Research Article

Modified Three-Term Liu–Storey Conjugate Gradient Method for Solving Unconstrained Optimization Problems and Image Restoration Problems

Yulun Wu ¹, Mengxiang Zhang ¹, and Yan Li ²

¹College of Mathematics and Information Science, Guangxi University, Nanning, Guangxi, China

²College of Mathematics and Statistics, Baise University, Baise, Guangxi, China

Correspondence should be addressed to Yan Li; lybsxy@163.com

Received 26 July 2020; Accepted 28 September 2020; Published 19 October 2020

Academic Editor: Wenjie Liu

Copyright © 2020 Yulun Wu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

A new three-term conjugate gradient method is proposed in this article. The new method was able to solve unconstrained optimization problems, image restoration problems, and compressed sensing problems. The method is the convex combination of the steepest descent method and the classical LS method. Without any linear search, the new method has sufficient descent property and trust region property. Unlike previous methods, the information for the function $f(x)$ is assigned to d_k . Next, we make some reasonable assumptions and establish the global convergence of this method under the condition of using the modified Armijo line search. The results of subsequent numerical experiments prove that the new algorithm is more competitive than other algorithms and has a good application prospect.

1. Introduction

Consider the following unconstrained optimization:

$$\min\{f(x) \mid x \in \mathbb{R}^n\}, \quad f: \mathbb{R}^n \longrightarrow \mathbb{R}. \quad (1)$$

According to the research of other scholars, there are abounding effective forms to solve unconstrained optimization problems. For example, there are the steepest descent method, Newton method, and conjugate gradient method [1–8]. The nonlinear conjugate gradient method is a very effective method for solving large-scale unconstrained optimization problems. The conjugate gradient method has attracted more and more attention [9–14] because it is easy to calculate and has low memory requirement and application in many fields [15–18]. The conjugate gradient iteration formula of (1) is defined as

$$x_{k+1} = x_k + \alpha_k d_k, \quad k = 0, 1, 2, \dots, \quad (2)$$

where α_k is the step size of the k th iteration obtained by a certain line search rule and d_k is search direction of step k ,

which are two important factors for solving unconstrained optimization problems. Among them, d_k is defined as

$$d_k := \begin{cases} -g_k, & \text{if } k = 0, \\ -g_k + \beta_k d_{k-1}, & \text{if } k \geq 1, \end{cases} \quad (3)$$

where the parameter β_k called the CG parameter is a scalar and $g_k = g(x_k) = \nabla f(x_k)$. The method of calculating the CG parameters will affect the performance and stability of the whole algorithm, so CG parameters play an utterly significant part. Among the nonlinear conjugate gradient method, we set $y_k = g_k - g_{k-1}$ and let $\|\cdot\|$ represent the Euclidean norm. Then, the classical methods include the following:

HS method [19] (the parameter $\beta_k^{\text{HS}} = g_k^T y_k / -d_{k-1}^T y_k$)

FR method [20] (the parameter $\beta_k^{\text{FR}} = \|g_k\|^2 / \|g_{k-1}\|^2$)

PRP method [21, 22] (the parameter $\beta_k^{\text{PRP}} = g_k^T y_k / \|g_{k-1}\|^2$)

LS method [23] (the parameter $\beta_k^{\text{LS}} = g_k^T y_k / -d_{k-1}^T g_{k-1}$)

CD method [24] (the parameter $\beta_k^{\text{CD}} = \|g_k\|^2 / -d_{k-1}^T g_{k-1}$)
 DY method [25] the parameter ($\beta_k^{\text{DY}} = \|g_k\|^2 / -d_{k-1}^T y_k$)

This paper mainly studies the Liu–Storey method. At the earliest, Liu and Storey proposed a conjugate gradient method to solve the optimization problem in [9]. In this paper, they demonstrate the convergence of the method when using Wolfe line search. The experimental results show that the algorithm is feasible. Later, it was called the LS method. The LS method is the same as the PRP method with exact linear search. Therefore, the LS method and the PRP method have similar forms. As is known to all, the HS method and the PRP method are supposed be the most effective two methods in practical calculation, and a lot of results have been obtained. Consequently, we hope that the theories and analysis techniques to the PRP method can also be applied to the LS method. However, the LS method does not have the nature of automatic descent. In order to surmount the disadvantage, Li [26] proposed an improved Liu–Storey method using Crippo–Lucide search technology, where the direction of search d_k is computed as

$$d_k := \begin{cases} -g_k, & k = 0, \\ -g_k + \beta_k^{\text{NLS}} d_{k-1} + \theta_k (y_k - s_k), & k \geq 1, \end{cases} \quad (4)$$

where $y_k = g_k - g_{k-1}$, $s_k = x_k - x_{k-1}$ and

$$\beta_k^{\text{NLS}} = \frac{g_k^T y_k}{-d_{k-1}^T g_k} = \frac{\|y_k\|^2 g_k^T d_{k-1}}{(d_{k-1}^T g_k)^2}, \quad (5)$$

$$\theta_k = \frac{g_k^T d_{k-1}}{d_{k-1}^T g_k}.$$

Nevertheless, this method assumes a minimum step size, and the main reason is the lack of direction of the trust zone. Therefore, a number of scholars have considered the combination of conjugate gradient method and trust region property. The three-term conjugate gradient algorithm is easy to converge because it automatically has sufficient descent. Therefore, this kind of method has been concerned by many scholars. The conjugate gradient method with three terms was first proposed by Zhang et al. [27]; they defined d_k as

$$d_k := \begin{cases} -g_k, & k = 0, \\ -g_k + \frac{g_k^T y_k d_{k-1} - d_{k-1}^T g_k y_k}{\|g_{k-1}\|^2}, & k \geq 1. \end{cases} \quad (6)$$

Recently, a three-term conjugate gradient algorithm was proposed in Yuan's paper [28]. The search direction d_k of this algorithm is a based on LS method and the gradient descent method. On the basis of Yuan's research, a modified three-term conjugate gradient method is proposed, which has more functional information than the original method, where d_k is computed by

$$d_k := \begin{cases} -g_k, & k = 0, \\ -\theta_k g_k + (1 - \theta_k) \frac{g_k^T y_k^* d_{k-1} - d_{k-1}^T g_k y_k^*}{\max\{2\lambda \|d_{k-1}\| \|y_k^*\|, -d_{k-1}^T g_{k-1}\}}, & k \geq 1, \end{cases} \quad (7)$$

where $y_k = g_k - g_{k-1}$, $s_k = x_k - x_{k-1}$, constant $\lambda \in (0, 1)$, and

$$\theta_k = \frac{\|y_k\|^2}{y_k^T s_k^*},$$

$$s_k^* := s_k + \max\left\{0, \frac{-s_k^T y_k}{\|y_k\|^2}\right\} y_k + y_k,$$

$$y_k^* = y_k + v_k s_k,$$

$$v_k = \max\left\{0, \frac{(g(x_{k+1}) + g(x_k))^T s_k + 2(f(x_k) - f(x_{k+1}))}{\|s_k\|^2}\right\}. \quad (8)$$

The vector y_k^* not only has gradient information but also has functional information, with good theoretical results and numerical performance (see [29]). One might think that the resulting method is indeed better than the original one; that is why we use y_k^* instead of y_k .

According to the meaning of y_k^* and s_k^* , the following inequality is obtained:

$$y_k^T s_k^* \geq y_k^T s_k - \frac{s_k^T y_k}{\|y_k\|^2} \|y_k\|^2 + \|y_k\|^2$$

$$= \|y_{k-1}\|^2 > 0. \quad (9)$$

Therefore,

$$\theta_k := \frac{\|y_k\|^2}{y_k^T s_k^*} \in (0, 1). \quad (10)$$

In [28], Yuan et al. used modified Armijo linear search, which selects the largest α_k as the step size in set $\{\alpha_k | \alpha_k = \overline{\alpha}_k \rho_l^i, l \in \{1, 2, 3, \dots\}\}$ such that

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \delta \alpha_k^2 g_k^T d_k, \quad (11)$$

where $\overline{\alpha}_k$ is the trial step length, which is often set to 1, and $\rho_1 \in (0, 1)$, $\delta \in (0, 1)$ are given constants. As is known to all, Armijo line search technology is a basic and the cheapest method. It is used in various algorithms [30–32]. Basically, a number of other methods of line search can be regarded as a modification of Armijo line search. In an unpublished article by Yuan, a new modification of Armijo linear search was designed based on [16, 33], which is defined as

$$f(x_k + \alpha_k d_k) \leq f_k + \gamma \alpha_k g_k^T d_k + \alpha_k \min\left[-\gamma_1 g_k^T d_k, \gamma \frac{\alpha_k}{2} \|d_k\|^2\right], \quad (12)$$

where $\gamma \in (0, 1)$, $\gamma_1 \in (0, \gamma)$, and $\alpha_k = \max\{\rho^i | i = 0, 1, 2, 3, \dots\}$ satisfying (12). The modification of Armijo

linear search is verified to be effective in improving the efficiency of the algorithm.

