# Solving Unconstrained Minimization Problems Using the Steepest Descent and Conjugate Gradient Algorithms with a New Beta-k Parameter

Huda H. Al-Zobiadi and Adawiya A. Mahmood Al-Nuaimi
*Department of Mathematics, College of Science, University of Diyala, 32001 Baqubah, Iraq*
*scimathms242508@uodiyala.edu.iq, dr.adawiya@uodiyala.edu.iq*

Keywords: Nonlinear, Optimization, Unconstrained Problems, Steepest Descent Algorithm, Conjugate Gradient Algorithm, MATLAB Program.

Abstract: Decision-making of almost any kind can be framed as an optimisation problem, falling into one of two broad families: constrained or unconstrained. Building on this framework, the present paper develops a Steepest Descent (SD) algorithm and a Conjugate Gradient (CG) algorithm, both constructed around a newly proposed parameter, $\beta_k^{`}$ itself. To evaluate their performance, both algorithms are implemented in MATLAB and tested on a set of nonlinear unconstrained test functions that have not previously appeared in the literature for this problem's numerical examples, ensuring an independent assessment of their behavior. The paper's central contribution is $\beta_k^{`}$, proposed specifically to make the CG algorithm more effective at locating the minimum of the objective function. Wherever it appears below, $\beta_k^{`}$ denotes this new parameter, and all results obtained using it are compared directly against those of the classical SD algorithm, allowing for a clear, side-by-side evaluation of efficiency and convergence behavior. Across the MATLAB experiments, performance was tracked along two key dimensions: accuracy of the final solution and running time required to reach it. The results show that the SD algorithm consistently needed more iterations to converge than the CG algorithm equipped with the new $\beta_k^{`}$, which reached the solution more efficiently while maintaining a runtime close to that of SD. This combination of faster convergence and comparable computational cost suggests that the new parameter offers a genuine practical advantage rather than a trade-off. These results point toward the possibility of tackling larger, more complex unconstrained problems involving additional variables, and lay the groundwork for extending the proposed approach to broader classes of optimisation problems in future work.

## 1 INTRODUCTION

Unconstrained optimisation underpins a wide range of fields, from economics, political science and management to manufacturing, biology, physics and engineering [1], and its rigorous mathematical treatment took shape over the course of the twentieth century [2]. Among the available techniques, the CG algorithm stands out for being simple to implement and economical in memory, which makes it particularly well suited to problems with a large number of variables [3]. Other lines of work have pursued different tools for the same goal: a filled-function approach with an adjustable parameter has been used to tackle practical optimisation problems [4], and Zhang refined a nonlinear CG scheme that satisfies the descent condition under a strong Wolfe line search [5]. The SD algorithm, for its part, is one of the oldest and most basic tools for unconstrained optimisation [6]-[8], and it continues to attract research attention [6], [9]-[12]. Fliege and Svaiter proposed a Pareto-descent procedure for multi-objective optimisation [13], while Drummond and Svaiter, in [7], devised a Cauchy-type method for smooth, unconstrained vector optimisation and established global convergence to a weak minimiser. Fliege and Svaiter further extended SD to unconstrained multicriteria optimisation and introduced a feasible-descent-direction variant for the constrained multicriteria case [13]. Interest in gradient-based methods was later renewed by the step-size strategy of Borwein and Barzilai [14], who showed that two step-length choices, close in magnitude to the Cauchy step, could be obtained by approximating the secant equation, giving rise to a

two-point-step-size SD method. Building on this, Fletcher proposed a technique that exploits the availability of "long" saved vectors for objective functions, yielding substantial gains in performance [9]. CG itself sits conceptually between Newton's method and SD [15], modifying the SD search direction by a positive multiple of the previous direction [15]; because it avoids storing any matrices, it scales well to problems with many variables, though its convergence is only linear [16]. Fletcher and Reeves were the first to apply a CG algorithm to the general unconstrained minimisation problem [17], presenting a quadratically convergent method for locating a relative minimiser of a multivariable function; as a machine method, CG outperforms elimination, needing less storage and being easier to implement since it reaches a solution in n steps [18]-[20]. Stanimirović et al. surveyed and compared the leading Quasi-Newton and CG methods to give a global picture of progress in the field [21], while Lasdon et al. extended the Fletcher-Reeves CG method to optimal-control problems, finding it to converge markedly faster than SD in every case tested [22]. More recently, Yunis proposed two new CG methods for nonlinear unconstrained optimisation, one built on a newly discovered search direction and the other a spectral variant [23]; Salih and Faraj compared CG against SD for minimising nonlinear functions [24]; Zaki et al. combined the Dai-Yuan, Hestenes-Stiefel and Hager-Zhang CG formulas into a new hybrid method [25]; and a further parameterised CG approach was proposed for large-scale unconstrained problems [26]. Evada et al. accelerated convergence to a minimum by reducing the number of SD iterations required [27], and nonlinear CG has also been applied successfully to control a three-degree-of-freedom robotic planar motion system [28].

# 2 FORMULATIONS OF UNCONSTRUCTION MINIMIZATION PROBLEM

We consider the unconstrained minimisation problem in the general form [29]

$$\begin{matrix} Minimize f(x) \\ Subject\ to\ x \epsilon \Omega \end{matrix}. \quad (1)$$

Where $f$ denotes a real-valued objective function, while $\Omega \subseteq E^n$ is the feasible set. Problem (1) then raises a natural question: does a solution exist? A classical result guarantees one whenever $f$ is continuous and $\Omega$ is compact — this is the essential fact underlying everything that follows, and it should be borne in mind both when characterising solution points and when designing algorithms to locate them. Two kinds of solution point are relevant to problem (1): the local minimiser and the global minimiser.

**Definition (1):**[29] A point $X^* \epsilon\, \Omega$ is called a relative (local) minimiser of $f\ on\ \Omega$ if there exists some $\gamma > 0$ such that $f(X) \geq f(X^*)\ \forall\, X \epsilon\, \Omega$ within a distance $\gamma$ of X* (i.e $, X \epsilon \Omega\ and |X - X^*| < \gamma). Iif\ f(X) > f(X^*)\ \forall\, X \epsilon\, \Omega, X \neq X*,$ within a displacement $\gamma$ of $X^*$ , then $X^*$ is called a strict local minimiser of $f on\ \Omega$ .

**Definition (2):**[29] a point $X^* \epsilon\, \Omega$ is called a global minimiser of $f\ on\ \Omega$ if$f(X) \geq f(X^*)\ \forall\, X \epsilon\, \Omega$. If $f(X) > f(X^*)\ \forall X \epsilon \Omega, X \neq X^*,$ then $X^*$is called a strict global minimiser of $f\ over\ \Omega$.

In constructing and solving issue (1) we are searching for a relative minimum point.

We introduce two algorithms, the steepest descent (SD)algorithm and conjugate gradient (CG) algorithm to obtain minimum points.

To solve the unconstrained minimization problem, two important concepts should first be introduced: the local minimizer and the global minimizer.

- X: ($x_1$,$x_2$) Vector or point;
- $f(x)$: Nonlinear function;
- m: gradient vector of $f(x) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \end{bmatrix}$.
- H: Hessian matrix $of\ f(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} \end{bmatrix}$.

# 3 SD ALGORITHEM

Suppose $f$ has continuous first-order partial derivatives on $E^n$. Since the gradient vector of $f$ is frequently required and thus, we provide certain simplified notation. By convention the gradient $\nabla f(\mathrm{x})$ is taken to be an n-dimensional row vector, and for convenience its transpose is written as the n-dimensional column vector $m(\mathrm{x}) = \nabla\, f(\mathrm{x})^T$. Where no confusion can arise the argument x is dropped, so that we may write $m_k\, for\, m(\mathrm{x}_k) = \nabla f(\mathrm{x}_k)^T$.

