KYBERNETIKA — VOLUME *52* (2016), NUMBER 5, PAGES 791–823

IMPULSE NOISE REMOVAL BASED ON NEW HYBRID CONJUGATE GRADIENT APPROACH

Morteza Kimiaei and Majid Rostami

Image denoising is a fundamental problem in image processing operations. In this paper, we present a two-phase scheme for the impulse noise removal. In the first phase, noise candidates are identified by the adaptive median filter (AMF) for salt-and-pepper noise. In the second phase, a new hybrid conjugate gradient method is used to minimize an edge-preserving regularization functional. The second phase of our algorithm inherits advantages of both Dai–Yuan (DY) and Hager–Zhang (HZ) conjugate gradient methods to produce the new direction. The descent property of new direction in each iteration and the global convergence results are established under some standard assumptions. Furthermore, we investigate some conjugate gradient algorithms and the complexity analysis of theirs. Numerical experiments are given to illustrate the efficiency of the new hybrid conjugate gradient (HCGN) method for impulse noise removal.

Keywords: image processing, impulse noise, unconstrained optimization, conjugate gradient method, Wolfe conditions, complexity analysis

Classification: 90C30, 90C25, 90C90, 68U10, 03D15

1. INTRODUCTION

Images are often corrupted by impulse noise due to noisy sensors or communication channels. There are two main models to represent impulse noise. One is the ***salt-and-pepper*** noise in which the noisy pixels can take only the maximal and minimal pixel values in dynamic range $[s_{\min}, s_{\max}]$ of original image. Other impulse noise is the ***random-valued*** noise in which the noisy pixels can take any random values between the maximal and minimal pixel values of dynamic range. Removing above both noise is one of the most important problems in image processing. For this purpose, there are two popular types of methods for removing impulse noise: (1) The ***median filter*** and its variants [10] which can detect the noisy pixels but restore them poorly when the noise ratio is high. The gray levels of uncorrupted pixels are unchanged. The recovered image may loose its details and be distorted. (2) The ***variational approach*** is capable for retaining the details and the edges well but the gray level of every pixel is changed including uncorrupted ones, cf. [21].

DOI: 10.14736/kyb-2016-5-0791

1.1. Review of Two-Phase Methods

Recently, a two-phase procedure has been proposed in [4] to remove impulse noise. The first phase of this approach is the detection of the noisy pixels by using the ***adaptive median filter*** (AMF) [19] for salt-and-pepper noise while for random-valued noise, it is prepared by using the ***adaptive center–weighted median filter*** (ACWMF) [10], which is first ameliorated by employing the variable window technique to improve its detection ability in highly corrupted images. In this paper, we only use the salt-and-pepper noise.

Let x denote the original image with M-by-N pixels and

$$\mathcal{A} := \{(i,j) \mid i = 1,2,3,\cdots,M \ \text{ and } \ j = 1,2,3,\cdots,N\},$$

be the index set of x. In addition, the observed noisy image of x contaminated by the salt-and-pepper noise is considered by y, the set of the four closest neighbours of the pixel at position $(i,j) \in \mathcal{A}$ is showed by $\mathcal{V}_{i,j}$ and the gray level of x at pixel location (i,j) is denoted by $x_{i,j}$. Hence, the gray level of y at pixel location (i,j) can be considered as

$$y_{i,j} := \begin{cases} s_{\min} & \text{with probability } p, \\ s_{\max} & \text{with probability } q, \\ x_{i,j} & \text{with probability } 1-p-q. \end{cases}$$

Let us denote the image obtained by applying AMF to the noisy image y by $\widetilde{y}$. Based on this fact that noisy pixels take the value either $s_{\min}$ or $s_{\max}$, we define the ***noise candidate set*** as

$$\mathcal{N} := \{(i,j) \in \mathcal{A} \mid \widetilde{y}_{i,j} \neq y_{i,j} \text{ and } \ y_{i,j} = s_{\min} \text{ or } \ s_{\max}\},$$

which denotes the set of indices of the noisy pixels detected in the first phase and take $\mathcal{N}^c := \{(i,j) \in \mathcal{A} \mid (i,j) \notin \mathcal{N}\}$ as complementary of $\mathcal{N}$. The success of this two-phase approach relies on the accurate detection of $\mathcal{N}^c$ by AMF to detect salt-and-pepper impulse noise in the first phase.