The following is an introduction to the paper: In the second section, we propose an improvement algorithm and establish two important properties (sufficient descent property and trust region property) of the algorithm without using linear search. Then, some other properties of the algorithm are given, and the global convergence of the algorithm is proved under appropriate assumptions. The third section gives numerical experiments on normal unconstrained optimization problems, image restoration problems, and compressive sensing problems. In the last section, the conclusion of this paper is given.

2. Conjugate Gradient Algorithm and Convergence Analysis

This section will give a new modified Liu–Storey method combining (7) and (12), as shown below:

Lemma 1. d_k is generated by formula (7); then, it clears that

$$g_k^T d_k \leq -\|g_k\|^2, \quad (13)$$

$$\|g_k\| \leq \|d_k\| \leq \lambda^* \|g_k\|. \quad (14)$$

Proof. When $k = 0$, it is obvious that $g_0^T d_0 = -\|g_0\|^2$ and $\|d_0\| = \|g_0\|$.

When $k > 0$, we have

$$\begin{aligned} g_k^T d_k &= g_k^T \left[-\theta_k g_k + (1 - \theta_k) \frac{g_k^T y_k^* d_{k-1} - d_{k-1}^T g_k y_k^*}{\max\{2\lambda \|d_{k-1}\| \|y_k^*\|, -d_{k-1}^T g_{k-1}\}} \right] \\ &= -\theta_k \|g_k\|^2 + (1 - \theta_k) \frac{g_k^T y_k^* g_k^T d_{k-1} - d_{k-1}^T g_k g_k^T y_k^*}{\max\{2\lambda \|d_{k-1}\| \|y_k^*\|, -d_{k-1}^T g_{k-1}\}} \\ &= -\theta_k \|g_k\|^2. \end{aligned} \quad (15)$$

Using (10), we have $g_k^T d_k < -\|g_k\|^2$, and then, (13) holds. Also,

$$\max\{2\lambda \|d_{k-1}\| \|y_k^*\|, -d_{k-1}^T g_{k-1}\} \geq 2\lambda \|d_{k-1}\| \|y_k^*\|. \quad (16)$$

Then, we have

$$\begin{aligned} \|d_k\| &= \left\| -\theta_k g_k + (1 - \theta_k) \frac{g_k^T y_k^* d_{k-1} - d_{k-1}^T g_k y_k^*}{\max\{2\lambda \|d_{k-1}\| \|y_k^*\|, -d_{k-1}^T g_{k-1}\}} \right\| \\ &\leq \theta_k \|g_k\| + (1 - \theta_k) \frac{\|g_k\| \|y_k^*\| \|d_{k-1}\| + \|d_{k-1}\| \|g_k\| \|y_k^*\|}{\max\{2\lambda \|d_{k-1}\| \|y_k^*\|, -d_{k-1}^T g_{k-1}\}} \\ &\leq \theta_k \|g_k\| + (1 - \theta_k) \frac{\|g_k\| \|y_k^*\| \|d_{k-1}\| + \|d_{k-1}\| \|g_k\| \|y_k^*\|}{2\lambda \|d_{k-1}\| \|y_k^*\|} \\ &\leq \left(\theta_k + \frac{(1 - \theta_k)}{\lambda} \right) \|g_k\|. \end{aligned} \quad (17)$$

Setting $\lambda^* = (\theta_k + ((1 - \theta_k)/\lambda))$, we get the right half of inequality (14). By Cauchy–Schwartz inequality and (13), we obviously get the left half of inequality (14). So, we get the results we want to prove.

Then, we will focus on the global convergence of the algorithm. In order to achieve this goal, we have to make the following assumptions. $\square$

Assumption 1

- (a) The level set $L_0 = \{x | f(x) < f(x_0)\}$ is bounded
- (b) Function: $f(x): \mathfrak{R} \rightarrow \mathfrak{R}$ is continuously differentiable and the gradient $g(x_k)$ is Lipschitz continuous, and there exists a constant (Lipschitz constant) $L > 0$ such that

$$\|g(x) - g(y)\| \leq L\|x - y\|, \quad \forall x, y \in \mathfrak{R}^n. \quad (18)$$

Theorem 1. Assumption 1 is kept true, and then, there is a positive constant α , which satisfies (12) in Algorithm 1.

Proof. We construct such a function

$$\varphi(\alpha) = f(x_k + \alpha d_k) - f_k - \delta \alpha g_k^T d_k - \alpha \min \left[-\delta_1 g_k^T d_k, \delta \frac{\alpha}{2} \|d_k\|^2 \right]. \quad (19)$$

From Assumption 1, $f(x_k + \alpha d_k)$ and f_k are bounded, note that $0 < \delta_1 < \delta < 1$, and from (13), we have $g_k^T d_k \leq 0$. Obviously, $\varphi(0) = 0$ holds. Now, we discuss the following two scenarios:

Case 1: if $\min[-\delta_1 g_k^T d_k, \delta(\alpha/2)\|d_k\|^2] = -\delta_1 g_k^T d_k$, then $\varphi(\alpha) = f(x_k + \alpha d_k) - f_k - \delta \alpha g_k^T d_k + \alpha \delta_1 g_k^T d_k$

For all positive α that are small enough, we can obtain the following inequalities:

$$\begin{aligned} \varphi(\alpha) &= f(x_k + \alpha d_k) - f_k - \delta \alpha g_k^T d_k + \alpha \delta_1 g_k^T d_k \\ &= \alpha(1 - \delta + \delta_1) g_k^T d_k + o(\alpha) \\ &< 0. \end{aligned} \quad (20)$$

Case 2: if $\min[-\delta_1 g_k^T d_k, \delta(\alpha/2)\|d_k\|^2] = \delta(\alpha/2)\|d_k\|^2$, then $\varphi(\alpha) = f(x_k + \alpha d_k) - f_k - \delta \alpha g_k^T d_k - \delta(\alpha^2/2)\|d_k\|^2$

Same as Case 1, we have

$$\begin{aligned} \varphi(\alpha) &= f(x_k + \alpha d_k) - f_k - \delta \alpha g_k^T d_k - \delta \frac{\alpha^2}{2} \|d_k\|^2 \\ &= \alpha(1 - \delta) g_k^T d_k - \delta \frac{\alpha^2}{2} \|d_k\|^2 + o(\alpha) \\ &< 0. \end{aligned} \quad (21)$$

So, from (20) and (21), there exists, at least, a positive constant ω such that $\varphi(\omega) < 0$, which also implies that there

Input: input parameters $\varepsilon \in (0, 1)$, $\gamma \in (0, 1)$, $\gamma_1 \in (0, \gamma)$ and $\rho \in (0, 1)$.
Output: x_k, f_k .
(1) For an initial solution $x_0 \in \mathfrak{R}^n$ compute $d_0 = -g(x_0)$; set $k = 0$;
(2) **while** $\|g_k\| > \varepsilon$ **do**
(3) Find $\alpha_k = \max\{\rho^i | i = 0, 1, 2, 3, \dots\}$ satisfying (12).
(4) Set $x_{k+1} = x_k + \alpha_k d_k$.
(5) Compute d_k by (7).
(6) $k := k + 1$.
(7) **End**

ALGORITHM 1: Three-term modified Liu–Storey conjugate gradient (TTMLS) algorithm.

must be, at least, one local minimum point $\alpha^* \in (0, \omega)$ such that

$$\begin{aligned} \varphi(\alpha^*) &= f(x_k + \alpha^* d_k) - f_k - \delta \alpha^* g_k^T d_k \\ &\quad - \alpha^* \min \left[-\delta_1 g_k^T d_k, \delta \frac{\alpha^*}{2} \|d_k\|^2 \right] < 0, \end{aligned} \quad (22)$$

that is,

$$\begin{aligned} f(x_k + \alpha^* d_k) &< f_k + \delta \alpha^* g_k^T d_k \\ &\quad + \alpha^* \min \left[-\delta_1 g_k^T d_k, \delta \frac{\alpha^*}{2} \|d_k\|^2 \right]. \end{aligned} \quad (23)$$

Thus, α^* satisfies (12). The proof is completed. So, quite evidently, Algorithm 1 is well defined. $\square$