The SD method proceeds via the iterative rule

$$\mathrm{x}_{k+1} = \mathrm{x}_k - \alpha_k m_k.$$

Where $\alpha_k$ is a non-negative parameter that may minimize $f(x_k - \alpha m_k)$. To put it another way, the search is performed along the negative gradient direction $-m_k$ starting from $x_k$ toward a minimizer along this line, is taken as $x_{k+1}$ [29].

The (SD) algorithm can be written as in the following steps: [2], [24], [29]

Step (1): Given $X_0$ (starting value of design variable) Compute $f(X_0)$ and $m_0$=grad $f(X_0)$, and tol (tolerance on gradient value).

Step (2): put $H_k$ is Hessian matrix of $f(x_k)$,

$\alpha_k = \frac{m_k^T m_k}{m_k^T H_k m_k} \quad \forall k \epsilon N$

Step (3): put $x_{k+1} = x_k - \alpha_k m_k$, and $m_{k+1} = grad(f(x_{k+1}))$.

Step (4): if $||m_{k+1}|| < tol$ , then stop, $x_{k+1}$ is minimum point and $f(x_{k+1})$ is the minimum value, else, go to step (2).

Global convergence properties for the SD algorithm is given in [29].

# 4 CG TECHNIQUE

The CG technique builds on a simple underlying procedure, standard CG, originally devised to minimise a strictly convex quadratic function [30], [31]. Applied to the nonlinear unconstrained optimisation problem

$$Minimize\ f\ (x),$$

In which $f: R^n \rightarrow R$ is a function with continuous derivatives, and each non-linear CG algorithm produces the following sequence $\{x_k\}$.

$$x_{k+1} = x_k + \alpha_k d_k.$$

Where $\alpha_k$ is the step size selected by search of line and $d_k$ is the direction of search calculated by:

$$d_{k+1} = -m_{k+1} + \beta_k d_k$$

For $k \geq 0$, where $\beta_k$ denotes CG factor with $m_{k+1} = \nabla(f(x_{k+1}))$. In CG procedures, $d_0 = -m_0$ [29].

## 4.1 CG Algorithm

The presented algorithm can be described as follows [24], [29]:

Step (1): Given $X_0$ (starting value of design variable) Compute $f(X_0)$ and $m_0 = grad\ f(X_0)$, tol (tolerance on gradient value) and $d_0$= -$m_0$

Step (2): put $H_k$ is Hessian matrix of $f(x_k)$, $\alpha_k = -\frac{m_k^T d_k}{d_k^T H_k d_k}$ , ∀kϵ N

Step (3): put $x_{k+1} = x_k + \alpha_k d_k$, $m_{k+1} = grad(f(x_{k+1}))$.

Step (4): if $||m_{k+1}|| < tol$ ,then stop, $x_{k+1}$ the minimum point and $f(x_{k+1})$ is the minimum value.

Step (5): find $\beta_k = \frac{m_{k+1}^T H_k d_k}{d_k^T H_k d_k}$, $d_{k+1} = -m_{k+1} + \beta_k d_k$ and go to step (2).

Different choices of $\beta_k$ give rise to the various CG methods documented in [3]. Building on these, we propose a new parameter, $\beta_k^{`}$, aimed at improving the effectiveness of the CG algorithm:

$$\beta_k^{`} = \frac{m_{k+1}^T\ H_k\ m_{k+1}}{d_k^T\ H_k\ m_k}$$

Where $m_{k+1}^T$ is the transpose of the $grad\ (f(x_{k+1}))$, $d_k^T = -m_k^T$,

$m_{k+1} = grad\ (f(x_{k+1}))$.

$H_k$ = Hessian matrix of $f(x_k)$.

$m_k$= grad of $f(x_k)$.

The well-known cases of $\beta_k$ that define the different Conjugate gradient methods can be taken as [3]:

$\beta_k^{FR} = \frac{\nabla^T f(x_k) \nabla f(x_k)}{\nabla^T f(x_{k-1})\ \nabla f(x_{k-1})}$ (Fletcher-Reeves)

$\beta_k^{PRP} = \frac{\nabla^T f(x_k)\ (\nabla f(x_k) - \nabla f(x_{k-1}))}{\nabla^T f(x_{k-1})\ \nabla f(x_{k-1})}$ (Polak −Ribiere-Polyak)

## 4.2 Global Convergence Analysis for the Conjugate Gradient (CG) Algorithm with New Proposed $\beta_k^{`}$

We begin the global-convergence analysis of the proposed parameter $\beta_k^{`}$, starting from the sufficient-descent condition.

### 4.2.1 Sufficient Descent Condition

To satisfy the condition of sufficient descent:

$$m_k^T d_k \leq -c\|m_k\|^2\ \forall k \geq 0, c > 0. \qquad (2)$$

**Theorem 4.2.1**. Let $\{x_k\}$ and $\{d_k\}$ be a sequence obtained by algorithm (CG) with new propose $\beta_k^{`} = \frac{m_{k+1}^T\ H_k\ m_{k+1}}{d_k^T\ H_k\ m_k}$.

Therefore, (2) satisfied, ∀ $k \geq 0$.

**Proof**. We prove by induction, that if k= 0 then $m_0^T d_0 = -c\ \|m_0\|^2$

Thus, the condition holds; we now prove that:

$$m_k^T d_k \leq -c\ \|m_k\|^2 for\ k \geq 1.$$

From the search direction $d_k$ $defined\ by$:

$$d_k = \begin{cases} -m_k & if\ k = 0 \\ -m_k + \beta_k^{`}\, d_{(k-1)} & if\ k \geq 1 \end{cases}. \quad (3)$$

We have $d_{k+1} = -m_{k+1} + \beta_{k+1}^{`} d_k$, Multiply both sides by $m_{k+1}^T$.

$$m_{k+1}^T d_{k+1} = m_{k+1}^T\left(-m_{k+1} + \beta_{k+1}^{`} d_k\right) = -\left|\left|m_{k+1}\right|\right|^2 + \beta_{k+1}^{`} m_{k+1}^T d_k.$$

With the use of an exact line search, we obtain that $m_{k+1}^T d_k$=0. Thus $m_{k+1}^T d_{(k+1)} = -\|m_{k+1}\|^2$, hence this criterion meets for k+1. Thus (2) is valid.

This confirms that the proposed $\beta_k^{`}$ satisfies condition (2).

### 4.2.2 Global Convergence Behaviors

To show globally converging properties, the following hypotheses is required.

**Hypotheses 4.2.2 [3]**

(i) $\Omega = \{x \in R^n, f(x) \leq f(x_0)\}$ is bounded set, which $x_0$ is the initial point.

(ii) within a neighbor of $\Omega$, the function of objective has continuous derivative, and Lipschitz continuous is its gradient, i.e., there is a fixed value L > 0 s.t:

$$\|m(x) - m(y)\| \leq L\,\|x - y\|\ \forall x, y \in N \quad (5)$$

The next lemma that is well-known for globally converging characteristics in [3].

**Lemma 4.2.2.** Assume hypotheses (4.2.2) satisfies, suppose $x_k$ constructed by CG algorithm with new proposed $\beta_k^{`}$ and $d_k$ meets $d_k = -m_k + \beta_{k-1}^{`} d_{k-1}\ \forall k \geq 1$, and $\alpha_k$ is attained by $\alpha_k = -\frac{m_k^T d_k}{d_k^T H_k d_k}$, therefore we get: $\sum_{k=1}^{\infty} \frac{(m_k^T d_k)^2}{\|d_k\|^2} < \infty$ [3].