The goal of the second phase is the recovering of the noisy pixels obtained by the first phase. For each pixel location (i,j), which belongs to $\mathcal{N}$, it is detected as uncorrupted and hence we naturally keep its original value, i. e., $u^*_{i,j} := y_{i,j}$. But for each pixel location (i,j), which does not belong to $\mathcal{N}$, we must restore $y_{i,j}$. Moreover, if $(m,n) \in \mathcal{V}_{i,j} \setminus \mathcal{N}$, then we set $u^*_{m,n} := y_{m,n}$ and if $(m,n) \in \mathcal{V}_{i,j} \cap \mathcal{N}$, then we restore $y_{m,n}$. Therefore, similar to [2, 4, 5, 6, 7, 9], the second phase is the restoring of all $y_{i,j}$ by minimization the following ***regular image inpainting problem***:

$$\begin{aligned} &\min \ \textstyle\sum_{(i,j)\in\mathcal{N}} \Big\{ \underbrace{\mid u_{i,j} - y_{i,j} \mid}_{\text{nonsmooth term}} + \tfrac{\beta}{2}\Psi(u_{i,j}) \Big\} \\ &u \in \mathbb{R}^{|\mathcal{N}|}, \end{aligned} \tag{1}$$

where $|\mathcal{N}|$ is the cardinal of $\mathcal{N}$ and

$$\Psi(u_{i,j}) := 2 \sum_{(m,n)\in\mathcal{V}_{i,j}\setminus\mathcal{N}} \varphi_\alpha(u_{i,j} - y_{m,n}) + \sum_{(m,n)\in\mathcal{V}_{i,j}\cap\mathcal{N}} \varphi_\alpha(u_{i,j} - u_{m,n}),$$

β is the regularization parameter, $u := [u_{i,j}]_{(i,j)\in\mathcal{N}}$ is a column vector of length $|\mathcal{N}|$ ordered lexicographically and φ_α is an edge-preserving function, which must be: (a) twice continuously differentiable, (b) $\varphi_\alpha^{''} > 0$ and (c) even. Example of such edge-preserving function is $\varphi_\alpha(t) := \sqrt{t^2+\alpha}$ where $\alpha > 0$ is a parameter. For more examples of these functions, cf. [21]. From the above properties, we can conclude that $\varphi_\alpha(t)$ is strictly increasing with $|t|$ and coercive, i. e., $\varphi_\alpha(t) \to \infty$ as $t \to \infty$.

However, because of the $|u_{i,j} - y_{i,j}|$ term, the functional of problem (1) is nonsmooth. It is generally believed that this nonsmooth term can remove from (1) because on the one hand it keeps the minimizer u near the original image y so that the pixels, uncorrupted in the original image, are not altered. However, in the two-phase method, the functional of problem (1) is cleaning only the noise pixels while the uncorrupted pixels are unchanged. Hence, the nonsmooth term is not required. On the other hand, removing the nonsmooth term will convert the functional of problem (1) to a smooth functional which can be efficiently obtained the optimizer, see [4]. In fact, the nonsmooth data-fitting term is omitted and only noisy pixels are restored in the minimization. Then, the following smooth functional is obtained

$$F_\alpha(u) := \sum_{(i,j)\in\mathcal{N}} \Psi(u_{i,j}). \tag{2}$$

Due to φ_α is an even function, similar to [4], we can get the gradient of $F_\alpha(u)$ as

$$(g(u))_{(i,j)\in\mathcal{N}} := (\nabla F_\alpha(u))_{(i,j)\in\mathcal{N}} := 2\Big(\sum_{(m,n)\in\mathcal{V}_{i,j}\setminus\mathcal{N}} \varphi'_\alpha(u_{i,j}-y_{m,n}) + \sum_{(m,n)\in\mathcal{V}_{i,j}\cap\mathcal{N}} \varphi'_\alpha(u_{i,j}-u_{m,n})\Big). \tag{3}$$