Theorem 2. Let Assumption 1 be satisfied and the iterative sequence $\{x_k, \alpha_k, d_k, g_k\}$ be generated by Algorithm 1. After that, we obtain

$$\lim_{k \rightarrow \infty} \|g_k\| = 0. \quad (24)$$

Proof. We will use the contradiction to draw this conclusion. When (24) is not correct, there exists a constant $\varepsilon_0 > 0$ such that

$$\|g_k\| \geq \varepsilon_0, \quad \forall k > 0. \quad (25)$$

From the third step of the Algorithm (12), we have

$$\begin{aligned} f(x_k + \alpha_k d_k) - f_k &\leq \gamma \alpha_k g_k^T d_k + \alpha_k \min \left[-\gamma_1 g_k^T d_k, \gamma \frac{\alpha_k}{2} \|d_k\|^2 \right] \\ &\leq (\gamma - \gamma_1) \alpha_k g_k^T d_k, \end{aligned} \quad (26)$$

namely,

$$\begin{aligned} (\gamma_1 - \gamma) \alpha_0 g_0^T d_0 &\leq f_0 - f_1, \\ (\gamma_1 - \gamma) \alpha_1 g_1^T d_1 &\leq f_1 - f_2, \\ &\dots \\ (\gamma_1 - \gamma) \alpha_k g_k^T d_k &\leq f_k - f_{k+1}, \\ &\dots, \end{aligned} \quad (27)$$

and summing up the two sides of the abovementioned inequalities and from Assumption 1 (ii), it is obtained that

$$(\gamma - \gamma_1) \sum_{k=0}^{\infty} \alpha_k \|g_k\|^2 \leq (\gamma_1 - \gamma) \sum_{k=0}^{\infty} \alpha_k g_k^T d_k \leq f_0 - f_{\infty} < +\infty. \quad (28)$$

Thus, from (28), we have

$$\alpha_k \|g_k\|^2 \longrightarrow 0, \quad k \longrightarrow \infty. \quad (29)$$

Now, our certificates are divided into the following two cases:

Case 1: the step size $\alpha_k > 0$ as $k \longrightarrow \infty$. From (29), it is obvious that $\|g_k\| \longrightarrow 0$ as $k \longrightarrow \infty$. This contradicts (24).

Case 2: the step size $\alpha_k \longrightarrow 0$ as $k \longrightarrow \infty$.

From Algorithm 1, for the obtained step size α_k , it is clear that α_k/ρ does not satisfy (12), and by (13) and (14),

$$\begin{aligned} f\left(x_k + \frac{\alpha_k}{\rho} d_k\right) - f(x_k) &> \gamma \frac{\alpha_k}{\rho} g_k^T d_k + \frac{\alpha_k}{\rho} \min \left[-\gamma_1 g_k^T d_k, \gamma \frac{\alpha_k}{2\rho} \|d_k\|^2 \right] \\ &> -\gamma \frac{\alpha_k}{\rho} \|g_k\|^2 + \frac{\alpha_k}{\rho} \min \left[\gamma_1 \|g_k\|^2, \gamma \frac{\alpha_k}{2\rho} \|d_k\|^2 \right] \\ &> (\gamma_0 - \gamma) \frac{\alpha_k}{\rho} \|g_k\|^2, \end{aligned} \quad (30)$$

where $\gamma_0 = \min\{\gamma_1, \gamma(\alpha_k/2\rho)\}$; then, according to the mean value theorem, there exists an $\zeta_k \in (0, 1)$ such that

$$\begin{aligned}
f\left(x_k + \frac{\alpha_k}{\rho} d_k\right) - f(x_k) &= \frac{\alpha_k}{\rho} g\left(x_k + \zeta_k \frac{\alpha_k}{\rho} d_k\right)^T d_k \\
&= \frac{\alpha_k}{\rho} g\left(x_k + \zeta_k \frac{\alpha_k}{\rho} d_k\right)^T d_k \\
&\quad - \frac{\alpha_k}{\rho} g(x_k)^T d_k + \frac{\alpha_k}{\rho} g(x_k)^T d_k \\
&\leq \left(\frac{\alpha_k}{\rho}\right)^2 L \zeta_k \|d_k\|^2 + \frac{\alpha_k}{\rho} g(x_k)^T d_k \\
&\leq L \left(\frac{\alpha_k}{\rho}\right)^2 \|d_k\|^2 - \frac{\alpha_k}{\rho} \|g_k\|^2.
\end{aligned} \tag{31}$$

Combined with (30)–(32), we have

$$\alpha_k > \frac{\rho(1 + \gamma_0 - \gamma) \|g_k\|^2}{L \|d_k\|^2} \geq \frac{\rho(1 + \gamma_0 - \gamma)}{L(\lambda^*)^2}. \tag{32}$$

So, we get the contradiction. Thus, result (24) is true, and this completes the proof. $\square$

3. Numerical Experiments

In this section, we have designed three experiments to study the computational efficiency and performance of the proposed algorithm. The first subsection is normal unconstrained optimization problems, the second subsection is the image restoration problem, and the third subsection is compressive sensing problems. All the programs are compiled with Matlab R2017a and implemented on an Intel(R) Core (TM) i7-4710MQ CPU @ 2.50 GHz, RAM 8.0 GB, and the Windows 10 operating system.

3.1. Normal Unconstrained Optimization Problems. In order to test the numerical performance of the TTMLS algorithm, the NLS algorithm [28], LS method with the normal WWP line search (LS-WWP), and PRP method with the normal WWP line search (PRP-WWP) are also experimented as the comparison group. The results can be seen in Tables 1–4. The data used in the experiment are as follows:

Dimension: we choose 3000 dimensions, 6000 dimensions, and 9000 dimensions to test.

Parameters: all the algorithms run with $\gamma = 0.6$, $\gamma_1 = 0.3$, $\rho = 0.9$, $\gamma = 0.7$, and $\varepsilon = 10^{-6}$.

Stop rule (the Himmelblau stop rule [34]): if $|f(x_k)| > e_1$, let $\text{stop}_1 = |f(x_k) - f(x_{k+1})|/f(x_k)$; otherwise, let $\text{stop}_1 = |f(x_k) - f(x_{k+1})|$. If conditions $\|g(x)\| < \varepsilon$ or $\text{stop}_1 < e_2$ are satisfied or the iteration number of more than 1000 is satisfied, we stop the process, where $e_1 = e_2 = 10^{-5}$.

Symbol representation: NI: the iteration number. NFG: the total number of function and gradient evaluations. CPU: the CPU time in seconds.

Text problems: we have tested 74 unconstrained optimization problems, and the list of problems can be seen in Yuan's work [16].

Dolan and Moré [35] provided a way to analyze the efficiency of these algorithms. From Figures 1–3, we can see that the performance of the TTMLS algorithm, TTMLS algorithm, and NLS algorithm is significantly better than that of the LS-WWP method and PRP-WWP method. Figures 1 and 2 show that the TTMLS algorithm and NLS algorithm can better approximate to the target function than the LS-WWP algorithm and PRP-WWP algorithm; thus, the number of iterations and the total number of function and gradient evaluation are smaller. The reason is that the search direction d_k of the TTMLS algorithm contains more function information. Also, the CPU time in Figure 4, TTMLS algorithm is basically the same as the NLS algorithm, which is better than the other two. To sum up, the proposed algorithm has significant advantages.

3.2. Image Restoration Problems. The image restoration problem is a difficult problem in the field of optimization. We will use the TTMLS algorithm and NLS algorithm to minimize Z to recover the original image from an image corrupted by impulse noise. Afterwards, we compare the performance of the two algorithms. The data used in the experiment are as follows:

Parameters: all the algorithms run with $\gamma = 0.6$, $\gamma_1 = 0.3$, $\rho = 0.9$, and $\gamma = 0.7$.

Stop rule: if $\|g_{k+1} - g_k\|/\|g_k\| < 10^{-4}$ or $\|x_{k+1} - x_k\|/\|x_k\| < 10^{-4}$ is satisfied, we stop the process.

Symbol representation: CPU: the CPU time in seconds.

Total: the total CPU time of the four pictures.

The information of noise: 30%, 45%, and 60% salt-and-pepper noise.

Text problems: we restore the original image from the image destroyed by impulsive noise. The experiments chose Lena (512×512), Barbara (512×512), Man (1024×1024), and Baboon (512×512) as the test images.

From Figures 4–6, we can Table 5 see that both algorithms can recover 30%, 45%, and 60% salt-and-pepper noise images very well. The data show that, for image restoration problems, the TTMLS algorithm has shorter CPU time than the NLS algorithm when the salt-and-pepper noise is 30%, 45%, and 60%. In conclusion, the TTMLS algorithm is promising and competitive.

