.0157	3	6	0.0024	3	6	0.0024	9	14	0.0151
P38	Quadratic QF1	4	(-0.011,...,0.011)	3	21	0.0023	9	36	0.0024	***	***	***	***	***	***
P39	DQUARTC	4	(0,1,0,1)	2	7	0.0004	4	9	0.0009	4	9	0.0007	***	***	***
P40	DQUARTC	4	(0,0.1,0,0.1)	2	7	0.0004	3	7	0.0004	3	7	0.0005	***	***	***
P41	HAGER	2	(2,2)	5	15	0.0005	6	11	0.0004	7	13	0.0006	10	20	0.0006
P42	HAGER	4	(2,2,2,2)	9	23	0.0004	11	18	0.0007	8	17	0.0006	15	27	0.0003
P43	HAGER	20	(2,...,2)	23	52	0.0004	15	30	0.0004	15	32	0.0004	25	42	0.0008
P44	DIAGONAL 5	10	(0.5,...,0.5)	2	8	0.0013	8	9	0.0042	2	8	0.0036	5	8	0.06
P45	DIAGONAL 5	30	(0.5,...,0.5)	2	8	0.0017	8	9	0.0031	2	8	0.0024	4	8	0.0034
P46	DIAGONAL 5	50	(0.5,...,0.5)	2	8	0.0019	8	9	0.0018	2	8	0.002	4	8	0.0026
P47	SUMSQUARE	10	(3,...,3)	3	54	0.0004	***	***	***	6	159	0.0003	15	69	0.0004
P48	SUMSQUARE	20	(1,....,1)	3	53	0.0004	***	***	***	***	***	***	20	103	0.0004
P49	SUMSQUARE	50	(1,....,1)	2	35	0.0004	***	***	***	***	***	***	39	241	0.0009
P50	HIMMELBH	4	(0.5,...,0.5)	6	14	0.0005	4	9	0.0006	4	9	0.0007	***	***	***
P51	HIMMELBH	20	(0.5,...,0.5)	6	13	0.0008	5	11	0.0004	5	11	0.0005	***	***	***
P52	HIMMELBH	50	(0.5,...,0.5)	6	13	0.0005	5	11	0.0006	5	11	0.0006	15	24	0.0005
P53	HIMMELBH	100	(0.5,...,0.5)	6	13	0.0005	5	11	0.0004	5	11	0.0004	13	21	0.0005
P54	DIAGONAL 7	50	(1,...,1)	2	8	0.0028	3	7	0.0028	3	7	0.0024	5	11	0.002
P55	DIAGONAL 7	100	(1,...,1)	2	7	0.0011	3	7	0.0013	3	7	0.0021	4	10	0.0024
P56	DIAGONAL 7	300	(1,...,1)	2	8	0.0023	3	7	0.0015	3	7	0.0028	3	10	0.0025
P57	RAYDAN 2	100	(1,...,1)	1	5	0.0013	5	9	0.002	3	6	0.0024	9	10	0.0157
P58	RAYDAN 2	500	(1,...,1)	1	5	0.0012	5	9	0.0016	3	6	0.0021	10	11	0.0608
P59	RAYDAN 2	1000	(1,...,1)	1	5	0.0024	5	9	0.0024	3	6	0.0023	12	13	0.3567
P60	Extended quadratic penalty QP2	30	(2,...,2)	6	32	0.0059	***	***	***	***	***	***	***	***	***
P61	Extended quadratic penalty QP2	50	(2,...,2)	6	31	0.0041	***	***	***	***	***	***	***	***	***
P62	Extended quadratic penalty QP1	500	(0.5,...,0.5)	3	12	0.0034	7	16	0.0049	8	19	0.0033	***	***	***
P63	Extended quadratic penalty QP1	1000	(0.5,...,0.5)	3	12	0.0073	7	16	0.0038	7	17	0.0035	***	***	***
P64	Extended quadratic penalty QP1	2000	(0.5,...,0.5)	3	12	0.0073	7	16	0.0054	7	17	0.0048	***	***	***
P65	Extended quadratic penalty QP1	4000	(0.5,...,0.5)	3	12	0.0057	8	18	0.0103	7	17	0.0073	***	***	***
P66	Extended quadratic penalty QP1	5000	(0.5,...,0.5)	3	12	0.0069	8	18	0.0139	7	17	0.0089	***	***	***
P67	Extended Penalty	500	(0.5,...,0.5)	2	7	0.0009	5	11	0.0004	4	10	0.0006	4	12	0.0007
P68	Extended Penalty	800	(0.5,...,0.5)	2	7	0.0005	5	11	0.0008	3	8	0.0004	5	15	0.0015
P69	Extended Penalty	1000	(0.5,...,0.5)	2	7	0.0005	5	11	0.0004	3	8	0.0004	5	16	0.0006
P70	Extended Penalty	2000	(0.5,...,0.5)	2	7	0.0006	4	9	0.0008	3	8	0.0004	6	18	0.0007
P71	DIAGONAL 8	1000	(1,...,1)	2	7	0.0035	3	7	0.0056	4	9	0.0038	6	18	0.0073
P72	DIAGONAL 8	2000	(1,...,1)	2	7	0.0038	3	7	0.0034	4	9	0.004	***	***	***

Table 3. Performance comparison based on NOI, NOF, and CPU time for problems 73–109.