Notation: The symbol $\|\cdot\|$ denotes the Euclidean vector norm. Let us denote $n_1 := |\mathcal{N}|$, $n_2 := |\mathcal{V}_{i,j} \setminus \mathcal{N}|$ and $n_3 := |\mathcal{V}_{i,j} \cap \mathcal{N}|$. $P_{[a,b]}(x)$ sets a number $x \in \mathbb{R}$ onto $[a,b]$, which is defined by

$$P_{[a,b]}(c) := \begin{cases} c & \text{if} \quad a \le c \le b, \\ a & \text{if} \quad c < a, \\ b & \text{if} \quad c > b. \end{cases}$$

1.2. Background of conjugate gradient methods

Conjugate gradient (CG) methods are suitable to solve unconstrained optimization problem

$$\begin{array}{ll} \min & F_\alpha(u) \\ \text{s.t.} & u \in \mathbb{R}^{n_1}, \end{array} \tag{4}$$

where $F_\alpha : \mathbb{R}^{n_1} \longrightarrow \mathbb{R}$ is the smooth function of (2), because these algorithms have low computational costs and have strong local and global convergence properties. The key feature of these algorithms is that they require no matrix storage. In 1952, Hestenes and Stiefel [18] (the ***CGHS method***) have presented a technique to solve the linear system that is symmetric and positive-definite. The first nonlinear CG method is proposed by Fletcher and Reeves [14] (the ***CGFR method***). The ***CGPR method*** is introduced by Polak and Ribière [25] which has a high numerical performance but does not have strong convergence. The global convergence of this method has been established in [25, 26]. By using an example, Powell showed that the global convergence of proposed method is

uncertain when the function is not strongly convex, see [27]. To overcome this drawback, Powell [27] modified the parameter of the CGPR method while Gilbert and Nocedal [15] proved the convergence of the modified CGPR method. Using the Wolfe conditions, the ***CGDY method*** [12] generated descent directions while its global convergence under the Lipschitz assumption holds. Hager and Zhang [16] (the ***CGHZ method***) have taken advantages of the CGHS method to produce an efficient method, independent of the line search, guarantees the descent property. The mentioned methods produce an iterative sequence $\{u_k\}$ in the form $u_{k+1} := u_k + \alpha_k d_k$ where α_k is a step-size and d_k is a direction defined by

$$d_k := \begin{cases} -g_k & \text{if } k = 0, \\ -g_k + \beta_k d_{k-1} & \text{if } k \geq 1, \end{cases} \tag{5}$$

where β_k is a scalar and $g_k := \nabla F_\alpha(u_k)$. These methods have been presented several famous formula for β_k as

$$\beta_k^{HS} := \frac{g_k^T y_{k-1}}{d_{k-1}^T y_{k-1}}, \tag{6}$$

$$\beta_k^{FR} := \frac{\|g_k\|^2}{\|g_{k-1}\|^2}, \tag{7}$$

$$\beta_k^{PR} := \frac{g_k^T y_{k-1}}{\|g_{k-1}\|^2}, \tag{8}$$

$$\beta_k^{DY} := \frac{\|g_k\|^2}{d_{k-1}^T y_{k-1}}, \tag{9}$$

$$\beta_k^{HZ} := \beta_k^{HS} - 2\|y_{k-1}\|^2 \frac{d_{k-1}^T g_k}{(d_{k-1}^T y_{k-1})^2}, \tag{10}$$

where $y_{k-1} := g_k - g_{k-1}$. The global convergence of the CG methods have been studied in [14, 16, 18, 24, 25, 26, 27]. These methods usually require that the step-size α_k should be obtained by exact or inexact line search technique in order to establish the convergence results. The ***Strong Wolfe conditions*** are the main inexact line search conditions which find a step-size α_k satisfying the following conditions