3.3. Compressive Sensing Problems. The main work of this section is to accurately recover the image from a few of random projections by compressive sensing. The experimental method derives from the model proposed by Dai and Sha [36]. Then, the performance of the TTMLS algorithm and LS method with line search (12) is compared.

It is noted that the gradients g_k and d_k are square matrices in this experiment, and the matrix obtained by

TABLE 1: Test results of the TTMLS algorithm.

Nr	Dim	TTMLS algorithm		
		NI	NFG	CPU
1	3000	5	26	0.203125
1	6000	5	26	0.109375
1	9000	5	26	0.234375
2	3000	85	402	3.640625
2	6000	90	426	6.171875
2	9000	91	434	9.125
3	3000	6	32	0.0625
3	6000	6	32	0.0625
3	9000	6	32	0.046875
4	3000	5	26	0.0625
4	6000	5	26	0.15625
4	9000	5	26	0.171875
5	3000	5	26	0.015625
5	6000	5	26	0.046875
5	9000	5	26	0.0625
6	3000	4	20	0.0625
6	6000	4	20	0.0625
6	9000	4	20	0.0625
7	3000	78	342	0.140625
7	6000	80	353	0.234375
7	9000	80	354	0.328125
8	3000	10	40	0.03125
8	6000	10	40	0.0625
8	9000	10	40	0.0625
9	3000	6	20	0
9	6000	6	20	0.0625
9	9000	6	20	0
10	3000	3	14	0.0625
10	6000	3	14	0.0625
10	9000	3	14	0.0625
11	3000	1000	2999	2.359375
11	6000	1000	2999	4.625
11	9000	1000	2999	6.859375
12	3000	3	14	0.0625
12	6000	3	14	0.0625
12	9000	3	14	0.0625
13	3000	4	17	0
13	6000	4	17	0.0625
13	9000	4	20	0.109375
14	3000	7	38	0.25
14	6000	7	38	0.375
14	9000	7	38	0.46875
15	3000	7	38	0.1875
15	6000	7	38	0.265625
15	9000	7	38	0.390625
16	3000	4	20	0
16	6000	4	20	0.046875
16	9000	4	20	0.0625
17	3000	5	26	0.125
17	6000	5	26	0.140625
17	9000	5	26	0.25
18	3000	1000	3073	0.8125
18	6000	1000	3073	1.421875
18	9000	1000	3073	1.984375
19	3000	10	29	0.0625
19	6000	10	29	0.0625
19	9000	10	29	0.125
20	3000	7	38	0.0625
20	6000	7	38	0.0625

TABLE 1: Continued.

Nr	Dim	TTMLS algorithm		
		NI	NFG	CPU
20	9000	7	38	0.109375
21	3000	6	32	0.1875
21	6000	6	32	0.265625
21	9000	6	32	0.34375
22	3000	9	50	0.15625
22	6000	9	50	0.25
22	9000	9	50	0.390625
23	3000	6	32	0.125
23	6000	6	32	0.125
23	9000	6	32	0.171875
24	3000	232	714	0.4375
24	6000	243	747	0.890625
24	9000	249	765	1.296875
25	3000	6	32	0.0625
25	6000	6	32	0.0625
25	9000	6	32	0.0625
26	3000	4	14	0
26	6000	4	14	0
26	9000	4	14	0.0625
27	3000	79	346	0.125
27	6000	80	353	0.234375
27	9000	81	358	0.28125
28	3000	5	26	0
28	6000	5	26	0
28	9000	5	26	0.046875
29	3000	7	38	0.015625
29	6000	7	38	0.0625
29	9000	7	38	0.0625
30	3000	76	333	0.0625
30	6000	78	342	0.171875
30	9000	78	344	0.203125
31	3000	5	26	0.03125
31	6000	5	26	0
31	9000	5	26	0.0625
32	3000	5	26	0.125
32	6000	5	26	0.1875
32	9000	5	26	0.1875
33	3000	6	32	0
33	6000	6	32	0
33	9000	6	32	0.0625
34	3000	5	26	0.0625
34	6000	5	26	0.0625
34	9000	5	26	0.046875
35	3000	6	28	0
35	6000	6	28	0
35	9000	6	28	0
36	3000	5	26	0.078125
36	6000	5	26	0.125
36	9000	5	26	0.21875
37	3000	82	366	0.078125
37	6000	84	378	0.140625
37	9000	85	382	0.34375
38	3000	5	26	0
38	6000	5	26	0
38	9000	5	26	0.0625
39	3000	5	26	0.0625
39	6000	5	26	0.0625
39	9000	5	26	0.0625
40	3000	5	26	0.125

TABLE 1: Continued.

Nr	Dim	TTMLS algorithm		
		NI	NFG	CPU
40	6000	5	26	0.15625
40	9000	5	26	0.25
41	3000	73	312	0.0625
41	6000	73	312	0.140625
41	9000	73	312	0.265625
42	3000	37	204	0.1875
42	6000	30	170	0.296875
42	9000	43	238	0.59375
43	3000	5	26	0.140625
43	6000	5	26	0.25
43	9000	5	26	0.359375
44	3000	5	26	0.1875
44	6000	5	26	0.25
44	9000	5	26	0.265625
45	3000	5	26	0.140625
45	6000	5	26	0.21875
45	9000	5	26	0.265625
46	3000	5	26	0.125
46	6000	5	26	0.25
46	9000	5	26	0.28125
47	3000	82	368	3.9375
47	6000	85	386	11.29688
47	9000	87	396	23.98438
48	3000	6	32	0.1875
48	6000	6	32	0.171875
48	9000	6	32	0.3125
49	3000	78	342	0.0625
49	6000	80	353	0.1875
49	9000	80	354	0.296875
50	3000	78	342	0.9375
50	6000	80	353	1.8125
50	9000	80	354	2.234375
51	3000	7	38	0.125
51	6000	7	38	0.28125
51	9000	7	38	0.375
52	3000	4	20	0.0625
52	6000	4	20	0.125
52	9000	4	20	0.0625
53	3000	1000	3032	0.875
53	6000	1000	3032	1.609375
53	9000	1000	3033	2.4375
54	3000	5	26	0
54	6000	5	26	0
54	9000	5	26	0.046875
55	3000	24	74	0.109375
55	6000	26	80	0.375
55	9000	26	80	0.5
56	3000	1000	3010	0.90625
56	6000	1000	3010	1.6875
56	9000	1000	3010	2.46875
57	3000	5	26	0.125
57	6000	5	26	0.21875

TABLE 1: Continued.

Nr	Dim	TTMLS algorithm		
		NI	NFG	CPU
57	9000	5	26	0.265625
58	3000	5	26	0.125
58	6000	5	26	0.21875
58	9000	5	26	0.25
59	3000	5	26	0.125
59	6000	5	26	0.234375
59	9000	5	26	0.28125
60	3000	5	26	0.125
60	6000	5	26	0.25
60	9000	5	26	0.28125
61	3000	5	26	0.1875
61	6000	5	26	0.25
61	9000	5	26	0.3125
62	3000	5	26	0.1875
62	6000	5	26	0.25
62	9000	5	26	0.359375
63	3000	5	26	0.1875
63	6000	5	26	0.25
63	9000	5	26	0.3125
64	3000	5	26	0.140625
64	6000	5	26	0.25
64	9000	5	26	0.28125
65	3000	6	32	0.203125
65	6000	6	32	0.3125
65	9000	6	32	0.28125
66	3000	6	32	0.078125
66	6000	6	32	0.1875
66	9000	6	32	0.25
67	3000	12	61	0.0625
67	6000	14	74	0.09375
67	9000	13	69	0.1875
68	3000	11	62	0.0625
68	6000	11	62	0.0625
68	9000	11	62	0.0625
69	3000	6	32	0.03125
69	6000	6	32	0.0625
69	9000	6	32	0.0625
70	3000	6	32	0.1875
70	6000	6	32	0.25
70	9000	6	32	0.34375
71	3000	1000	3010	1.015625
71	6000	1000	3010	1.8125
71	9000	1000	3010	2.59375
72	3000	82	368	4.1875
72	6000	86	389	11.51563
72	9000	87	396	23.92188
73	3000	76	333	0.09375
73	6000	78	342	0.15625
73	9000	78	344	0.234375
74	3000	76	333	0.0625
74	6000	78	342	0.125
74	9000	78	344	0.21875

TABLE 2: Test results of the NLS algorithm.