				mDFP			CG_DESCENT			MDL			NSMA		
S/N	**Test Functions**	**DIM**	**Initial Point**	**NI**	**FE**	**CPU**	**NI**	**FE**	**CPU**	**NI**	**FE**	**CPU**	**NI**	**FE**	**CPU**
P73	DIAGONAL 8	4000	(1,...,1)	2	7	0.0073	3	7	0.0061	4	9	0.0061	***	***	***
P74	Quadratic QF2	50	(1,...,1)	2	6	0.0026	4	9	0.0031	2	6	0.0023	***	***	***
P75	Quadratic QF2	1000	(1,...,1)	1	4	0.001	4	9	0.0021	1	4	0.0017	***	***	***
P76	Quadratic QF2	5000	(1,...,1)	1	4	0.0022	4	9	0.0037	1	4	0.0032	***	***	***
P77	DIAGONAL 4	10000	(0.5,...,0.5)	5	11	0.0062	4	8	0.0068	5	9	0.0058	***	***	***
P78	DIAGONAL 4	20000	(0.5,...,0.5)	5	11	0.0123	4	8	0.0093	5	9	0.0104	***	***	***
P79	DIAGONAL 4	50000	(0.5,...,0.5)	5	11	0.0245	4	8	0.0181	5	9	0.0268	***	***	***
P80	DIAGONAL 4	100000	(0.5,...,0.5)	7	15	0.0849	4	8	0.0347	5	9	0.0461	***	***	***
P81	DIAGONAL 4	200000	(0.5,...,0.5)	7	15	0.1705	4	8	0.0933	5	9	0.1003	***	***	***
P82	DIAGONAL 4	500000	(0.5,...,0.5)	9	19	0.7563	4	8	0.3045	5	9	0.3732	***	***	***
P83	SHALLOW	10000	(0.1,...,0.1)	17	47	0.0008	13	25	0.001	8	18	0.0007	***	***	***
P84	SHALLOW	20000	(0.1,...,0.1)	17	47	0.0014	13	25	0.0012	8	18	0.001	***	***	***
P85	SHALLOW	50000	(0.1,...,0.1)	18	49	0.003	13	25	0.0031	8	18	0.0024	***	***	***
P86	SHALLOW	100000	(0.1,...,0.1)	19	51	0.0042	12	22	0.0051	8	18	0.0049	***	***	***
P87	SHALLOW	200000	(0.1,...,0.1)	19	51	0.0121	14	26	0.0093	8	18	0.0115	***	***	***
P88	SHALLOW	500000	(0.1,...,0.1)	20	53	0.0354	13	24	0.0343	8	18	0.0383	***	***	***
P89	Linear Perturbed	10000	(13,...,13)	2	5	0.0007	3	7	0.0006	2	6	0.0007	2	5	0.0012
P90	Linear Perturbed	20000	(13,...,13)	3	7	0.0014	3	7	0.001	***	***	***	2	5	0.0029
P91	Linear Perturbed	50000	(13,...,13)	3	7	0.0021	3	7	0.0016	***	***	***	3	7	0.004
P92	Linear Perturbed	100000	(13,...,13)	3	7	0.0033	4	9	0.0032	***	***	***	3	7	0.0081
P93	Linear Perturbed	200000	(13,...,13)	4	9	0.0107	4	9	0.0114	***	***	***	3	7	0.0281
P94	Linear Perturbed	500000	(13,...,13)	4	9	0.0241	5	11	0.0308	***	***	***	3	7	0.0541
P95	DENSCHNF	10000	(11,...,11)	11	53	0.0017	11	44	0.002	***	***	***	***	***	***
P96	DENSCHNF	20000	(11,...,11)	11	53	0.0028	10	42	0.0023	***	***	***	***	***	***
P97	DENSCHNF	50000	(11,...,11)	11	53	0.0051	10	42	0.0051	***	***	***	***	***	***
P98	DENSCHNF	100000	(11,...,11)	11	53	0.0093	10	42	0.0092	***	***	***	***	***	***
P99	DENSCHNF	200000	(11,...,11)	11	53	0.018	10	42	0.0209	***	***	***	***	***	***
P100	DENSCHNF	500000	(11,...,11)	11	53	0.0846	10	42	0.1074	***	***	***	***	***	***
P101	Extended DENSCHNB	10000	(2,...,2)	1	3	0.0032	1	3	0.001	1	3	0.0018	1	3	0.0044
P102	Extended DENSCHNB	20000	(2,...,2)	1	3	0.0011	1	3	0.0013	1	3	0.0014	1	3	0.0027
P103	Extended DENSCHNB	50000	(2,...,2)	1	3	0.0026	1	3	0.0026	1	3	0.0027	1	3	0.0055
P104	Extended DENSCHNB	100000	(2,...,2)	1	3	0.0044	1	3	0.0056	1	3	0.0052	1	3	0.0099
P105	Extended DENSCHNB	200000	(2,...,2)	1	3	0.0096	1	3	0.0141	1	3	0.0105	1	3	0.0191
P106	Extended DENSCHNB	500000	(2,...,2)	1	3	0.0374	1	3	0.0472	1	3	0.0381	1	3	0.0626
P107	DIAGONAL 6	10000	(0.5,...,0.5)	2	7	0.0066	4	8	0.0122	4	6	0.0065	4	18	0.0483
P108	DIAGONAL 6	20000	(0.5,...,0.5)	2	7	0.0141	4	8	0.0174	4	6	0.0097	4	19	0.035
P109	DIAGONAL 6	50000	(0.5,...,0.5)	2	7	0.0257	4	8	0.0348	4	6	0.0294	***	***	***

Table 4. Performance comparison based on NOI, NOF, and CPU time for problems 110–142.

				mDFP			CG_DESCENT			MDL			NSMA		
S/N	**Test Functions**	**DIM**	**Initial Point**	**NI**	**FE**	**CPU**	**NI**	**FE**	**CPU**	**NI**	**FE**	**CPU**	**NI**	**FE**	**CPU**
P110	DIAGONAL 6	100000	(0.5,...,0.5)	2	7	0.0469	4	8	0.084	4	6	0.0526	4	22	0.2
P111	DIAGONAL 6	200000	(0.5,...,0.5)	2	7	0.1419	4	8	0.2156	4	6	0.1367	***	***	***
P112	DIAGONAL 6	500000	(0.5,...,0.5)	2	7	0.367	***	***	***	4	6	0.3517	5	36	2.0131
P113	FLETCHCR	10000	(13,...,13)	3	20	0.0009	4	26	0.0021	3	45	0.0012	7	41	0.0014
P114	FLETCHCR	20000	(13,...,13)	3	20	0.0015	6	30	0.0023	3	45	0.0019	8	56	0.0024
P115	FLETCHCR	50000	(13,...,13)	3	20	0.0036	5	29	0.004	3	45	0.003	11	78	0.0072
P116	FLETCHCR	100000	(13,...,13)	3	20	0.0073	5	29	0.0058	3	45	0.0075	8	61	0.0118
P117	FLETCHCR	200000	(13,...,13)	3	20	0.0175	5	30	0.0165	3	45	0.0562	12	107	0.0286
P118	FLETCHCR	500000	(13,...,13)	3	20	0.0514	5	30	0.0534	3	45	0.0568	8	79	0.0712
P119	QUARTC	100	(11,...,11)	3	33	0.0041	76	402	0.0489	3	25	0.006	9	49	0.0225
P120	QUARTC	10000	(11,...,11)	3	33	0.0717	92	579	1.2255	3	25	0.0536	7	54	0.116
P121	QUARTC	50000	(11,...,11)	3	33	0.3016	98	657	8.7811	3	25	0.2249	5	43	0.4085
P122	QUARTC	100000	(11,...,11)	3	33	0.6088	100	685	22.948	3	25	0.4589	9	87	1.6285
P123	QUARTC	200000	(11,...,11)	3	33	1.307	102	713	71.659	3	25	1.0076	13	142	5.7455
P124	QUARTC	500000	(11,...,11)	3	33	3.2657	106	770	197.14	3	25	2.5036	14	162	15.852
P125	QUARTICM	1000	(11,...,11)	3	33	0.0006	79	486	0.0024	3	25	0.0007	7	41	0.0009
P126	QUARTICM	10000	(11,...,11)	3	33	0.0017	92	579	0.002	3	25	0.0015	7	54	0.0023
P127	QUARTICM	50000	(11,...,11)	3	33	0.0054	98	657	0.0156	3	25	0.0056	5	43	0.008
P128	QUARTICM	100000	(11,...,11)	3	33	0.0109	100	685	0.036	3	25	0.0124	9	87	0.0157
P129	QUARTICM	200000	(11,...,11)	3	33	0.0227	102	713	0.0245	3	25	0.0247	13	142	0.0355
P130	QUARTICM	500000	(11,...,11)	3	33	0.0583	106	770	0.1797	3	25	0.0626	14	162	0.0872
P131	DENSCHNA	1000	(0.3,...,0.3)	7	21	0.0006	9	17	0.002	6	14	0.0013	***	***	***
P132	DENSCHNA	10000	(0.3,...,0.3)	7	21	0.0013	9	17	0.0014	6	14	0.0016	***	***	***
P133	DENSCHNA	50000	(0.3,...,0.3)	11	42	0.0047	16	41	0.005	***	***	***	***	***	***
P134	DENSCHNA	100000	(0.3,...,0.3)	7	21	0.009	9	17	0.0095	6	14	0.01	***	***	***
P135	DENSCHNA	200000	(0.3,...,0.3)	7	21	0.0209	7	14	0.0215	6	14	0.0214	***	***	***
P136	DENSCHNA	500000	(0.3,...,0.3)	5	14	0.0611	6	12	0.0598	6	15	0.0673	***	***	***
P137	ARWHEAD	10000	(0,...,0)	2	8	0.0009	4	10	0.0009	3	9	0.0007	***	***	***
P138	ARWHEAD	20000	(0,...,0)	2	8	0.0016	4	10	0.0014	3	9	0.0011	***	***	***
P139	ARWHEAD	50000	(0,...,0)	2	8	0.0023	4	10	0.0024	3	9	0.0021	***	***	***
P140	ARWHEAD	100000	(0,...,0)	2	8	0.0038	4	10	0.0042	3	9	0.0058	***	***	***
P141	ARWHEAD	200000	(0,...,0)	2	8	0.0114	4	10	0.0129	3	9	0.0189	***	***	***
P142	ARWHEAD	500000	(0,...,0)	2	8	0.0311	4	10	0.0294	3	9	0.0365	***	***	***