**Theorem 4.2.2.** Assume that Hypotheses (4.2.2) is true, the sequence $\{x_k\}$ is obtained by CG algorithm with new proposed $\beta_k^{`}$, if $\|\alpha_k d_k\| \to 0$ while $k \to \infty$ then:

$$\lim_{k\to\infty} inf \|m_k\| = 0. \quad (6)$$

**Proof.** If (6) is not satisfied, then for all k, there is positive constant c s.t.

$$\|m_k\| \geq c. \quad (7)$$

By $\|\alpha_k d_k\| \to 0$ and (5), then $\exists\, M > 0\ for\ all\ k \geq M$, such that:

$$\left\|m_{(k+1)} - m_k\right\| \leq \frac{1}{10}\, c. \quad (8)$$

But by the Lemma (4.2.2), we get:

$$\sum_{k=1}^{\infty} \frac{(m_k^T d_k)^2}{\|d_k\|^2} < \infty$$

It follows that $\lim_{k\to\infty} inf \|m_k\| = 0$, which contradicts (7); this contradiction completes the proof. Thus, the proposed $\beta_k^{`}$ is globally convergent.

**Example 1**: Suppose the nonlinear function (4) $f(x_1,x_2) = 3(x_1-x_2)^2+(2x_2+1)^2$ such that $x_0=(2,0.5)^T$, *tol*=0.01.

**Solution**: The SD algorithm is applied as follows: The gradient of *f* at $x_0$ is:

$$m_0=\text{grad}(f(x_0))=\begin{bmatrix} 6x_1 - 6x_2 \\ -6x_1 + 14x_2 + 4 \end{bmatrix} = \begin{bmatrix} (6*2) - (6*0.5) \\ (-6*2) + (14*0.5) + 4 \end{bmatrix} = \begin{bmatrix} 9 \\ -1 \end{bmatrix}.$$

The Hessian matrix of $f(x_0)$ is $H=\begin{bmatrix} 6 & -6 \\ -6 & 14 \end{bmatrix}$. We get:

$\alpha_0=\frac{m_0^T m_0}{m_0^T H_0 m_0}=\frac{[9\ \ -1]\begin{bmatrix} 9 \\ -1 \end{bmatrix}}{[9\ \ -1]\begin{bmatrix} 6 & -6 \\ -6 & 14 \end{bmatrix}\begin{bmatrix} 9 \\ -1 \end{bmatrix}}=\frac{82}{[60\ \ -68]\begin{bmatrix} 9 \\ -1 \end{bmatrix}}=\frac{82}{608}=\frac{41}{304}$ put $x_1 = x_0 - \alpha_0 m_0 = \begin{bmatrix} 2 \\ 0.5 \end{bmatrix} - \frac{41}{304}\begin{bmatrix} 9 \\ -1 \end{bmatrix} = \begin{bmatrix} 2 \\ 0.5 \end{bmatrix} - \begin{bmatrix} \frac{369}{304} \\ \frac{-41}{304} \end{bmatrix} = \begin{bmatrix} 2 - \frac{369}{304} \\ 0.5 + \frac{41}{304} \end{bmatrix} = \begin{bmatrix} 0.786184 \\ 0.634868 \end{bmatrix}$

Then:

$$m_1 = \nabla f(x_1) = \begin{bmatrix} (6*0.786184) - (6*0.634868) \\ (-6*0.786184) + (14*0.634868) + 4 \end{bmatrix} = \begin{bmatrix} 0.907896 \\ 8.171048 \end{bmatrix}.$$

Thus:

$\|m_1\| = \sqrt{(0.907896)^2 + (8.171048)^2} = 8.221332 > tol$ .

Since the condition $\|m_1\| < tol$ is not met the algorithm is continued and repeat step (2) until we reach to the solution at 21 iterations.

Hence,$x = (-0.498177, -0.499271)^T$ is the minimum point and the minimum value of $f(x) = 0.000006$.

Now, we will apply the CG algorithm with new $\beta_k^{`}$ for example 1 as follows:

$$m_0=\text{grad}(f(x_0))=\begin{bmatrix} 6x_1 - 6x_2 \\ -6x_1 + 14x_2 + 4 \end{bmatrix} = \begin{bmatrix} (6*2) - (6*0.5) \\ (-6*2) + (14*0.5) + 4 \end{bmatrix} = \begin{bmatrix} 9 \\ -1 \end{bmatrix}$$ with $H=\begin{bmatrix} 6 & -6 \\ -6 & 14 \end{bmatrix}$.

Thus $d_0 = -m_0 = \begin{bmatrix}-9\\1\end{bmatrix}, \alpha_0 = -\frac{m_0^T d_0}{d_0^T H_0 d_0} =$

$$-\frac{\begin{bmatrix}9 & -1\end{bmatrix}\begin{bmatrix}-9\\1\end{bmatrix}}{\begin{bmatrix}-9 & 1\end{bmatrix}\begin{bmatrix}6 & -6\\-6 & 14\end{bmatrix}\begin{bmatrix}-9\\1\end{bmatrix}} = -\frac{-82}{608} = \frac{41}{304}$$

and $x_1 = x_0 + \alpha_0 d_0 = \begin{bmatrix}2\\0.5\end{bmatrix} + \frac{41}{304}\begin{bmatrix}-9\\1\end{bmatrix} = \begin{bmatrix}2\\0.5\end{bmatrix} + \begin{bmatrix}\frac{-369}{304}\\\frac{41}{304}\end{bmatrix} = \begin{bmatrix}0.7862\\0.6349\end{bmatrix}$.

$$m_1 = grad(f(x_1)) = \begin{bmatrix}(6 * 0.7862) - (6 * 0.6349)\\(-6 * 0.7862) + (14 * 0.6349) + 4\end{bmatrix}$$

$$m_{1=}\begin{bmatrix}0.9079\\8.1711\end{bmatrix}.$$

Thus $||m_1|| = \sqrt{(0.9079)^2+(8.1711)^2}$ = 8.2213 >tol, Since the condition $||m_1|| < tol$ is not met the algorithm is continued.

$$\beta_0{}` = \frac{m_1^T H_0 m_1}{d_0^T H_0 m_0} = \frac{\begin{bmatrix}0.9079 & 8.1711\end{bmatrix}\begin{bmatrix}6 & -6\\-6 & 14\end{bmatrix}\begin{bmatrix}0.9079\\8.1711\end{bmatrix}}{\begin{bmatrix}-9 & 1\end{bmatrix}\begin{bmatrix}6 & -6\\-6 & 14\end{bmatrix}\begin{bmatrix}9\\-1\end{bmatrix}} = -1.339.$$

The computations for the algorithm are continued until we obtain the solution at 11 iterations.

Hence $x = (-0.50018, -0.50006)^T$ is the minimum point and $f(x) = 5.5242e-08$ is the minimum value.

# 5 MATLAB RUN

The MATLAB programming is run for the minimization of three functions for example1, example 2 and example3. The SD algorithm and the CG algorithm with new $\beta_k`$ are run in MATLAB 2024. The input parameter settings for the two algorithms found in [24].

The results of the SD algorithm for example 1 are presented in Table 1. In this table we have:

k: Iteration.

$x_1, x_2$ = components of a vector x: $\alpha_k = \frac{m_k^T m_k}{m_k^T H_k m_k}$.

$\|m\|$ = norm of $\nabla f$.

$f(x)$ = value of nonlinear function $f$ $at$ x.

$m_1 = \nabla f(x_1)$.