Nr	Dim	NLS algorithm		
		NI	NFG	CPU
1	3000	5	26	0.125
1	6000	5	26	0.140625
1	9000	5	26	0.21875
2	3000	92	428	4.078125
2	6000	102	472	7.15625
2	9000	97	457	7.84375
3	3000	6	32	0.0625
3	6000	6	32	0
3	9000	6	32	0
4	3000	5	26	0.0625
4	6000	5	26	0.109375
4	9000	5	26	0.109375
5	3000	5	26	0.0625
5	6000	5	26	0.0625
5	9000	5	26	0.0625
6	3000	4	20	0
6	6000	4	20	0.0625
6	9000	4	20	0.0625
7	3000	83	361	0.125
7	6000	85	370	0.25
7	9000	86	377	0.34375
8	3000	1000	3046	2.078125
8	6000	10	40	0.0625
8	9000	10	40	0.0625
9	3000	6	20	0.0625
9	6000	6	20	0
9	9000	6	20	0.0625
10	3000	3	14	0
10	6000	3	14	0
10	9000	3	14	0.0625
11	3000	1000	2999	2.8125
11	6000	1000	2999	5.296875
11	9000	1000	2999	7.8125
12	3000	3	14	0
12	6000	3	14	0.03125
12	9000	3	14	0
13	3000	4	17	0
13	6000	4	17	0.0625
13	9000	4	20	0.0625
14	3000	7	38	0.203125
14	6000	7	38	0.375
14	9000	7	38	0.453125
15	3000	7	38	0.1875
15	6000	7	38	0.3125
15	9000	7	38	0.34375
16	3000	4	20	0
16	6000	4	20	0
16	9000	4	20	0.0625
17	3000	5	26	0.125
17	6000	5	26	0.140625
17	9000	5	26	0.234375
18	3000	1000	3075	0.75
18	6000	1000	3075	1.234375
18	9000	1000	3075	2.046875
19	3000	10	29	0
19	6000	10	29	0.0625
19	9000	10	29	0.1875
20	3000	7	38	0.0625
20	6000	7	38	0.0625

TABLE 2: Continued.

Nr	Dim	NLS algorithm		
		NI	NFG	CPU
20	9000	7	38	0.0625
21	3000	6	32	0.125
21	6000	6	32	0.265625
21	9000	6	32	0.375
22	3000	9	50	0.1875
22	6000	9	50	0.25
22	9000	9	50	0.28125
23	3000	6	32	0.0625
23	6000	6	32	0.125
23	9000	6	32	0.125
24	3000	505	1535	1.109375
24	6000	529	1607	2.3125
24	9000	542	1646	3.515625
25	3000	6	32	0.0625
25	6000	6	32	0
25	9000	6	32	0.046875
26	3000	4	14	0
26	6000	4	14	0.0625
26	9000	4	14	0
27	3000	84	365	0.125
27	6000	85	370	0.3125
27	9000	85	372	0.359375
28	3000	5	26	0.0625
28	6000	5	26	0
28	9000	5	26	0.03125
29	3000	7	38	0
29	6000	7	38	0
29	9000	7	38	0.0625
30	3000	83	357	0.0625
30	6000	83	361	0.1875
30	9000	84	366	0.21875
31	3000	5	26	0.0625
31	6000	5	26	0
31	9000	5	26	0
32	3000	5	26	0.125
32	6000	5	26	0.125
32	9000	5	26	0.125
33	3000	6	32	0.0625
33	6000	6	32	0
33	9000	6	32	0.0625
34	3000	5	26	0.0625
34	6000	5	26	0
34	9000	5	26	0
35	3000	7	33	0
35	6000	7	33	0
35	9000	7	33	0.0625
36	3000	5	26	0.125
36	6000	5	26	0.125
36	9000	5	26	0.1875
37	3000	87	384	0.125
37	6000	90	398	0.25
37	9000	89	396	0.390625
38	3000	5	26	0.0625
38	6000	5	26	0
38	9000	5	26	0.0625
39	3000	5	26	0.0625
39	6000	5	26	0
39	9000	5	26	0.046875
40	3000	5	26	0.125

TABLE 2: Continued.

Nr	Dim	NLS algorithm		
		NI	NFG	CPU
40	6000	5	26	0.140625
40	9000	5	26	0.25
41	3000	79	334	0.125
41	6000	79	334	0.171875
41	9000	79	334	0.25
42	3000	46	235	0.25
42	6000	39	204	0.4375
42	9000	55	272	0.71875
43	3000	5	26	0.125
43	6000	5	26	0.25
43	9000	5	26	0.3125
44	3000	5	26	0.125
44	6000	5	26	0.15625
44	9000	5	26	0.28125
45	3000	5	26	0.109375
45	6000	5	26	0.203125
45	9000	5	26	0.3125
46	3000	5	26	0.078125
46	6000	5	26	0.21875
46	9000	5	26	0.375
47	3000	88	390	6.078125
47	6000	91	406	19.84375
47	9000	92	415	41.73438
48	3000	6	32	0.125
48	6000	6	32	0.1875
48	9000	6	32	0.234375
49	3000	83	361	0.125
49	6000	85	370	0.171875
49	9000	86	377	0.1875
50	3000	83	361	1.125
50	6000	85	370	2.0625
50	9000	86	377	2.453125
51	3000	7	38	0.1875
51	6000	7	38	0.25
51	9000	7	38	0.296875
52	3000	4	20	0.0625
52	6000	4	20	0.0625
52	9000	4	20	0.015625
53	3000	1000	3015	1.03125
53	6000	1000	3016	1.8125
53	9000	1000	3016	2.859375
54	3000	5	26	0
54	6000	5	26	0.0625
54	9000	5	26	0
55	3000	24	74	0.1875
55	6000	26	80	0.421875
55	9000	26	80	0.515625
56	3000	788	2374	0.875
56	6000	788	2374	1.59375
56	9000	788	2374	2.234375
57	3000	5	26	0.125
57	6000	5	26	0.203125

TABLE 2: Continued.

Nr	Dim	NLS algorithm		
		NI	NFG	CPU
57	9000	5	26	0.28125
58	3000	5	26	0.125
58	6000	5	26	0.21875
58	9000	5	26	0.34375
59	3000	5	26	0.125
59	6000	5	26	0.25
59	9000	5	26	0.234375
60	3000	5	26	0.125
60	6000	5	26	0.234375
60	9000	5	26	0.34375
61	3000	5	26	0.109375
61	6000	5	26	0.203125
61	9000	5	26	0.28125
62	3000	5	26	0.125
62	6000	5	26	0.25
62	9000	5	26	0.359375
63	3000	5	26	0.125
63	6000	5	26	0.25
63	9000	5	26	0.265625
64	3000	5	26	0.1875
64	6000	5	26	0.25
64	9000	5	26	0.25
65	3000	6	32	0.1875
65	6000	6	32	0.28125
65	9000	6	32	0.328125
66	3000	6	32	0.125
66	6000	6	32	0.1875
66	9000	6	32	0.234375
67	3000	22	99	0.125
67	6000	14	65	0.125
67	9000	21	95	0.234375
68	3000	11	62	0.0625
68	6000	11	62	0.0625
68	9000	11	62	0.0625
69	3000	6	32	0
69	6000	6	32	0.0625
69	9000	6	32	0.0625
70	3000	6	32	0.125
70	6000	6	32	0.203125
70	9000	6	32	0.375
71	3000	788	2374	0.8125
71	6000	788	2374	1.46875
71	9000	788	2374	2.265625
72	3000	88	390	5.984375
72	6000	91	406	19.34375
72	9000	92	415	42.32813
73	3000	83	357	0.125
73	6000	83	361	0.09375
73	9000	84	366	0.203125
74	3000	83	357	0.0625
74	6000	83	361	0.15625
74	9000	84	366	0.203125

TABLE 3: Test results of the LS-WWP algorithm.

Nr	Dim	LS-WWP		
		NI	NFG	CPU
1	3000	1000	5998	20.5625
1	6000	1000	5986	39.26563
1	9000	1000	5986	52.96875
2	3000	63	228	0.375
2	6000	66	231	0.765625
2	9000	61	223	1.046875
3	3000	325	1076	0.3125
3	6000	1000	5206	1.828125
3	9000	1000	5685	2.65625
4	3000	1000	5934	11.5
4	6000	1000	5835	22.32813
4	9000	1000	5841	31.6875
5	3000	132	819	0.4375
5	6000	133	826	0.765625
5	9000	134	832	1.203125
6	3000	48	198	0.125
6	6000	51	207	0.125
6	9000	43	202	0.1875
7	3000	1000	5994	1.71875
7	6000	1000	5994	3
7	9000	1000	5994	4.375
8	3000	33	136	0.125
8	6000	33	136	0.125
8	9000	32	130	0.140625
9	3000	4	19	0.0625
9	6000	4	19	0
9	9000	4	19	0.0625
10	3000	2	9	0
10	6000	2	9	0
10	9000	2	9	0.0625
11	3000	1000	3001	2.671875
11	6000	1000	3001	5.046875
11	9000	1000	3001	7.828125
12	3000	18	105	0.125
12	6000	19	111	0.25
12	9000	19	111	0.359375
13	3000	3	13	0
13	6000	13	66	0.1875
13	9000	2	9	0
14	3000	6	33	0.1875
14	6000	5	27	0.265625
14	9000	5	27	0.28125
15	3000	1000	3010	19
15	6000	1000	3010	29.40625
15	9000	1000	3010	35.84375
16	3000	22	72	0.0625
16	6000	15	84	0.125
16	9000	15	81	0.15625
17	3000	74	432	1.625
17	6000	74	432	2.96875
17	9000	74	432	4.046875
18	3000	7	42	0.0625
18	6000	8	48	0
18	9000	8	48	0.0625
19	3000	4	18	0
19	6000	4	18	0.0625
19	9000	4	18	0.09375
20	3000	14	74	0
20	6000	14	74	0.046875

TABLE 3: Continued.