From the above tables, it can be seen that the proposed mDFP algorithm demonstrated robust performance across most of the test problems, showing significant improvements in the NOI, NOF, and computational time compared to the classical CG-DESCENT, MDL, and NSMA algorithms. For example, in the FLETCHCR problem with DIM = 50,000, the proposed mDFP method required 40% fewer iterations, 33% fewer function evaluations, and 3.7% less CPU time. However, for DENSCHNF problems with DIM = $10,000$, both mDFP and CG-DESCENT algorithms required the same NOI to attain the optimal value. While CG-DESCENT required fewer NOF, the proposed mDFP method had less CPU time to reach the solution. On the other hand, both MDL and NSMA algorithms are unsuccessful in this problem. Even though the CG-DESCENT and MDL methods also performed well in some of the problems, the mDFP method showed relatively better efficiency in most cases.

3.4. Accuracy

To demonstrate the accuracy of the algorithms on the considered benchmark functions, this subsection evaluates the number of problems each algorithm can solve and compares the computed solutions against other obtained reference solutions. A summary of this result is presented in Table 5, which illustrates the percentage of failure and success recorded by all algorithms in the computed solutions.

Table 5. Percentage of success and failure recorded by the proposed algorithms compared to other algorithms.

	mDFP	CG-DESCENT	MDL	NSMA
Success	100 %	95.8%	84.5%	53.5%
Failure	0%	4.2%	15.5 %	46.5%

According to Table 5, the NSMA method, a quasi-Newton type method, recorded the lowest percentage of successfully solved problems (53.5%), while the proposed mDFP formula presented the lowest percentage of failure (0%). Though the classical CG-DESCENT and modified MDL algorithms recorded 95.8% and 84.5% success rates, respectively, with 100% success rate, the proposed mDFP algorithm can be considered as more efficient and robust.

3.5. Convergence behavior

The convergence behavior of all the algorithms was evaluated using the performance profile tool introduced by Dolan and Moré [19]. This tool plots the probability that a solver will be within the bounds of a certain factor of the best performance on every function. The x-axis on the profile curve defines the performance ratio, while the y-axis represents the proportion of problems for which the given method is within that performance ratio. A solver whose curve attains higher values faster than others is more robust, meaning it consistently outperforms the other algorithms across various problems.

To generate the profile curve, for each problem p and solver s, we compute the performance ratio:

$$r_{p,s} = \frac{e_{p,s}}{\min_{s\in S}\{e_{p,s}\}},$$

where $e_{p,s}$ denotes the performance metrics including: NOI, NOF, and CPU time of solver s on problem p, while $\min_{s \in S}\{e_{p,s}\}$ represents the best performance attained by any solver s on p problem. In the case of infeasibility, that is, when a solver fails to solve a problem, a large value to $r_{p,s}$, such as a predefined penalty factor τ_{max}, will be assigned.

For each performance ratio τ, the fraction of problems P that solver s solves within a factor τ of the best time is calculated as follows:

$$\rho_s(\tau) = \frac{1}{n_p}|\{p \in P : r_{p,s} \leq \tau\}|,$$

where n_p denotes the total number of problems.

By graphing τ on the x-axis and $\rho_s(\tau)$ on the y-axis for every solver s, we generated the following curves: Figure 1 for NOI, Figure 2 for NOF, and Figure 3 for CPU time. Each curve starts at $(1, \rho_S(1))$ and illustrates the cumulative distribution of performance ratios.

Based on the description of the performance profile, the algorithm whose curve rises fastest is regarded as the best performer on the majority of problems. On the other hand, the height at $\tau = \tau_{max}$ illustrates the fraction of functions for which each algorithm was able to provide a solution, even if not the best. This implies that solvers with higher curves are regarded as more efficient as they solve a higher number of problems within a smaller factor of the best time. From the performance profile curve presented in Figures 7–9, it can be seen that the curve of mDFP algorithm rises fastest on all the metrics. This means that the mDFP algorithm is more efficient as it was able to solve a higher number of problems within a smaller factor of the best time when compared to the CG-DESCENT, MDL, and NSMA methods.

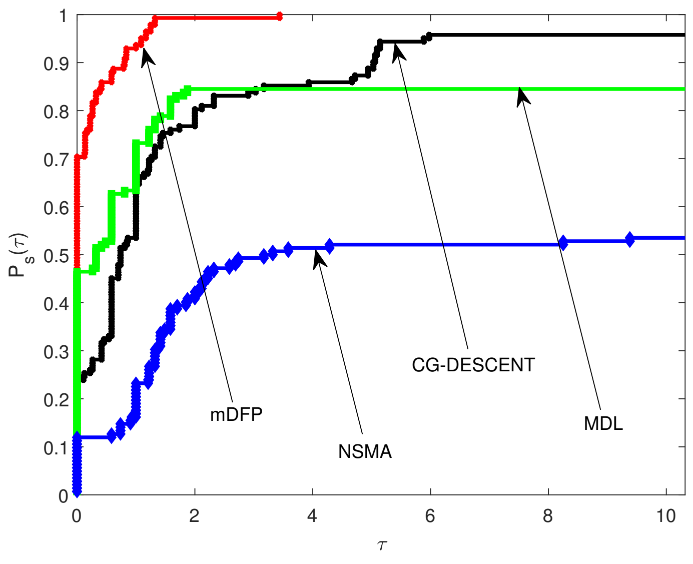

Figure 7. Performance profile comparison with existing methods based on NOI.