$m_2 = \nabla f(x_2)$.

The minimum value of the function in example 1 is obtained within 21 iterations for the SD algorithm. Thus, $x$ =(-0.498177, -0.499271)$^T$ is the minimum point and $f(x)$=0.000006 represents the minimal value and $m$=(0.006564, -0.000732)$^T$ $||m||$=0.006605 < tol. The execution time for the SD algorithm with plots is 0.711204 seconds. The plots of the function and the decrease of f($x_k$) per iteration for are presented in Figure 1 and 2 respectively for example 1.

The outcomes of the CG algorithm with new $\beta_k`$ for example 1 is presented in Table 2. In this table we have:

k: Iteration.

$x_1, x_2$ = components of a vector x

$\alpha_k = \frac{m_k^T m_k}{m_k^T H_k m_k}$.

$\|m\|$ = norm of $\nabla f$.

$f(x)$ = value of nonlinear function $f$ $at$ $x$.

$m_1 = \nabla f(x_1)$.

$m_2 = \nabla f(x_2)$.

$New\ proposed\ \beta_k` = \frac{m_{k+1}^T H_k m_{k+1}}{d_k^T H_k m_k}$.

For CG algorithm with new $\beta_k`$, the minimum value of the function in example 1 is obtained after eleven iterations. Thus $x = (-0.50018, -0.50006)^T$ is the minimum point and $f(x) = 5.5242e-08$ represents the minimal value and m = (-0.0007, 0.0002)$^T$ ||m||=0.000746. The execution time for the CG algorithm with new $\beta_k`$ with plots is 0.8171378 seconds. The plot of the decrease of $f(x_k)$ per iteration for (CG) is shown in Figure 3 for example 1.

**Example2**: Suppose the nonlinear function $f(x_1, x_2) = e^{x_1^2} + x_2^2 + x_1 x_2 - 4x_1$ such that $x_0 = (0,0)^T, tol = 0.1$. The results for the SD algorithm of example 2 are presented in Table 3 by using MATLAB.

The minimum value of the function in example 2 is obtained by the SD algorithm after14 iterations. Thus $x = (0.931385, -0.447436)^T$ is the minimum point and $f(x)$ =-1.561178 is the minimal value and $m = (-0.012369, 0.036513)^T$ ||m||=0.038551< *tol*. The execution time for the SD algorithm with plots is 0.382256 seconds. The plots of the function and the decrease of $f(x_k)$ per iteration for SD are presented in Figure 4 and 5 respectively for example 2.

The outcomes for the CG algorithm with new $\beta_k`$ of example 2 are presented in Table 4 using MATLAB. For CG algorithm with new $\beta_k`$ the function is minimized within thirteen iterations for example 2. Thus $x = (0.92744, -0.46124)^T$ is the minimum point and $f(x) = -1.5613$ is the minimal value and $m = (-0.0772, 0.005)^T$ $\|m\|$ =0.077352< tol. The execution time for the CG algorithm with new $\beta_k`$, with plots is 0.773001 seconds. The plot of function value per iteration for CG is shown in Figure 6 for example 2.

Table 1: Numerical results of example 1 using the SD algorithm.

| Iteration(k) | $x_1$ | $x_2$ | $\alpha_k$ | $m_1$ | $m_2$ | $\|m\|$ | $f(x_k)$ |
|---|---|---|---|---|---|---|---|
| 0 | 2 | 0.5 | 0.134868 | 9 | -1 | 9.055385 | 10.75 |
| 1 | 0.786184 | 0.634868 | 0.079457 | 0.907895 | 8.171053 | 8.221337 | 5.220395 |
| 2 | 0.714045 | -0.014382 | 0.134868 | 4.370563 | -0.485618 | 4.397459 | 2.535118 |
| 3 | 0.124594 | 0.051113 | 0.079457 | 0.44089 | 3.968011 | 3.99243 | 1.231099 |
| 4 | 0.089562 | -0.264175 | 0.134868 | 2.122425 | -0.235825 | 2.135486 | 0.597844 |
| 5 | -0.196686 | -0.23237 | 0.079457 | 0.214104 | 1.926938 | 1.938796 | 0.290324 |
| 6 | -0.213698 | -0.385479 | 0.134868 | 1.030688 | -0.114521 | 1.037031 | 0.140987 |
| 7 | -0.352705 | -0.370034 | 0.079457 | 0.103973 | 0.935756 | 0.941515 | 0.068466 |
| 8 | -0.360966 | -0.444387 | 0.134868 | 0.500521 | -0.055613 | 0.503601 | 0.033248 |
| 9 | -0.428471 | -0.436886 | 0.079457 | 0.050491 | 0.454420 | 0.457217 | 0.016146 |
| 10 | -0.432483 | -0.472993 | 0.134868 | 0.243062 | -0.027007 | 0.244558 | 0.007841 |
| 11 | -0.465264 | -0.469351 | 0.079457 | 0.024519 | 0.220675 | 0.222033 | 0.003808 |
| 12 | -0.467212 | -0.486885 | 0.134868 | 0.118035 | -0.013115 | 0.118762 | 0.001849 |
| 13 | -0.483132 | -0.485116 | 0.079457 | 0.011907 | 0.107164 | 0.107823 | 0.000898 |
| 14 | -0.484078 | -0.493631 | 0.134868 | 0.057320 | -0.006369 | 0.057673 | 0.000436 |
| 15 | -0.491808 | -0.492772 | 0.079457 | 0.005782 | 0.052041 | 0.052361 | 0.000212 |
| 16 | -0.492268 | -0.496907 | 0.134868 | 0.027836 | -0.003093 | 0.028007 | 0.000103 |
| 17 | -0.496022 | -0.496490 | 0.079457 | 0.002808 | 0.025272 | 0.025427 | 0.000050 |
| 18 | -0.496245 | -0.498498 | 0.134868 | 0.013518 | -0.001502 | 0.013601 | 0.000024 |
| 19 | -0.498068 | -0.498295 | 0.079457 | 0.001364 | 0.012272 | 0.012348 | 0.000012 |
| 20 | -0.498177 | -0.499271 | 0.134868 | 0.006564 | -0.000732 | 0.006605 | 0.000006 |

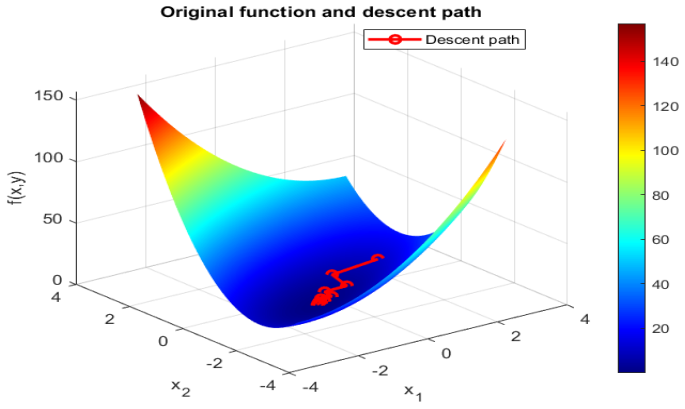


Figure 1: Function $f(x_1,x_2) = 3(x_1-x_2)^2+(2x_2+1)^2$.