Nr	Dim	LS-WWP		
		NI	NFG	CPU
20	9000	14	74	0.0625
21	3000	45	150	0.0625
21	6000	21	111	0.125
21	9000	20	111	0.171875
22	3000	12	58	0
22	6000	12	58	0.125
22	9000	12	58	0.09375
23	3000	1000	5959	11.54688
23	6000	1000	5959	22.4375
23	9000	1000	5959	31.98438
24	3000	59	313	0.1875
24	6000	62	274	0.3125
24	9000	7	46	0.0625
25	3000	33	98	0.0625
25	6000	44	131	0.0625
25	9000	1000	3073	1.921875
26	3000	51	194	0.125
26	6000	50	188	0.25
26	9000	50	188	0.375
27	3000	1000	3013	1.15625
27	6000	1000	3013	2.1875
27	9000	1000	3013	3.140625
28	3000	92	503	0.0625
28	6000	92	503	0.1875
28	9000	92	503	0.1875
29	3000	3	15	0
29	6000	3	15	0
29	9000	3	15	0
30	3000	1000	5994	1.28125
30	6000	1000	5994	2.453125
30	9000	1000	5994	3.328125
31	3000	10	51	0
31	6000	12	64	0.0625
31	9000	12	64	0.0625
32	3000	10	52	0.0625
32	6000	1000	5935	5.921875
32	9000	10	54	0.0625
33	3000	3	9	0.0625
33	6000	3	9	0
33	9000	2	6	0
34	3000	3	15	0
34	6000	7	20	0
34	9000	8	23	0.0625
35	3000	15	44	0
35	6000	19	56	0.046875
35	9000	21	62	0.0625
36	3000	1000	5988	17.95313
36	6000	1000	5997	32.17188
36	9000	1000	5991	43.54688
37	3000	1000	5991	2.0625
37	6000	1000	5988	3.84375
37	9000	1000	5988	5.421875
38	3000	6	31	0
38	6000	6	31	0
38	9000	6	31	0
39	3000	42	125	0
39	6000	44	131	0.125
39	9000	46	137	0.09375
40	3000	1000	5982	25.26563

TABLE 3: Continued.

Nr	Dim	NI	LS-WWP	
			NFG	CPU
40	6000	1000	5898	41.26563
40	9000	1000	5867	55.25
41	3000	1000	3003	0.96875
41	6000	1000	3003	1.640625
41	9000	1000	3003	2.421875
42	3000	6	26	0
42	6000	6	26	0.0625
42	9000	7	32	0.0625
43	3000	8	46	0.25
43	6000	8	46	0.4375
43	9000	8	46	0.5625
44	3000	12	68	0.328125
44	6000	11	62	0.46875
44	9000	11	62	0.6875
45	3000	9	50	0.25
45	6000	9	50	0.375
45	9000	9	50	0.5625
46	3000	563	3367	16.0625
46	6000	708	4237	33.29688
46	9000	812	4864	51.15625
47	3000	144	435	11.59375
47	6000	293	882	72.65625
47	9000	446	1341	230.6875
48	3000	51	294	0.8125
48	6000	50	288	1.5
48	9000	50	288	2.078125
49	3000	1000	5994	1.296875
49	6000	1000	5994	2.328125
49	9000	1000	5994	3.359375
50	3000	1000	5994	17.14063
50	6000	1000	5994	31.34375
50	9000	1000	5994	41.54688
51	3000	14	50	0.1875
51	6000	13	47	0.375
51	9000	13	47	0.40625
52	3000	39	208	0.25
52	6000	42	224	0.5
52	9000	43	227	0.71875
53	3000	1000	3003	0.9375
53	6000	1000	3003	1.609375
53	9000	1000	3003	2.453125
54	3000	1000	5955	1.4375
54	6000	1000	5958	2.59375
54	9000	1000	5955	3.96875
55	3000	6	32	0.0625
55	6000	6	32	0.1875
55	9000	6	32	0.1875
56	3000	1000	2999	1.0625
56	6000	1000	2999	2.0625
56	9000	1000	2999	2.75
57	3000	9	50	0.25
57	6000	9	50	0.375

TABLE 3: Continued.

Nr	Dim	NI	LS-WWP	
			NFG	CPU
57	9000	9	50	0.578125
58	3000	53	314	1.546875
58	6000	45	266	2.1875
58	9000	41	242	2.734375
59	3000	8	44	0.25
59	6000	8	44	0.328125
59	9000	8	44	0.5
60	3000	563	3367	16.15625
60	6000	710	4252	33.78125
60	9000	1000	3013	43.10938
61	3000	9	50	0.25
61	6000	9	50	0.390625
61	9000	9	50	0.53125
62	3000	6	32	0.140625
62	6000	6	32	0.3125
62	9000	6	32	0.296875
63	3000	6	32	0.125
63	6000	6	32	0.265625
63	9000	5	26	0.265625
64	3000	8	44	0.1875
64	6000	8	44	0.328125
64	9000	8	44	0.5625
65	3000	10	46	0.1875
65	6000	10	46	0.34375
65	9000	9	43	0.453125
66	3000	1000	5989	16.51563
66	6000	1000	5989	29.70313
66	9000	1000	5989	40.28125
67	3000	27	108	0.0625
67	6000	18	78	0.109375
67	9000	21	112	0.9375
68	3000	45	136	0
68	6000	47	142	0.0625
68	9000	48	145	0.0625
69	3000	36	114	0.046875
69	6000	37	117	0.046875
69	9000	37	117	0.0625
70	3000	1000	3701	14.15625
70	6000	1000	3125	23.60938
70	9000	1000	3188	31.20313
71	3000	1000	5996	1.375
71	6000	1000	5996	2.546875
71	9000	1000	5996	3.71875
72	3000	1000	3010	82
72	6000	1000	3010	246.3281
72	9000	1000	3010	519.7656
73	3000	1000	5994	1.3125
73	6000	1000	5994	2.46875
73	9000	1000	5994	3.375
74	3000	1000	5997	1.25
74	6000	1000	5997	2.390625
74	9000	1000	5994	3.21875

TABLE 4: Test results of the PRP-WWP algorithm.

Nr	Dim	PRP-WWP		
		NI	NFG	CPU
1	3000	11	67	0.3125
1	6000	22	89	0.625
1	9000	21	86	0.78125
2	3000	70	242	0.4375
2	6000	80	263	0.90625
2	9000	76	255	1.21875
3	3000	33	109	0
3	6000	7	36	0
3	9000	6	36	0
4	3000	7	42	0.125
4	6000	10	56	0.25
4	9000	12	71	0.40625
5	3000	19	88	0.0625
5	6000	19	88	0.109375
5	9000	19	88	0.15625
6	3000	70	255	0.109375
6	6000	74	274	0.171875
6	9000	85	304	0.296875
7	3000	1000	3044	1.0625
7	6000	1000	3044	2.09375
7	9000	1000	3044	2.8125
8	3000	23	97	0.0625
8	6000	23	97	0.125
8	9000	23	97	0.15625
9	3000	5	25	0
9	6000	5	25	0
9	9000	5	25	0
10	3000	2	9	0
10	6000	2	9	0
10	9000	2	9	0
11	3000	669	2024	1.765625
11	6000	21	86	0.125
11	9000	1000	3018	7.546875
12	3000	19	70	0.125
12	6000	19	70	0.171875
12	9000	19	70	0.203125
13	3000	3	13	0
13	6000	10	54	0.1875
13	9000	2	9	0.0625
14	3000	6	33	0.171875
14	6000	5	27	0.25
14	9000	5	27	0.359375
15	3000	31	138	0.75
15	6000	55	233	2.046875
15	9000	39	177	1.90625
16	3000	10	42	0
16	6000	21	66	0.125
16	9000	22	69	0.1875
17	3000	40	136	0.5625
17	6000	41	139	1.171875
17	9000	44	164	1.71875
18	3000	3	16	0
18	6000	3	16	0
18	9000	3	16	0
19	3000	4	21	0
19	6000	4	21	0.0625
19	9000	4	21	0.109375
20	3000	33	107	0
20	6000	34	110	0.046875

TABLE 4: Continued.