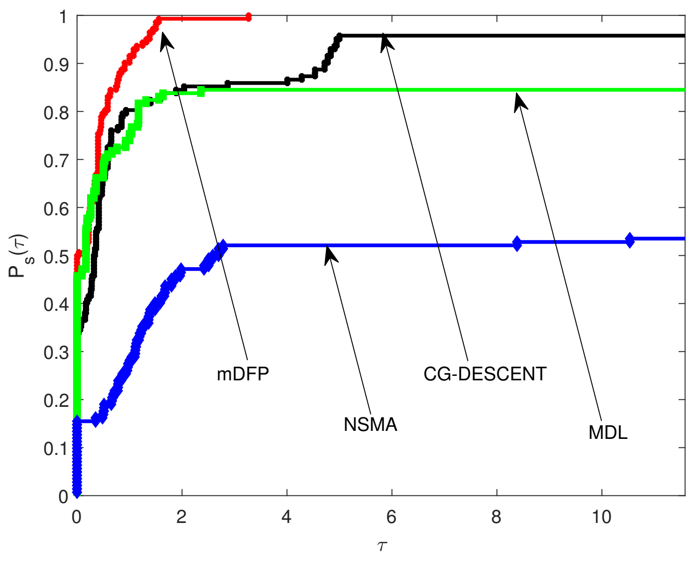

Figure 8. Performance profile comparison with existing methods based on NOF.

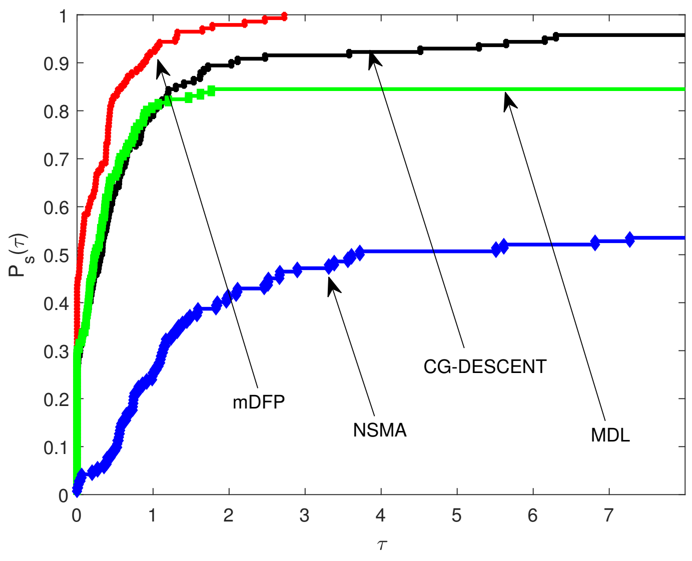

Figure 9. Performance profile comparison with existing methods based on CPU time.

4. Signal recovery

In this subsection, we aim to substantiate the performance of the proposed mDFP in solving a sparse signal problem, which is essentially a task for finding the sparse solution of an underdetermined system of equations. The problem is popularly expressed as a sum of two component functions:

$$\min_{x\in\mathbb{R}^n} \|Ax - y\|_2^2 + \psi R(x), \tag{4.1}$$

where $A \in \mathbb{R}^{m\times n}$, $m \ll n$, and $y \in \mathbb{R}^m$ are referred to as a sensing matrix and measurement vector, respectively. The symbol $\|\cdot\|$ represents the norm ℓ_2, while $\psi > 0$ is the regularization constant or parameter that aids in controlling the trade-off between sparsity and the quality of the recovered sparse signal. Additionally, the term $R(x)$ represents a regularization function that is often adopted as the ℓ_1-norm. The sparse signal problem has applications in many areas of science and engineering; these areas include compressing radar and imaging, machine learning/deep learning [20], wireless networks [21], etc.

Considering that the regularization function, $R(x)$, is non-smooth, smooth-based descent methods may not be applicable to solve (4.1). To overcome this, considerable effort has recently been put into formulating a smooth approximation of $R(x)$. One of such approximations considered in this work was introduced in [22] and used in [23, 24], where the regularizer, $R(x)$ is defined as $R(x) := x \tanh(x/\alpha)$ and α is a smoothing parameter.

Now, by incorporating the smooth formulation of $R(x)$ mentioned above into (4.1), we can approach solving the resultant smooth problem using mDFP and the other two second-best performing methods obtained from the previous numerical section, namely CG-DESCENT and MDL.

Initializations and data generation: The implementation of mDFP in comparison to CG-DESCENT and MDL for solving (4.1) starts with some initializations/definition of signal matrix operator, $A \in \mathbb{R}^{m\times n}$ where $n := 2^{12}$ represent the dimension of the signal, $m = 0.2n$ denotes the number of observation and all algorithms are initialized with $\bar{x}^0 = \|A^T y\|_\infty * ones(n, 1)$.

In addition, the model (4.1) parameter constants include the regularization parameter ψ, and the smoothing parameter α, are $\psi := \max\{2^{-10}, \ 10^{-2}\|A^T y\|_\infty\}$, and 0.1, respectively. However, the algorithms' specific parameter set up, such as the line-search constants and other algorithmic-related terms required to be initialized, remain the same as reported in the respective papers of the compared methods, while those corresponding to mDFP are set as $e = 10^{-1}$, $\vartheta = 10^{-20}$, and Wolfe parameters $\sigma = 0.5$ and $\delta = 10^{-4}$. We compare the performances of mDFP, CG-DESCENT, and MDL methods using the relative error metric [25], which can be defined as

$$\frac{\|\bar{x} - x_{sol}\|}{\|x_{sol}\|} \times 100,$$

in which x_{sol} is an approximate solution of the smooth approximation of model (4.1) or simply the restored signal, while $\bar{x}$ is the original signal.

The outputs of the experiment are described in Figures 10–14, and the original signal and damaged signal are portrayed in Figures 10 and 11, respectively. Moreover, the recovered signals achieved using mDFP, CG-DESCENT, and MDL are depicted in Figures 12–14; these signals can be seen as marked red circles. From these results, we can easily observe that the mDFP algorithm performs considerably better than CG-DESCENT while being close to MDL with respect to the *relative error* metric used. Therefore, we conclusively say that the proposed mDFP has the potential to perform well on signal processing problems.