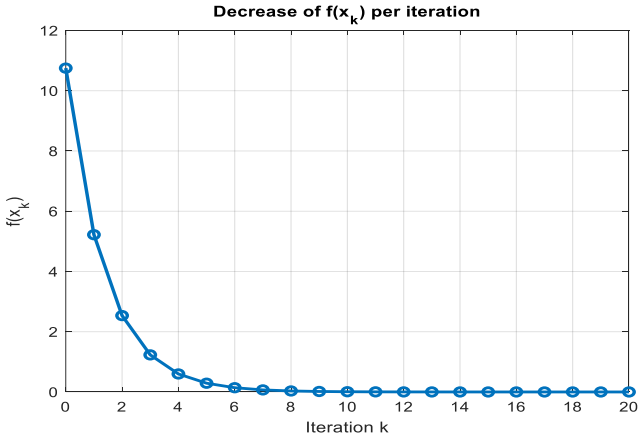


Figure 2: the decrease of $f(x_k)$ per iteration.

Table 2: Numerical results of example1 using the CG algorithm with new proposed $\beta_k^{`}$.

| Iteration (k) | $x_1$ | $x_2$ | $m_1$ | $m_2$ | $\|m\|$ | $\alpha_k$ | *New proposed* $\beta_k^{`}$ | $f(x_k)$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 0.5 | 9 | -1 | 9.0554 | 0.1349 | -1.3991 | 10.7500 |
| 1 | 0.7862 | 0.6349 | 0.9079 | 8.1711 | 8.2213 | 0.0196 | -0.0916 | 5.2204 |
| 2 | 1.0155 | 0.4470 | 3.4113 | 4.1647 | 5.3835 | 0.3050 | -0.6993 | 4.5570 |
| 3 | -0.3512 | -0.5557 | 1.2271 | -1.6727 | 2.0745 | 0.0284 | -0.1291 | 0.1379 |
| 4 | -0.2971 | -0.4431 | 0.8759 | -0.4205 | 0.9716 | 0.1468 | -0.2791 | 0.0769 |
| 5 | -0.4618 | -0.4566 | -0.0311 | 0.3782 | 0.3795 | 0.0369 | -0.0416 | 0.0076 |
| 6 | -0.4491 | -0.4696 | 0.1230 | 0.1202 | 0.1720 | 0.3109 | -1.7995 | 0.0050 |
| 7 | -0.4918 | -0.5024 | 0.0636 | -0.0828 | 0.1044 | 0.0170 | -0.2346 | 3.6011e-04 |
| 8 | -0.4887 | -0.4978 | 0.0545 | -0.0367 | 0.0657 | 0.1213 | -0.0836 | 2.6759e-04 |
| 9 | -0.5005 | -0.5011 | 0.0033 | -0.0120 | 0.0124 | 0.0716 | -0.0026 | 5.5773e-06 |
| 10 | -0.50018 | -0.50006 | -0.0007 | 0.0002 | 0.000746 | 0.071581 | -0.002612 | 5.5242e-08 |

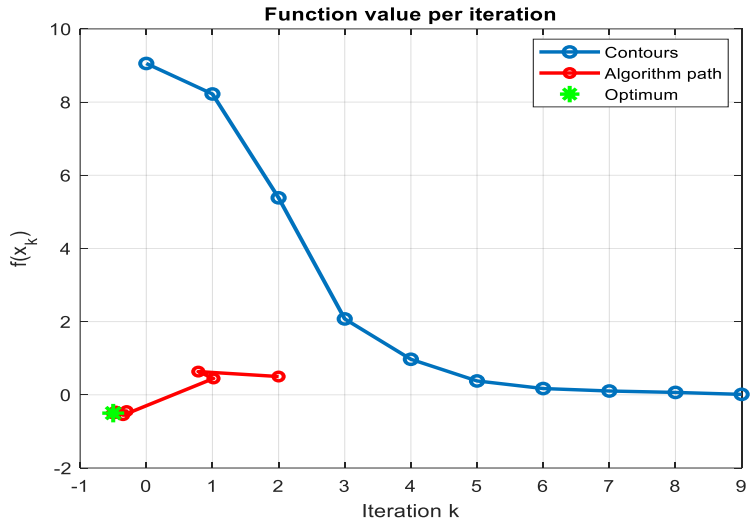


Figure 3: The decrease of $f(x_k)$ per iteration.

Table 3: Numerical results of example 2 using the SD algorithm

| Iteration(k) | $x_1$ | $x_2$ | $\alpha_k$ | $m_1$ | $m_2$ | $\|m\|$ | $f(x_k)$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0.5 | -4 | 0 | 4 | 1 |
| 1 | 2 | 0 | 0.001018 | 214.3926 | 2 | 214.401929 | 46.59815 |
| 2 | 1.781833 | -0.002035 | 0.002844 | 81.259133 | 1.777763 | 81.278577 | 16.794172 |
| 3 | 1.5507 | -0.007092 | 0.007785 | 30.340274 | 1.536517 | 30.379156 | 4.861043 |
| 4 | 1.3145 | -0.019054 | 0.020109 | 10.779273 | 1.276393 | 10.85458 | 0.346197 |
| 5 | 1.097745 | -0.04472 | 0.046449 | 3.281419 | 1.008305 | 3.43284 | -1.101166 |
| 6 | 0.945326 | -0.091555 | 0.148854 | 0.529218 | 0.762216 | 0.927925 | -1.415458 |
| 7 | 0.866549 | -0.205014 | 0.16746 | -0.532712 | 0.456521 | 0.701565 | -1.482899 |
| 8 | 0.955757 | -0.281463 | 0.097383 | 0.483869 | 0.392831 | 0.623254 | -1.519855 |
| 9 | 0.908636 | -0.319718 | 0.250862 | -0.170348 | 0.2692 | 0.318571 | -1.539537 |
| 10 | 0.95137 | -0.38725 | 0.083906 | 0.31669 | 0.17687 | 0.362733 | -1.551744 |
| 11 | 0.924798 | -0.402091 | 0.338589 | -0.051904 | 0.120617 | 0.13131 | -1.557402 |
| 12 | 0.942372 | -0.44293 | 0.079735 | 0.137794 | 0.056512 | 0.148933 | -1.560284 |
| 13 | 0.931385 | -0.447436 | 0.395726 | -0.012369 | 0.036513 | 0.038551 | -1.561178 |

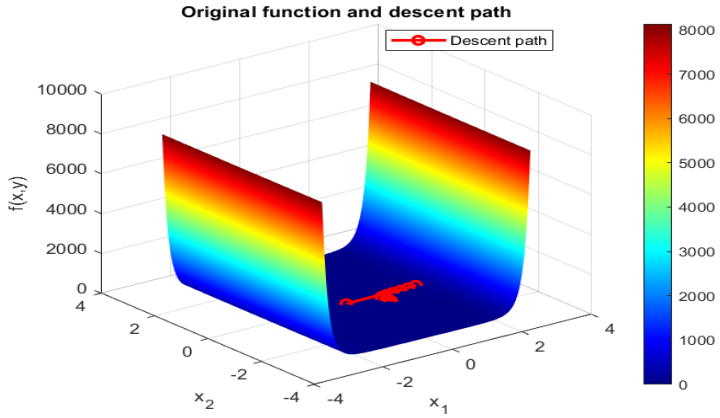


Figure 4: The function f $(x_1, x_2)$= $e^{x_1^2} + x_2^2 + x_1 x_2 - 4x_1$.