Nr	Dim	PRP-WWP		
		NI	NFG	CPU
20	9000	35	113	0.125
21	3000	26	95	0.046875
21	6000	19	83	0.125
21	9000	19	83	0.15625
22	3000	13	67	0.0625
22	6000	13	67	0.0625
22	9000	13	67	0.140625
23	3000	1000	3017	8.953125
23	6000	1000	3023	17.34375
23	9000	290	1208	8.171875
24	3000	65	222	0.125
24	6000	83	276	0.34375
24	9000	92	303	0.578125
25	3000	33	98	0
25	6000	43	128	0.0625
25	9000	54	182	0.125
26	3000	50	184	0.125
26	6000	52	190	0.25
26	9000	53	193	0.328125
27	3000	30	157	0.046875
27	6000	35	204	0.109375
27	9000	46	268	0.1875
28	3000	24	108	0.0625
28	6000	24	108	0
28	9000	24	108	0.0625
29	3000	4	19	0
29	6000	4	19	0
29	9000	4	22	0
30	3000	1000	3113	0.75
30	6000	1000	3116	1.125
30	9000	1000	3141	1.984375
31	3000	20	81	0
31	6000	20	81	0.0625
31	9000	20	81	0.125
32	3000	26	88	0.0625
32	6000	44	142	0.1875
32	9000	61	193	0.359375
33	3000	3	9	0
33	6000	3	9	0
33	9000	2	6	0.046875
34	3000	3	15	0
34	6000	7	20	0
34	9000	7	20	0.03125
35	3000	14	41	0
35	6000	18	53	0
35	9000	21	62	0.03125
36	3000	19	88	0.3125
36	6000	27	126	0.75
36	9000	17	99	0.75
37	3000	1000	3015	1.3125
37	6000	1000	3206	2.53125
37	9000	1000	3098	3.609375
38	3000	7	40	0
38	6000	5	28	0
38	9000	5	28	0.0625
39	3000	38	113	0.0625
39	6000	40	119	0.0625
39	9000	41	122	0.109375
40	3000	1000	3134	22.75

TABLE 4: Continued.

Nr	Dim	PRP-WWP		
		NI	NFG	CPU
40	6000	1000	3282	36.375
40	9000	1000	3144	42.73438
41	3000	19	73	0
41	6000	19	73	0
41	9000	19	73	0.0625
42	3000	11	43	0.0625
42	6000	11	46	0.0625
42	9000	11	46	0.0625
43	3000	7	40	0.203125
43	6000	8	46	0.4375
43	9000	8	46	0.46875
44	3000	12	68	0.3125
44	6000	11	62	0.453125
44	9000	11	62	0.71875
45	3000	9	50	0.25
45	6000	9	50	0.46875
45	9000	9	50	0.546875
46	3000	260	835	6.0625
46	6000	529	1613	18.75
46	9000	611	1859	26.6875
47	3000	46	161	3.59375
47	6000	57	181	13.54688
47	9000	84	347	40.0625
48	3000	53	165	0.5625
48	6000	46	144	0.875
48	9000	40	126	0.953125
49	3000	1000	3113	0.75
49	6000	1000	3116	1.328125
49	9000	1000	3140	1.96875
50	3000	1000	3245	11.32813
50	6000	1000	3086	19.51563
50	9000	1000	3054	24.78125
51	3000	7	37	0.125
51	6000	7	40	0.25
51	9000	7	40	0.4375
52	3000	93	355	0.53125
52	6000	99	373	1.171875
52	9000	107	401	1.875
53	3000	1000	3022	0.875
53	6000	1000	3022	1.671875
53	9000	1000	3022	2.34375
54	3000	47	216	0.0625
54	6000	32	165	0.0625
54	9000	56	272	0.203125
55	3000	5	25	0.0625
55	6000	5	25	0.125
55	9000	5	25	0.1875
56	3000	1000	3012	0.984375
56	6000	1000	3012	1.671875
56	9000	1000	3012	2.5625
57	3000	9	50	0.1875
57	6000	9	50	0.390625

TABLE 4: Continued.

Nr	Dim	PRP-WWP		
		NI	NFG	CPU
57	9000	9	50	0.609375
58	3000	53	314	1.5625
58	6000	45	266	2.234375
58	9000	41	242	2.65625
59	3000	8	44	0.1875
59	6000	8	44	0.40625
59	9000	8	44	0.46875
60	3000	103	438	2.71875
60	6000	330	1048	12.03125
60	9000	265	916	12.73438
61	3000	9	50	0.25
61	6000	9	50	0.34375
61	9000	9	50	0.59375
62	3000	6	32	0.1875
62	6000	6	32	0.203125
62	9000	6	32	0.375
63	3000	6	32	0.140625
63	6000	6	32	0.3125
63	9000	5	26	0.265625
64	3000	8	44	0.203125
64	6000	8	44	0.4375
64	9000	8	44	0.453125
65	3000	12	46	0.25
65	6000	10	40	0.265625
65	9000	10	40	0.4375
66	3000	7	27	0.0625
66	6000	7	27	0.125
66	9000	7	27	0.171875
67	3000	24	125	0.5625
67	6000	25	131	1.125
67	9000	17	73	0.09375
68	3000	43	130	0.0625
68	6000	45	136	0.046875
68	9000	46	139	0.09375
69	3000	38	125	0
69	6000	39	128	0.046875
69	9000	40	131	0.125
70	3000	6	31	0.0625
70	6000	24	110	0.75
70	9000	6	31	0.234375
71	3000	1000	3009	0.875
71	6000	1000	3009	1.703125
71	9000	1000	3009	2.359375
72	3000	1000	3105	80.3125
72	6000	1000	4018	228.0625
72	9000	1000	3182	513.3125
73	3000	1000	3019	0.875
73	6000	1000	3019	1.5
73	9000	1000	3019	2.09375
74	3000	1000	3039	0.9375
74	6000	1000	3057	1.359375
74	9000	1000	3173	2.03125

TABLE 5: CPU times of the TTMLS algorithm and NLS algorithm in seconds.

30% noise	Lena	Barbara	Man	Baboon	Total
TTMLS algorithm	18.813	24.094	100.813	22.922	166.642
NLS algorithm	19.547	26.703	108.281	25.391	179.922
45% noise					
TTMLS algorithm	38.344	39.125	205.125	35.406	318.000
NLS algorithm	42.421	41.688	218.344	39.422	341.875
60% noise					
TTMLS algorithm	91.063	84.203	417.047	79.656	671.969
NLS algorithm	97.938	88.984	438.156	87.547	712.625

FIGURE 1: An overview of the performance of these four algorithms (NI).

FIGURE 2: An overview of the performance of these four algorithms (NFG).

FIGURE 3: An overview of the performance of these four algorithms (CPU).

FIGURE 4: From left to right: the images disturbed by 30% salt-and-pepper noise, the images restored by the TTMLS algorithm, and the images restored by the NLS algorithm.

FIGURE 5: From left to right: the images disturbed by 45% salt-and-pepper noise, the images restored by the TTMLS algorithm, and the images restored by the NLS algorithm.

FIGURE 6: From left to right: the images disturbed by 60% salt-and-pepper noise, the images restored by the TTMLS algorithm, and the images restored by the NLS algorithm.

FIGURE 7: From left to right: the original images, the images restored by the TTMLS algorithm, and the images restored by the LS algorithm.

TABLE 6: Peak signal-to-noise ratio (PSNR) of the TTMLS algorithm and LS algorithm.

Algorithm	Camera man	Fruits	Lena	Peppers	Total
TTMLS	31.929	33.053	31.996	33.075	130.053
LS	30.792	31.486	31.489	31.925	125.692

$g_k^T d_k$ may appear as a singular matrix, which results in the invalidation of the algorithm. But, when we calculate d_{k+1} , we only need the value of $g_k^T d_k$ without knowing the information of this square matrix, so in this experiment, we set $g_k^T d_k = \|g_k^T d_k\|_\infty$ in d_{k+1} . The data used in the experiment are as follows:

Parameters: all the algorithms run with $\gamma = 0.6$, $\gamma_1 = 0.3$, $\rho = 0.9$, and $\gamma = 0.7$.

Stop rule: if $\|g_k^T d_k\|_\infty < 10^{-3}$ or the number of iterations exceeds 500 is satisfied, we stop the process.