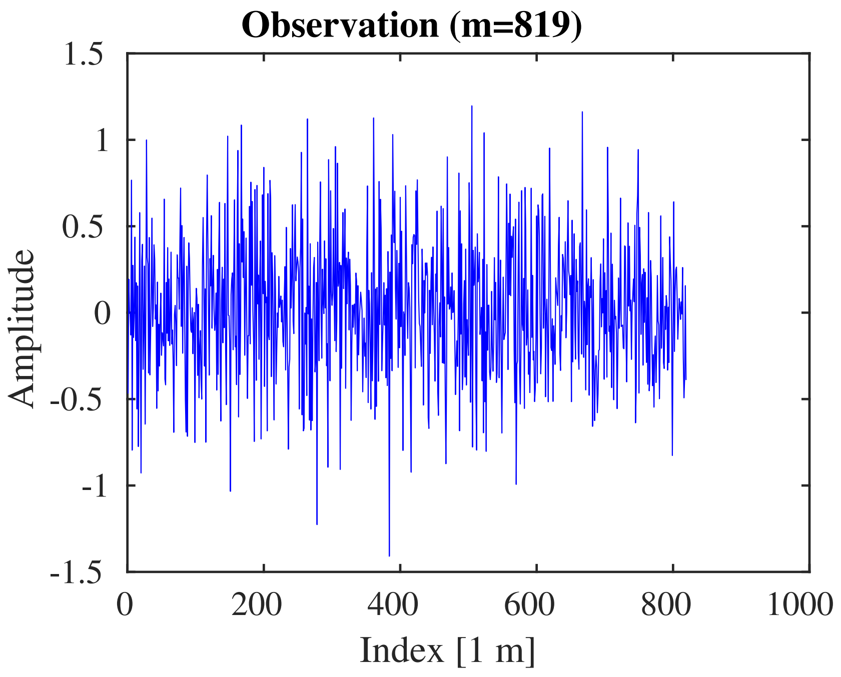

Figure 10. Random observation measurements.

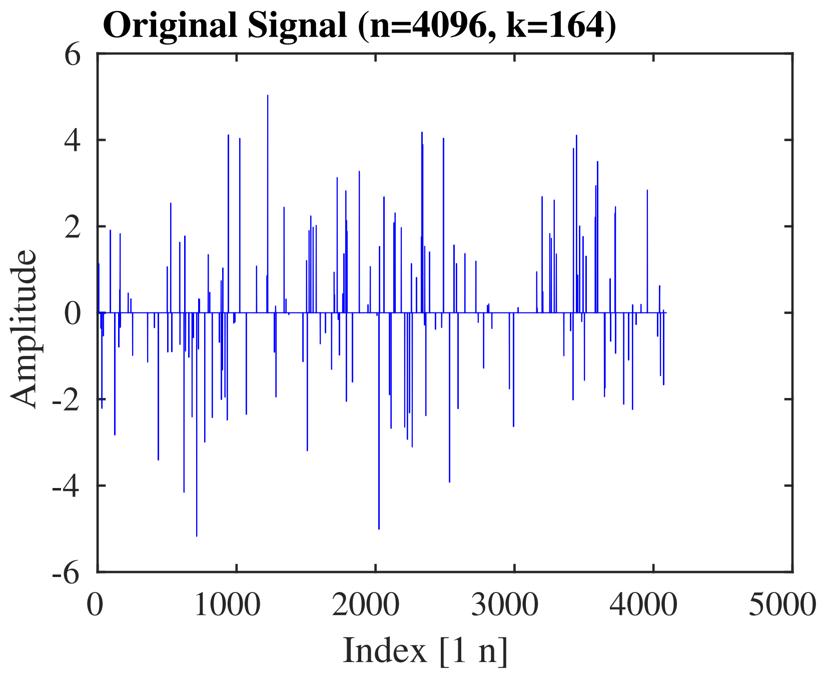

Figure 11. The original signal indicated by blue peaks.

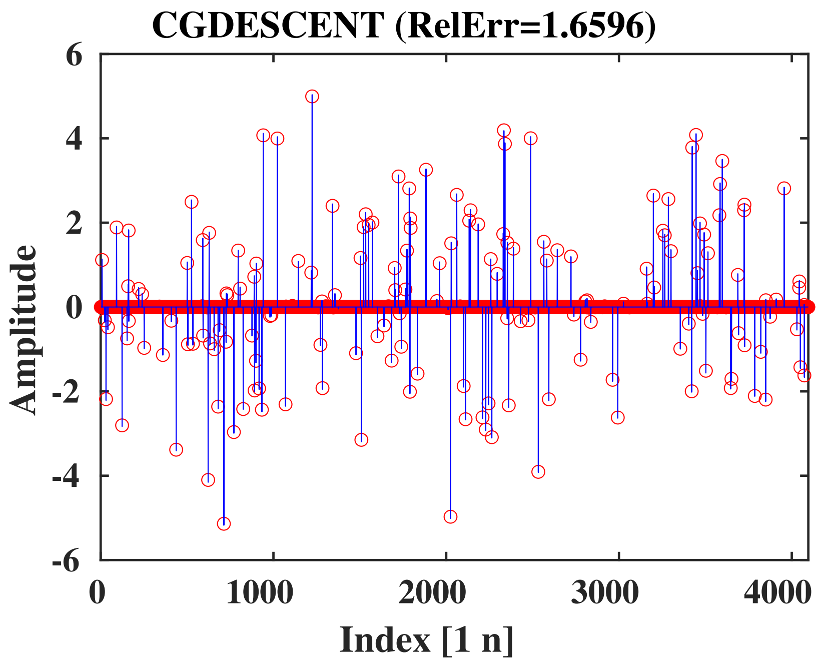

Figure 12. Restored signal (in red circles) using CG-DESCENT with relative error, *RelErr* = 1.6596.

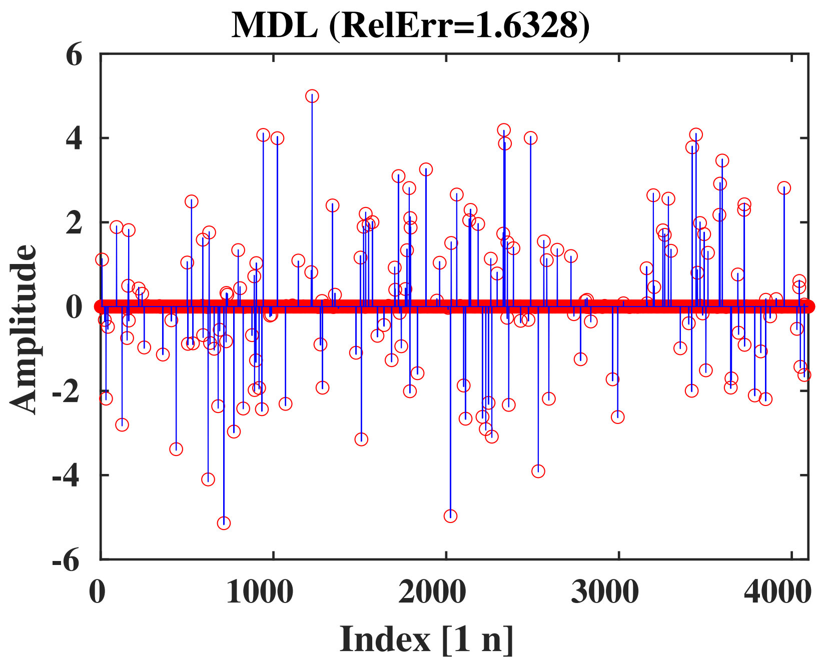

Figure 13. Restored signal (in red circles) using MDL with relative error, *RelErr* = 1.6328.