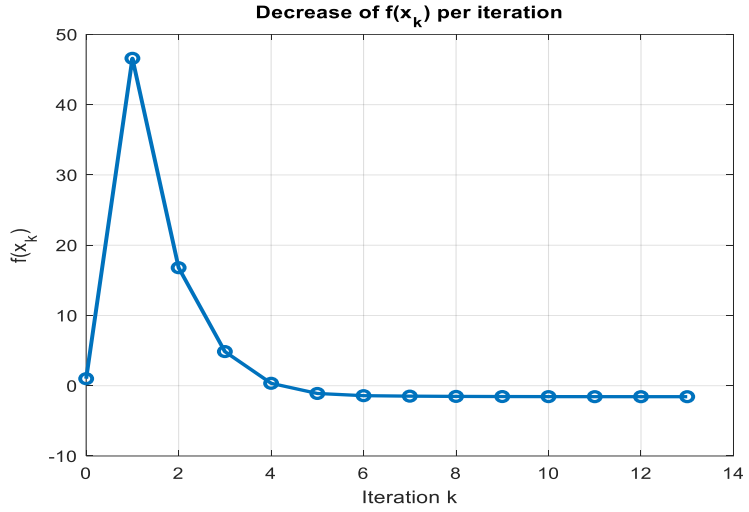


Figure 5: The decrease of $f(x_k)$ per iteration.

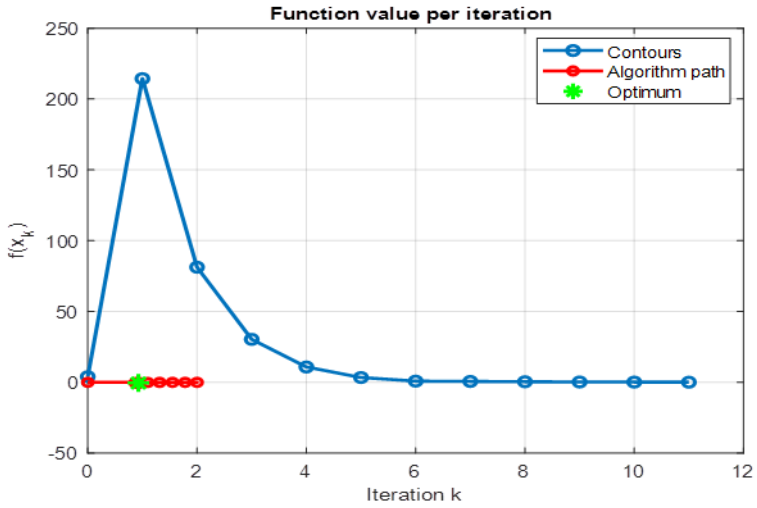


Figure 6: The decrease of $f(x_k)$ per iteration.

Table 4: Numerical results of example 2 using the CG with new proposed $\beta_k^{`}$.

| Iteration (k) | $x_1$ | $x_2$ | $m_1$ | $m_2$ | $\|m\|$ | $\alpha_k$ | New proposed $\beta_k^{`}$ | $f(x_k)$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | -4 | 0 | 4 | 0.5000 | $-2.8728\times10^{3}$ | 1 |
| 1 | 2 | 0 | 214.3926 | 2 | 214.4019 | $1.8636\times10^{-5}$ | -0.0026 | 46.5982 |
| 2 | 1.7818 | $-3.7273\times10^{-5}$ | 81.2662 | 1.7818 | 81.2858 | 0.0046 | -0.2245 | 16.7989 |
| 3 | 1.5507 | -0.0082 | 30.3341 | 1.5343 | 30.3729 | 0.0124 | -0.2018 | 4.8582 |
| 4 | 1.3144 | -0.0223 | 10.7710 | 1.2698 | 10.8456 | 0.0313 | -0.1454 | 0.3410 |
| 5 | 1.0973 | -0.0549 | 3.2609 | 0.9876 | 3.4072 | 0.0678 | -0.0482 | -1.1128 |
| 6 | 0.9445 | -0.1116 | 0.4978 | 0.7213 | 0.8764 | 0.1948 | -0.8427 | -1.4307 |
| 7 | 0.8687 | -0.2442 | -0.5494 | 0.3802 | 0.6681 | 0.0473 | -0.0697 | -1.5005 |
| 8 | 0.9102 | -0.2351 | -0.0669 | 0.4400 | 0.4451 | 0.4918 | -2.1656 | -1.5097 |
| 9 | 0.9130 | -0.4581 | -0.2556 | -0.0032 | 0.2556 | 0.0208 | -0.3488 | -1.5589 |
| 10 | 0.9180 | -0.4377 | -0.1726 | 0.0427 | 0.1778 | 0.0968 | -0.8960 | -1.5595 |
| 11 | 0.9266 | -0.4751 | -0.1025 | -0.0236 | 0.1052 | 0.0374 | -0.8694 | -1.5611 |
| 12 | 0.92744 | -0.46124 | -0.0772 | 0.005 | 0.077352 | 0.037446 | -0.869419 | -1.5613 |

Table 5: Numerical results of example 3 using the SD algorithm.

| Iteration (k) | $x_1$ | $x_2$ | $\alpha_k$ | $m_1$ | $m_2$ | $\|m\|$ | $f(x_k)$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | -1 | 1.414214 | 2 |
| 1 | -1 | 1 | 0.062825 | 1.135335 | 2.301228 | 2.566055 | 0.853617 |
| 2 | -1.071328 | 0.855424 | 0.208857 | 0.569237 | 0.196764 | 0.602285 | 0.622761 |
| 3 | -1.190217 | 0.814329 | 0.153539 | 0.197273 | -0.54437 | 0.579019 | 0.584515 |
| 4 | -1.220506 | 0.897912 | 0.131895 | 0.372945 | 0.239979 | 0.443483 | 0.561621 |
| 5 | -1.269696 | 0.866260 | 0.190182 | 0.17752 | -0.257985 | 0.313160 | 0.54846 |
| 6 | -1.303457 | 0.915324 | 0.106957 | 0.247800 | 0.212098 | 0.326175 | 0.539804 |
| 7 | -1.329961 | 0.892639 | 0.227817 | 0.126321 | -0.137682 | 0.186851 | 0.534039 |
| 8 | -1.358739 | 0.924005 | 0.094910 | 0.156541 | 0.161866 | 0.225179 | 0.530250 |
| 9 | -1.373596 | 0.908642 | 0.264006 | 0.080790 | -0.073401 | 0.109154 | 0.527820 |
| 10 | -1.394925 | 0.928020 | 0.088560 | 0.092196 | 0.108726 | 0.142553 | 0.526294 |
| 11 | -1.403090 | 0.918392 | 0.293885 | 0.047123 | -0.038059 | 0.060573 | 0.525388 |
| 12 | -1.416939 | 0.929577 | 0.085205 | 0.050554 | 0.065065 | 0.082396 | 0.524858 |
| 13 | -1.421246 | 0.924033 | 0.314731 | 0.025426 | -0.019119 | 0.031812 | 0.524567 |
| 14 | -1.429249 | 0.930050 | 0.083479 | 0.026139 | 0.035486 | 0.044074 | 0.524410 |
| 15 | -1.431431 | 0.927088 | 0.327280 | 0.012962 | -0.009365 | 0.015991 | 0.524328 |
| 16 | -1.435673 | 0.930153 | 0.082618 | 0.012984 | 0.018160 | 0.022324 | 0.524287 |
| 17 | -1.436746 | 0.928652 | 0.051225 | 0.006378 | -0.004513 | 0.007813 | 0.524266 |

**Example 3**. Suppose the nonlinear function $f(x_1, x_2) = e_1^{x_2^2} + e^{x_1 - x_2} + 3x_1x_2 + x_1^2$ such that $x_0 = (0,0)^T, tol = 0.01$.

The results for the SD algorithm of example 3 are presented in Table 5 by using MATLAB. The minimum value of the function in example 3 is obtained after 18 iterations for the SD algorithm.

Thus $x = (-1.436746, 0.928652)^T$ is the minimum point and $f(x) = 0.524266$ is the minimal value and:

$m = (0.006378, -0.004513)^{\mathrm{T}}$ ||m||= 0.007813< tol.