Symbol representation: PSNR: Peak Signal-to-Noise Ratio. It is an objective criterion for image evaluation.

Text problems: compressive sensing problems. The experiments chose Camera man (256×256), Fruits (256×256), Lena (256×256), and Baboon (256×256) as the test images.

From Figure 7 and Table 6, we can see that both algorithms are effective in compression sensing problems. Meanwhile, from the experimental data, we can see that the TTMLS algorithm has more advantages than the LS algorithm.

4. Conclusions

In this paper, based on the well-known LS method and combined with improved Armijo linear search, this paper presents a three-term conjugate gradient algorithm. Without any linear search, the search direction of the new three-term conjugate algorithm is proved to have two good properties: sufficient descent and trust region properties. Also, the global convergence of the algorithm is established. The numerical results indicate that the new algorithm is effective. The good performance of the algorithm in image restoration problems and compressive sensing problems also proves that the algorithm is competitive.

Data Availability

Data used in this study can be obtained from the corresponding author on reasonable request.

Conflicts of Interest

There are no potential conflicts of interest.

Acknowledgments

The authors want to thank the support of the funds. This work was supported by the High Level Innovation Teams and Excellent Scholars Program in Guangxi Institutions of Higher Education (Grant no. [2019] 32), the National Natural Science Foundation of China (Grant no. 11661009),

the Guangxi Natural Science Foundation (No. 2020GXNSFAA159069), and the Guangxi Natural Science Key Foundation (No. 2017GXNSFDA198046).

References

- [1] Y.-H. Dai, "New properties of a nonlinear conjugate gradient method," *Numerische Mathematik*, vol. 89, no. 1, pp. 83–98, 2001.
- [2] L. Grippo and S. Lucidi, "A globally convergent version of the Polak-Ribière conjugate gradient method," *Mathematical Programming*, vol. 78, no. 3, pp. 375–391, 1997.
- [3] J. Nocedal and S. Wright, "Numerical optimization," in *Springer Series in Operations Research*, Springer, Berlin, Germany, 2nd edition, 2006.
- [4] Z. Shi, "Restricted PR conjugate gradient method and its global convergence," *Advances in Mathematics*, vol. 31, pp. 47–55, 2002.
- [5] Z. Wei, S. Yao, and L. Liu, "The convergence properties of some new conjugate gradient methods," *Applied Mathematics and Computation*, vol. 183, no. 2, pp. 1341–1350, 2006.
- [6] G. Yuan, X. Wang, and Z. Sheng, "Family weak conjugate gradient algorithms and their convergence analysis for nonconvex functions," *Numerical Algorithms*, vol. 84, no. 3, pp. 935–956, 2020.
- [7] W. Zhou, "A short note on the global convergence of the unmodified PRP method," *Optimization Letters*, vol. 7, no. 6, pp. 1367–1372, 2013.
- [8] D. Li and M. Fukushima, "On the global convergence of the BFGS method for nonconvex unconstrained optimization problems," *SIAM Journal on Optimization*, vol. 11, pp. 1054–1064, 1999.
- [9] N. Andrei, "An unconstrained optimization test functions collection," *Advanced Modeling Optimization*, vol. 10, pp. 147–161, 2008.
- [10] J. C. Gilbert and J. Nocedal, "Global convergence properties of conjugate gradient methods for optimization," *Siam Journal on Optimization*, vol. 2, no. 1, pp. 21–42, 1992.
- [11] X. Li and Q. Ruan, "A modified PRP conjugate gradient algorithm with trust region for optimization problems," *Numerical Functional Analysis and Optimization*, vol. 32, no. 5, pp. 496–506, 2011.
- [12] Z. Wei, G. Li, and Q. Li, "Global convergence of the Polak-Ribière-Polyak conjugate gradient method with an Armijo-type inexact line search for nonconvex unconstrained optimization problems," *Mathematics of Computation*, 2008.
- [13] G. Yu, L. Guan, and Z. Wei, "Globally convergent Polak-Ribière-Polyak conjugate gradient methods under a modified Wolfe line search," *Applied Mathematics and Computation*, vol. 215, no. 8, pp. 3082–3090, 2009.
- [14] L. Zhang, W. Zhou, and D. Li, "Global convergence of a modified Fletcher-Reeves conjugate gradient method with Armijo-type line search," *Numerische Mathematik*, vol. 104, no. 4, pp. 561–572, 2006.
- [15] F. Wen and X. Yang, "Skewness of return distribution and coefficient of risk premium," *Journal of Systems Science and Complexity*, vol. 22, no. 3, pp. 360–371, 2009.

- [16] G. Yuan, J. Lu, and Z. Wang, "The PRP conjugate gradient algorithm with a modified WWP line search and its application in the image restoration problems," *Applied Numerical Mathematics*, vol. 152, pp. 1–11, 2020.
- [17] G. Yuan, Z. Wei, and Y. Yang, "The global convergence of the Polak-Ribiere-Polyak conjugate gradient algorithm under inexact line search for nonconvex functions," *Journal of Computational and Applied Mathematics*, vol. 362, pp. 262–275, 2019.
- [18] W. Zhou and F. Wang, "A PRP-based residual method for large-scale monotone nonlinear equations," *Applied Mathematics and Computation*, vol. 261, pp. 1–7, 2015.
- [19] 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, pp. 409–436, 1952.
- [20] R. Fletcher and C. Reeves, "Function minimization by conjugate gradients," *The Computer Journal*, vol. 7, no. 2, pp. 149–154, 1964.
- [21] E. Polak and G. Ribière, "Note sur la convergence de méthodes de directions conjuguées," *Revue Française d'informatique et de Recherche Opérationnelle. Série Rouge*, vol. 3, no. 16, pp. 35–43, 1969.
- [22] B. T. Polyak, "The conjugate gradient method in extremal problems," *USSR Computational Mathematics and Mathematical Physics*, vol. 9, no. 4, pp. 94–112, 1969.
- [23] Y. Liu and C. Storey, "Efficient generalized conjugate gradient algorithms, part 1: theory," *Journal of Optimization Theory and Applications*, vol. 69, no. 1, pp. 129–137, 1991.
- [24] R. Fletcher, *Practical Method of Optimization Vol. I: Unconstrained Optimization*, John Wiley & Sons, New York, NY, USA, 1987.
- [25] Y. H. Dai and Y. Yuan, "A nonlinear conjugate gradient method with a strong global convergence property," *SIAM Journal on Optimization*, vol. 10, no. 1, pp. 177–182, 1999.
- [26] Z. Li, "A new Liu-Storey type nonlinear conjugate gradient method for unconstrained optimization problems," *Journal of Computational and Applied Mathematics*, vol. 225, no. 1, pp. 146–157, 2009.
- [27] L. Zhang, W. Zhou, and D.-H. Li, "A descent modified Polak-Ribière-Polyak conjugate gradient method and its global convergence," *IMA Journal of Numerical Analysis*, vol. 26, no. 4, pp. 629–640, 2006.
- [28] G. Yuan, T. Li, and W. Hu, "A conjugate gradient algorithm for large-scale nonlinear equations and image restoration problems," *Applied Numerical Mathematics*, vol. 147, pp. 129–141, 2020.
- [29] G. Yuan and Z. Wei, "Convergence analysis of a modified BFGS method on convex minimizations," *Computational Optimization and Applications*, vol. 47, no. 2, pp. 237–255, 2010.
- [30] Z. Dai and F. Wen, "Global convergence of a modified hestenes-stiefel nonlinear conjugate gradient method with armijo line search," *Numerical Algorithms*, vol. 59, no. 1, pp. 79–93, 2012.
- [31] M. Li and A. Qu, "Some sufficient descent conjugate gradient methods and their global convergence," *Computational and Applied Mathematics*, vol. 33, pp. 333–347, 2004.
- [32] L. Zhang, W. Zhou, and D. Li, "Global convergence of the dy conjugate gradient method with armijo line search for unconstrained optimization problems," *Optimization Methods and Software*, vol. 22, no. 3, pp. 511–517, 2007.
- [33] G. Yuan, Z. Wei, and X. Lu, "Global convergence of BFGS and PRP methods under a modified weak Wolfe-Powell line search," *Applied Mathematical Modelling*, vol. 47, pp. 811–825, 2017.
- [34] Y. Yuan and W. Sun, *Theory and Methods of Optimization*, Science Press of China, Beijing, China, 1999.
- [35] E. Dolan and J. Moré, "Benchmarking optimization software with performance profiles," *Mathematical Programming*, vol. 91, no. 2, pp. 201–213, 2001.
- [36] Q. Dai and W. Sha, "The physics of compressive sensing and the gradient-based recovery algorithms," *Mathematics*, 2009.