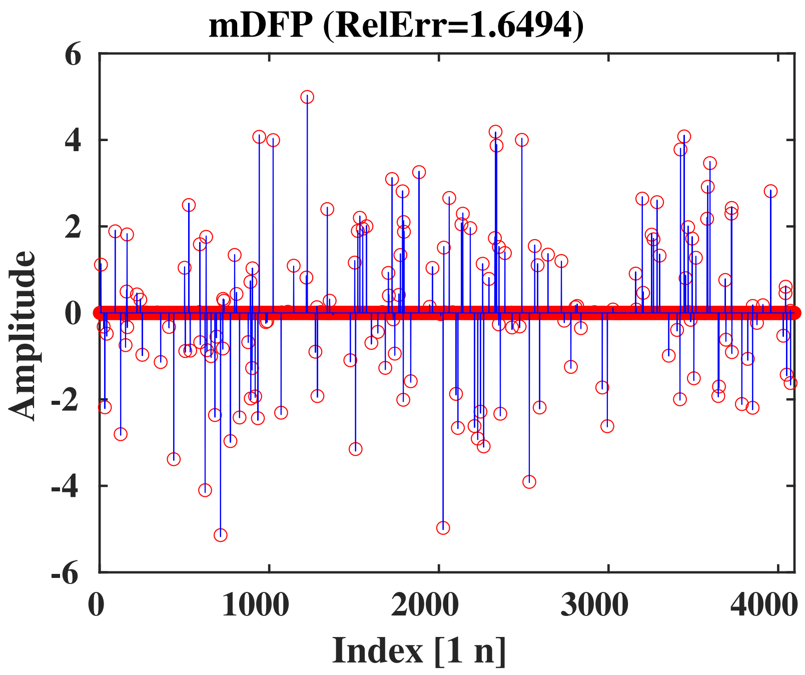

Figure 14. Figure 14, together with Figures 12 and 13, illustrates the performance of the mDFP, CG-DESCENT, and MDL methods, respectively, using the relative error metric for the signal recovery problem in (4.1). The original signal is shown as blue peaks in Figure 10. The noisy observation measurement is represented in Figure 11. The signal recovered by the proposed mDFP, CG-DESCENT, and MDL methods is described in Figures 12–14, respectively; these are plotted using red peaks ending with circular markers superimposed on the original blue peaks signals.

5. Application of mDFP for image recovery

The conjugate gradient (CG) method is renowned for its effectiveness in addressing large-scale convex optimization problems, both smooth and non-smooth, owing to its computational efficiency and minimal memory footprint. A prominent application lies in image restoration, which holds significant relevance in domains such as medical imaging, computational biology, and related fields. This section aims to evaluate the efficacy of a newly developed conjugate gradient algorithm in reconstructing images degraded by noise during data acquisition. Among various noise models, impulse noise remains one of the most prevalent challenges in restoration tasks, as evidenced by recent studies (see: [26–28]). The investigation focuses on leveraging the proposed algorithm to mitigate such noise artifacts while preserving critical image features.

This study focuses on reconstructing two 512 $\times$ 512 grayscale images, *Canal* and *Building*, degraded by salt-and-pepper impulse noise. Let x denote the original image with $M \times N$ pixels and index set $W = 1, 2, ..., M \times 1, 2, ..., N$. The noise-corrupted pixels are identified using an adaptive median filter $\bar{\xi}$ applied to the observed noisy image ξ, with the corrupted pixel set defined as: $K = \{(i, j) \in W |\ \bar{\xi}_{ij} \neq \xi_{ij},\ \xi_{ij} = s_{\min}$ or $s_{\max}\}$.

5.1. *Optimization model*

The restoration process minimizes the energy functional:

$$\min \mathcal{G}(u),$$

where,

$$\mathcal{G}(u) = \sum_{(i,j)\in K} \Big\{ \sum_{(m,n)\in V_{i,j}/K} \phi_\alpha(u_{i,j} - \xi_{m,n}) + \frac{1}{2} \sum_{(m,n)\in V_{i,j}\cap K} \phi_\alpha(u_{i,j} - u_{m,n}) \Big\}.$$

Here, $V_{i\ j} = \{(i,\ j-1),(i,j+1),(i-1,j),(i+1,j)\}$ denotes the 4-neighborhood of pixel (i,j) and $\phi_\alpha(t) = \sqrt{t^2+\alpha}$ is an edge-preserving regularizer with parameter $\alpha > 0$.

5.2. *Performance evaluation*

The proposed method is evaluated using three metrics:

- Peak signal-to-noise ratio (PSNR),
- Relative error (RelErr),
- CPU time (CPUT).

Experiments were conducted for noise intensities of 20%, and 80% using MATLAB 2023a on an Infinix InBook X1 Pro with a 14.00-inch display (1920 × 1080 resolution), Intel Core i7 processor, 16 GB RAM, 512 GB SSD, and Windows 11 operating system. The upgraded hardware significantly enhanced computational efficiency, enabling faster execution and improved handling of large datasets. Table 6 below present the image restoration outputs for three methods: mDFP, CG_DESCENT, and MDL, evaluated based on Peak Signal-to-Noise Ratio (PSNR), Relative Error (RelErr), and Computation Time (CPUT). The experiments were conducted on two images, PAPAYA and GOLDHILL, under two different noise levels (20% and 80%). These results provide insights into the effectiveness, accuracy, and efficiency of each algorithm in handling image restoration problems.

Table 6. Image restoration outputs for mDFP, CG_DESCENT, and MDL based on CPUT, RelErr, and PSNR metrics.