The execution time for SD algorithm with plots is 0.696170 seconds. The plots of the function and the decrease of $f(x_k)$ per iteration for SD are presented in Figure 7 and 8 respectively for example 3. The results for the CG algorithm with new proposed $\beta_k^{`}$ of example 3 are presented in Table 6 by using MATLAB. For the CG algorithm with new $\beta_k^{`}$ the minimum value of the function in example 3 is obtained after 12 iterations.

Thus $x = (-1.4425, 0.9309)^{\mathrm{T}}$ is the minimum point and $f(x) = 0.52425$ is the minimal value and $m = (0.0008, 0.0079)^{\mathrm{T}}$, $\|m\| = 0.007932$< tol.

The execution time for the CG algorithm with new proposed $\beta_k^{`}$, with plots is 0.805413 seconds.

The plot of function value per iteration for CG is shown in Figure 9 for example 3.

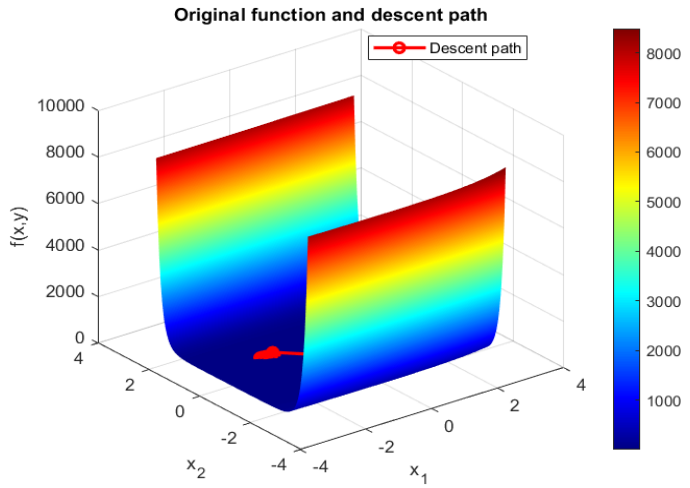


Figure 7: The function $f(x_1, x_2) = e_1^{x_2^2} + e^{x_1 - x_2} + 3x_1x_2 + x_1^2$.

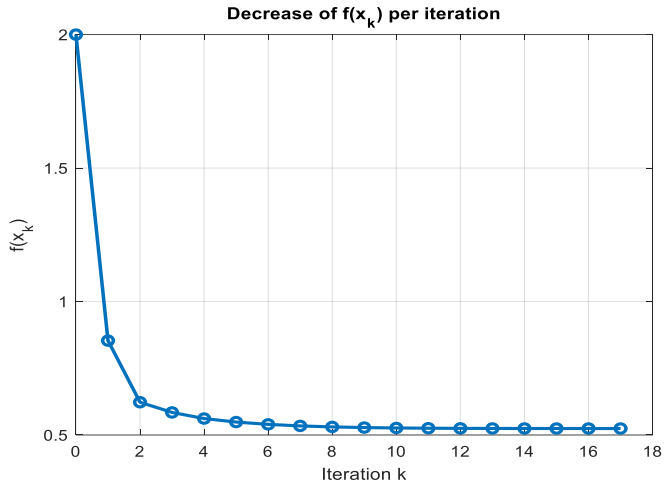


Figure 8: The decrease of $f(x_k)$ per iteration.

Table 6: Numerical results of example 3 using CG algorithm with new proposed $\beta_k^{`}$.

| Iteration (k) | $x_1$ | $x_2$ | $m_1$ | $m_2$ | $\|m\|$ | $\alpha_k$ | New proposed $\beta_k^{`}$ | $f(x_k)$ |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | -1 | 1.4142 | 1 | -8.1556 | 2 |
| 1 | -1 | 1 | 1.1353 | 2.3012 | 2.5661 | 0.0109 | -0.0723 | 0.8536 |
| 2 | -0.9238 | 0.8865 | 0.9755 | 0.9557 | 1.3656 | 0.2376 | -0.2902 | 0.7545 |
| 3 | -1.2762 | 0.8392 | 0.0856 | -0.5553 | 0.5619 | 0.0601 | -0.0252 | 0.5587 |
| 4 | -1.2555 | 0.8760 | 0.2356 | -0.1113 | 0.2606 | 0.7343 | -7.1171 | 0.5496 |
| 5 | -1.4349 | 0.9464 | 0.0617 | 0.2379 | 0.2457 | 0.0135 | -0.1869 | 0.5264 |
| 6 | -1.4123 | 0.9340 | 0.0730 | 0.1363 | 0.1546 | 0.09353 | -0.5696 | 0.5256 |
| 7 | -1.4485 | 0.9373 | 0.0070 | 0.0756 | 0.0759 | 0.05804 | -0.4879 | 0.5245 |
| 8 | -1.4361 | 0.9317 | 0.0167 | 0.0378 | 0.0413 | 0.0665 | -0.5382 | 0.5243 |
| 9 | -1.4441 | 0.9323 | 0.0017 | 0.0223 | 0.0224 | 0.0619 | -0.5804 | 0.5243 |
| 10 | -1.4402 | 0.9307 | 0.0050 | 0.0117 | 0.0127 | 0.05572 | -0.7335 | 0.5243 |
| 11 | -1.4425 | 0.9309 | 0.0008 | 0.0079 | 0.007932 | 0.055721 | -0.733478 | 0.52425 |

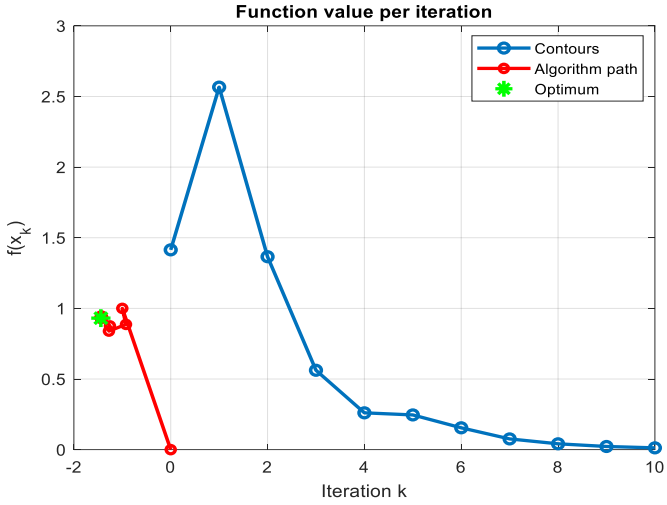


Figure 9: The decrease of $f(x_k)$ per iteration.

# 6 CONCLUSIONS

This study has presented the SD algorithm together with a CG algorithm built around a newly proposed parameter $\beta_k^{`}$, applying both to unconstrained minimisation. Three demanding nonlinear test functions were solved in total: the first by hand as well as in MATLAB, comparing the SD algorithm against the CG algorithm with the new $\beta_k^{`}$, and the second and third using MATLAB alone with the same $\beta_k^{`}$-based CG algorithm. Across all three functions, CG with $\beta_k^{`}$ reaches convergence in fewer iterations than SD and attains more accurate function values; the CG algorithm with $\beta_k^{`}$ is therefore more effective than SD overall, although SD's plotted runtimes remain somewhat shorter than those of CG with $\beta_k^{`}$. Combining the convergence theorem with these MATLAB results, we conclude that the CG algorithm with the proposed $\beta_k^{`}$ is highly effective. Future work should benchmark this proposed $\beta_k^{`}$ directly against the classical Fletcher–Reeves parameter. More broadly, this work lays groundwork for tackling larger, more complex unconstrained problems.