METHOD		**mDFP**			**CG_DESCENT**			**MDL**		
IMAGE	**NOISE**	**CPUT**	**RelErr**	**PSNR**	**CPUT**	**RelErr**	**PSNR**	**CPUT**	**RelErr**	**PSNR**
PAPAYA	20%	**16.2165**	**0.2727**	**30.2337**	22.0670	0.2957	29.7933	21.4961	0.3321	29.4387
	80%	**64.5934**	**0.6642**	**23.2748**	95.4190	0.8788	20.4555	131.5840	0.9649	20.2834
GOLDHILL	20%	8.8416	0.6845	34.2941	**5.5863**	0.7037	34.1898	9.7913	**0.6351**	**34.5224**
	80%	**19.5999**	**2.3251**	**27.1238**	39.6329	2.5936	26.3287	178.263	2.765	25.8217

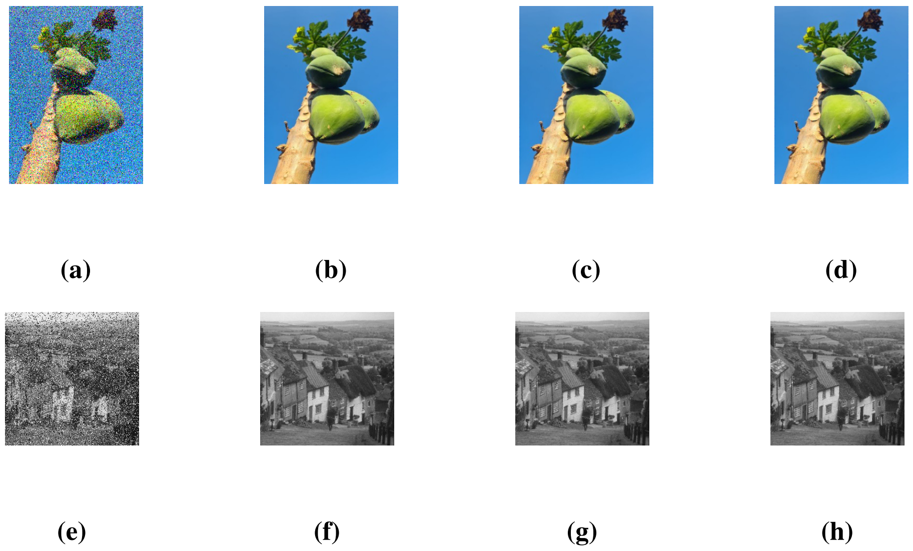

Figure 15. PAPAYA & GOLDHILL images corrupted by 20% salt-and-pepper noise (a) and (e); restored images using mDFP (b) and (f); CG_DESCENT (c) and (g); and MDL (d) and (h).

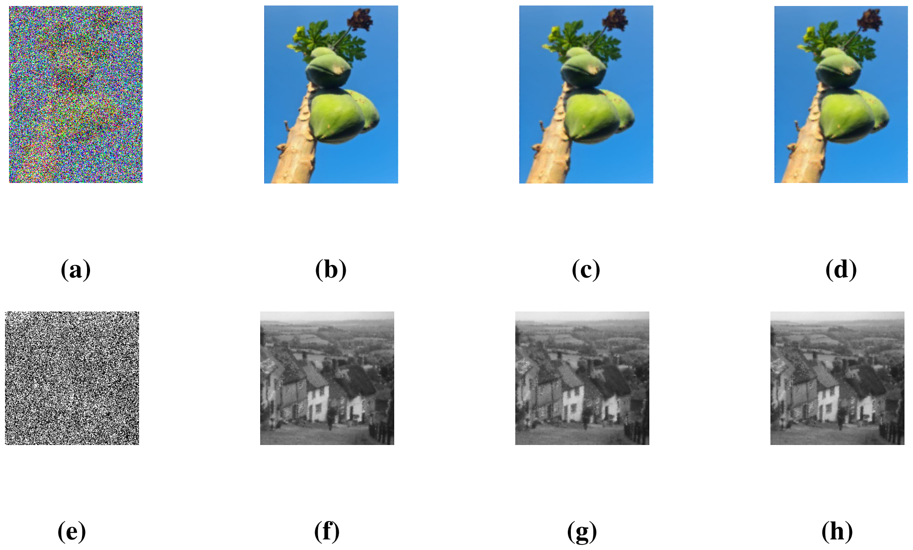

Figure 16. PAPAYA & GOLDHILL images corrupted by 80% salt-and-pepper noise (a) and (e); restored images using mDFP (b) and (f); CG_DESCENT (c) and (g); and MDL (d) and (h).

By considering the performance based on computational efficiency, it can be noted that the proposed mDFP algorithm demonstrates the lowest computation time for low noise levels (20%), particularly for PAPAYA (16.2165 s) and GOLDHILL (8.8416 s). However, as the noise level increases to 80%, the computation time for mDFP rises significantly (64.5934 s for PAPAYA and 19.5999 s for GOLDHILL). CG_DESCENT exhibits significantly higher computational times, particularly at 80% noise for PAPAYA (131.5840 s), making it the slowest algorithm in this case. Meanwhile, the method also has high computational times, particularly for GOLDHILL at 80% noise (128.345 s). These results indicate that while mDFP is efficient at low noise levels, its computational time increases with higher noise, though it still outperforms CG_DESCENT and MDL in terms of efficiency.

Regarding RelErr, which measures the accuracy of the restoration process, mDFP and MDL outperform CG_DESCENT. The lowest RelErr is recorded for GOLDHILL at 80% noise using mDFP (2.3251), demonstrating its robustness in high-noise conditions. In contrast, CG_DESCENT generally has the highest RelErr, indicating weaker performance in restoring images accurately. Interestingly, MDL performs better in some cases, such as PAPAYA at 20% noise, where it achieves the lowest error (0.6351). This suggests that MDL is also competitive when accuracy is the primary concern, but it comes at the cost of higher computation time.

Lastly, in terms of image quality after restoration, assessed using PSNR, mDFP consistently achieves the highest values, particularly for GOLDHILL at 80% noise (27.1238). Higher PSNR values indicate better restoration quality, meaning mDFP produces clearer, less noisy images compared to CG_DESCENT and MDL as illustrated in Figures 15 and 16. CG_DESCENT performs the worst in terms of PSNR, while MDL shows moderate performance, particularly excelling at low noise levels (e.g., 29.4387 for PAPAYA at 20% noise). This suggests that mDFP is the best method for achieving high-quality image restoration across different noise levels.

Overall, the analysis indicates that mDFP is the most well-balanced algorithm for image restoration, offering a combination of high PSNR, low relative error, and fast computation time, making it the best choice for most image restoration tasks. However, MDL algorithm is a strong alternative for cases where minimizing relative error is the priority, but it requires more computational time. On the other hand, CG_DESCENT is the least favorable method, as it is both computationally expensive and yields lower image restoration quality. Thus, for practical applications where speed and quality are essential, mDFP stands out as the most effective method.

6. Conclusions

In this paper, we have proposed a matrix-free optimization algorithm inspired by the famous DFP quasi-Newton updating matrix and developed a new search direction that ensures sufficient descent for all iterations. The new algorithm, mDFP, incorporated a reformulated search direction and a carefully designed scaling parameter to address degeneracy problems and ensure well-defined behavior. Theoretical analysis guaranteed the convergence of the method under standard assumptions, and numerical experiments demonstrated its robustness and efficiency. The sensitivity analysis underscores the importance of the scaling parameter in optimizing performance, while comparative evaluations demonstrated its competitiveness against other classical conjugate gradient-like formulas. Furthermore, the successful implementation of mDFP in compressive sensing and image restoration problems highlighted its practical relevance in real-world situations. These findings established mDFP as an efficient and promising scheme for large-scale minimization problems, particularly in applications like compressive sensing and image restoration. However, one of the issues that needs to be addressed in future research is how best to choose the values for the parameter r, which is suitable for general test problems.

Use of AI tools declaration

The authors declare they have not used Artificial Intelligence (AI) tools in the creation of this article.

Acknowledgments

The authors are grateful to Kuwait Technical College, Kuwait, for supporting this study.

Conflict of interest

The authors declare there are no conflicts of interest.

References

1. P. Singh, S. S. Bose, A quantum-clustering optimization method for COVID-19 CT scan image segmentation, *Expert Syst. Appl.*, **185** (2021), 115637. https://doi.org/10.1016/j.eswa.2021.115637
2. P. Singh, M. K. Muchahari, Solving multi-objective optimization problem of convolutional neural network using fast forward quantum optimization algorithm: Application in digital image classification, *Adv. Eng. Software*, **176** (2023), 103370. https://doi.org/10.1016/j.advengsoft.2022.103370
3. N. Salihu, P. Kumam, I. M. Sulaiman, I. Arzuka, W. Kumam, An efficient newton-like conjugate gradient method with restart strategy and its application, *Math. Comput. Simul.*, **226** (2024), 354–372. https://doi.org/10.1016/j.matcom.2024.07.008
4. M. Malik, S. S. Abas, M. Mamat, I. S. Mohammed, A new hybrid conjugate gradient method with global convergence properties, *Int. J. Adv. Sci. Technol.*, **29** (2020), 199–210.
5. W. Sun, Y. Yuan, *Optimization Theory and Methods: Nonlinear Programming*, Springer, 2006.
6. A. M. Awwal, P. Kumam, K. Sitthithakerngkiet, A. M. Bakoji, A. S. Halilu, I. M. Sulaiman, Derivative-free method based on dfp updating formula for solving convex constrained nonlinear monotone equations and application, *AIMS Math.*, **6** (2021), 8792–8814. https://doi.org/10.3934/math.2021510
7. L. Zhang, W. Zhou, D. Li, A descent modified polak–ribière–polyak conjugate gradient method and its global convergence, *IMA J. Numer. Anal.*, **26** (2006), 629–640. https://doi.org/10.1093/imanum/drl016
8. A. M. Awwal, L. Wang, P. Kumam, M. I. Sulaiman, S. Salisu, N. Salihu, et al., Generalized RMIL conjugate gradient method under the strong wolfe line search with application in image processing, *Math. Methods Appl. Sci.*, **46** (2023), 17544–17556. https://doi.org/10.1002/mma.9515
9. S. M. Ibrahim, N. Salihu, Two sufficient descent spectral conjugate gradient algorithms for unconstrained optimization with application, *Optim. Eng.*, **26** (2025), 655–679. https://doi.org/10.1007/s11081-024-09899-z

10. A. U. Omesa, S. M. Ibrahim, R. B. Yunus, I. A. Moghrab, M. Y. Waziri, A. Sambas, A brief survey of line search methods for optimization problems, *Results Control Optim.*, **19** (2025), 100550. https://doi.org/10.1016/j.rico.2025.100550

11. N. Andrei, Scaled conjugate gradient algorithms for unconstrained optimization, *Comput. Optim. Appl.*, **38** (2007), 401–416. https://doi.org/10.1007/s10589-007-9055-7

12. P. Wolfe, Convergence conditions for ascent methods, *SIAM Rev.*, **11** (1969), 226–235. https://doi.org/10.1137/1011036

13. P. Wolfe, Convergence conditions for ascent methods. ii: Some corrections, *SIAM Rev.*, **13** (1971), 185–188. https://doi.org/10.1137/1013035

14. G. Zoutendijk, Nonlinear programming, computational methods, *Integer Nonlinear Program.*, **1970** (1970), 37–86.

15. W. W. Hager, H. Zhang, Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent, *ACM Trans. Math. Software*, **32** (2006), 113–137. https://doi.org/10.1145/1132973.1132979

16. J. Zhang, Y. Xiao, Z. Wei, Nonlinear conjugate gradient methods with sufficient descent condition for large-scale unconstrained optimization, *Math. Probl. Eng.*, **2009** (2009), 243290. https://doi.org/10.1155/2009/243290

17. M. Jourak, S. Nezhadhosein, F. Rahpeymaii, A new self-scaling memoryless quasi-newton update for unconstrained optimization, *4OR*, **22** (2024), 235–252. https://doi.org/10.1007/s10288-023-00544-6

18. N. Andrei, An unconstrained optimization test functions collection, *Adv. Model. Optim.*, **10** (2008), 147–161.

19. E. D. Dolan, J. J. Moré, Benchmarking optimization software with performance profiles, *Math. Program.*, **91** (2002), 201–213. https://doi.org/10.1007/s101070100263

20. B. Sun, H. Feng, K. Chen, X. Zhu, A deep learning framework of quantized compressed sensing for wireless neural recording, *IEEE Access*, **4** (2016), 5169–5178. https://doi.org/10.1109/ACCESS.2016.2604397

21. I. F. Akyildiz, M. C. Vuran, *Wireless Sensor Networks*, John Wiley & Sons, 2010.

22. Y. J. J. Bagul, A smooth transcendental approximation to |x|, *Int. J. Math. Sci. Eng. Appls. (IJMSEA)*, **11** (2017), 213–217.

23. A. M. Awwal, M. M. Yahaya, N. Pakkaranang, N. Pholasa, A new variant of the conjugate descent method for solving unconstrained optimization problems and applications, *Mathematics (2227-7390)*, **12** (2024), 2430. https://doi.org/10.3390/math12152430

24. Z. Aminifard, S. Babaie-Kafaki, A nonmonotone admm-based diagonal quasi-newton update with application to the compressive sensing problem, *Math. Model. Anal.*, **28** (2023), 673–688. https://doi.org/10.3846/mma.2023.16993

25. Z. Aminifard, A. Hosseini, S. Babaie-Kafaki, Modified conjugate gradient method for solving sparse recovery problem with nonconvex penalty, *Signal Process.*, **193** (2022), 108424. https://doi.org/10.1016/j.sigpro.2021.108424

26. N. Salihu, P. Kumam, S. M. Ibrahim, W. Kumam, Some combined techniques of spectral conjugate gradient methods with applications to robotic and image restoration models, *Numer. Algor.*, **2024** (2024), 1–41. https://doi.org/10.1007/s11075-024-01970-1
27. Y. Liu, Z. Zhu, B. Zhang, Two sufficient descent three-term conjugate gradient methods for unconstrained optimization problems with applications in compressive sensing, *J. Appl. Math. Comput.*, **68** (2022), 1787–1816. https://doi.org/10.1007/s12190-021-01589-8
28. X. Jiang, W. Liao, J. Yin, J. Jian, A new family of hybrid three-term conjugate gradient methods with applications in image restoration, *Numer. Algor.*, **91** (2022), 161–191. https://doi.org/10.1007/s11075-022-01258-2