# ACKNOWLEDGMENTS

The authors gratefully acknowledge the support of department of mathematics, college of science, University of Diyala.

# REFERENCES

[1] J. R. Martins and A. Ning, Engineering Design Optimization, Cambridge University Press, 2021, [Online]. Available: https://doi.org/10.1017/9781108980647.

[2] R. K. Arora, Optimization: Algorithms and Applications, CRC Press, 2015, [Online]. Available: https://doi.org/10.5860/CHOICE.195857.

[3] M. A. Hamoda, M. Rivaie, and M. Mamat, “A new nonlinear conjugate gradient method with exact line search for unconstrained optimization,” Journal of Basic Sciences, vol. 30, no. 30, pp. 1-16, 2017, doi: 10.59743/jbs.v30i.74.

[4] M. I. A. Tahseen, “Unconstrained global optimization problems using filled function method with application,” Master’s thesis, Tikrit University, Iraq, 2023.

[5] L. Zhang, “An improved Wei-Yao-Liu nonlinear conjugate gradient method for optimization computation,” Applied Mathematics and Computation, vol. 215, no. 6, pp. 2269-2274, 2009, [Online]. Available: https://doi.org/10.1016/j.amc.2009.08.016.

[6] F. E. Curtis and W. Guo, “Handling nonpositive curvature in a limited memory steepest descent method,” IMA Journal of Numerical Analysis, vol. 36, no. 2, pp. 717-742, 2016, doi: 10.1093/imanum/drv034.

[7] L. G. Drummond and B. F. Svaiter, “A steepest descent method for vector optimization,” Journal of Computational and Applied Mathematics, vol. 175, no. 2, pp. 395-414, 2005, [Online]. Available: https://doi.org/10.1016/j.cam.2004.06.018.

[8] J. R. Shewchuk, An Introduction to the Conjugate Gradient Method Without the Agonizing Pain, 1994.

[9] R. Fletcher, "A limited memory steepest descent method," Mathematical Programming, vol. 135, no. 1, pp. 413-436, 2012, [Online]. Available: https://link.springer.com/article/10.1007/s10107-011-0479-6.

[10] C. C. Gonzaga and R. M. Schneider, "On the steepest descent algorithm for quadratic functions," Computational Optimization and Applications, vol. 63, no. 2, pp. 523-542, 2016, [Online]. Available: https://link.springer.com/article/10.1007/s10589-015-9775-z.

[11] T. E. Abrudan, J. Eriksson, and V. Koivunen, "Steepest descent algorithms for optimization under unitary matrix constraint," IEEE Transactions on Signal Processing, vol. 56, no. 3, pp. 1134-1147, 2008, doi: 10.1109/TSP.2007.908999.

[12] R. Pradhan and B. Subudhi, "A steepest-descent based maximum power point tracking technique for a photovoltaic power system," in 2012 2nd International Conference on Power, Control and Embedded Systems, pp. 1-6, 2012, doi: 10.1109/ICPCES.2012.6508050.

[13] J. Fliege and B. F. Svaiter, "Steepest descent methods for multicriteria optimization," Math. Methods Oper. Res., vol. 51, no. 3, pp. 479-494, 2000, doi: 10.1109/ICPCES.2012.6508050.

[14] J. Barzilai and J. M. Borwein, "Two-point step size gradient methods," IMA Journal of Numerical Analysis, vol. 8, no. 1, pp. 141-148, 1988, [Online]. Available: https://doi.org/10.1093/imanum/8.1.141.

[15] S. S. Djordjevic, "Some unconstrained optimization methods," Applied Mathematics, 2019, doi: 10.5772/intechopen.83679.

[16] M. D. Powell, "Restart procedures for the conjugate gradient method," Mathematical Programming, vol. 12, no. 1, pp. 241-254, 1977, [Online]. Available: https://doi.org/10.1007/BF01593790.

[17] R. Fletcher and M. Reeves, "Function minimization by conjugate gradients," The Computer Journal, vol. 7, no. 2, pp. 149-154, 1964, [Online]. Available: https://doi.org/10.1093/comjnl/7.2.149.

[18] M. R. Hestenes and E. Stiefel, "Methods of conjugate gradients for solving linear systems," Journal of Research of the National Bureau of Standards, vol. 49, no. 6, 1952.

[19] L. Nazareth, "Conjugate gradient method," Wiley Interdisciplinary Reviews, vol. 1, no. 3, pp. 348-353, 2009, [Online]. Available: https://doi.org/10.1002/wics.13.

[20] V. Faber and T. Manteuffel, "Necessary and sufficient conditions for the existence of a conjugate gradient method," SIAM Journal on Numerical Analysis, vol. 21, no. 2, pp. 352-362, 1984, [Online]. Available: https://doi.org/10.1137/0721026.

[21] P. S. Stanimirović, B. Ivanov, H. Ma, and D. Mosić, "A survey of gradient methods for solving nonlinear optimization," Electronic Research Archive, vol. 28, no. 4, p. 1573, 2020, doi: 10.3934/era.2020115.

[22] L. Lasdon, S. Mitter, and A. Waren, "The conjugate gradient method for optimal control problems," IEEE Transactions on Automatic Control, vol. 12, no. 2, pp. 132-138, 1967, doi: 10.1109/TAC.1967.1098538.

[23] Z. B. M. Yunis, "Developing some algorithms for the conjugate gradient in unconstrained optimization," Master's thesis, University of Tikrit, 2020.

[24] D. T. M. Salih and B. M. Faraj, "Comparison between steepest descent method and conjugate gradient method by using MATLAB," Journal of Studies in Science and Engineering, vol. 1, no. 1, pp. 20-31, 2021, [Online]. Available: https://doi.org/10.53898/josse2021113.

[25] S. S. M. Zaki, H. N. Jabbar, and S. S. Haider, "A novel conjugate gradient algorithm as a convex combination of classical conjugate gradient methods," Kurdistan Journal of Applied Research (KJAR), vol. 10, no. 1, pp. 83-98, 2025, doi: 10.24017/science.2025.1.6.

[26] H. N. Jabbar and B. A. Hassan, "Two-versions of descent conjugate gradient methods for large-scale unconstrained optimization," Indonesian Journal of Electrical Engineering and Computer Science, vol. 22, no. 3, 2021, [Online]. Available: http://doi.org/10.11591/ijeecs.v22.i3.pp1643-1649.

[27] Z. Evada and E. Herawati, "Development of the steepest descent method for unconstrained optimization of nonlinear function," Sinkron: Jurnal dan Penelitian Teknik Informatika, vol. 6, no. 3, pp. 2052-2060, 2022, doi: 10.33395/sinkron.v7i3.11596.

[28] L. Muhammad, M. Y. Waziri, I. M. Sulaiman, I. A. Moghrabi, and A. Sambas, "An improved preconditioned conjugate gradient method for unconstrained optimization problem with application in robot arm control," Engineering Reports, vol. 6, no. 12, p. e12968, 2024, doi: 10.1002/eng2.12968.

[29] D. G. Luenberger and Y. Ye, Linear and Nonlinear Programming, 4th ed., pp. 1-11, Springer, 2016, [Online]. Available: https://doi.org/10.1007/978-3-030-85450-8.

[30] O. Axelsson and G. Lindskog, "On the rate of convergence of the preconditioned conjugate gradient method," Numerische Mathematik, vol. 48, no. 5, pp. 499-523.

[31] Y. H. Dai and Y. Yuan, Nonlinear Conjugate Gradient Methods. Shanghai, China: Shanghai Science and Technology Press, 2000.