$$\textit{Armijo condition: } F_\alpha(u_k + \alpha_k d_k) - F_\alpha(u_k) \leq \alpha_k \eta_1 g_k^T d_k, \tag{11}$$

$$\textit{Curvature condition: } |g(u_k + \alpha_k d_k)^T d_k| \leq -\eta_2 g_k^T d_k, \tag{12}$$

where $0 < \eta_1 < \eta_2 < 1$. To simplify notation, we set $\psi(\alpha_k) := F_\alpha(u_k + \alpha_k d_k)$, which gives $\psi(0) := F_\alpha(u_k)$, $\psi'(0) := g_k^T d_k$ and $\psi'(\alpha_k) := g(u_k + \alpha_k d_k)^T d_k$. Therefore, by rewriting (11) and (12) we have

$$\psi(\alpha_k) - \psi(0) \leq \alpha_k \eta_1 \psi'(0), \tag{13}$$

$$|\psi'(\alpha_k)| \leq -\eta_2 \psi'(0). \tag{14}$$

1.3. CG algorithms and their complexity analysis

In order to present the generalized conjugate gradient algorithm and complexity analysis of it, we first introduce zoom and Line search algorithms, cf. [23], and then we

compute complexity analysis of them.

Algorithm 1: ZAL (zoom algorithm)

Input: α_{lo}, α_{hi}, $\psi(\alpha_{\text{hi}})$, $\psi(\alpha_{\text{lo}})$, $\psi'(\alpha_{\text{lo}})$;
Output: α^*, $j_{\max}$;
begin
  $j := 0$;
  while *not converged* **do**
    $c_1 := \alpha_{\text{hi}} - \alpha_{\text{lo}}$;
    $c_2 := (\psi(\alpha_{\text{hi}}) - \psi(\alpha_{\text{lo}}) - c_1\psi'(\alpha_{\text{lo}}))/c_1^2$;
    if $c_2 \leq 0$ **then**
      $\alpha_j := \alpha_{\text{lo}} + 0.5c_1$; (bisection)
    else
      $\alpha_j := (\alpha_{\text{lo}} - 0.5\psi'(\alpha_{\text{lo}}))/c_2$; (quadratic)
    end
    compute $\psi(\alpha_j)$;
    if $\psi(\alpha_j) > \psi(0) + \eta_1\alpha_j\psi'(0)$ *or* $\psi(\alpha_j) \geq \psi(\alpha_{lo})$ **then**
      $\alpha_{\text{hi}} := \alpha_j$;
    end
    compute $\psi'(\alpha_j)$;
    if $|\psi'(\alpha_j)| \leq -\eta_2\psi'(0)$ **then**
      $\alpha^* := \alpha_j$;
      stop;
    end
    $c_3 := \psi'(\alpha_j)(\alpha_{\text{hi}} - \alpha_{\text{lo}})$;
    if $c_3 \geq 0$ **then**
      $\alpha_{\text{hi}} := \alpha_{\text{lo}}$;
      $\alpha_{\text{lo}} := \alpha_j$;
    end
    $j := j + 1$;
  end
  $j_{\max} := j$;
end

Let us denote the ***number of multiplications*** by N_{mult}, the ***number of division*** by N_{div}, the ***number of summations*** by N_{sum} and the ***number of subtractions*** by N_{sub}. We suppose that each type of operation takes the same CPU time. Note that we overlook the root operation in computing φ_α and φ'_α. Now, we compute the ***total number of arithmetic operations*** (TNAO) for Algorithm 1 by the following procedure.

Procedure 1: Calculation of TNAO for an iteration of Algorithm 1

Step 1 (**Line 4**): $N_{\rm sub}(c_1)=1 \implies {\rm TNAO}(c_1)=1$.
Step 2 (**Line 5**): $N_{\rm sub}(c_2)=2$, $N_{\rm mult}(c_2)=2$ and $N_{\rm div}(c_2)=1 \implies {\rm TNAO}(c_2)=5$.
Step 3 (**Line 7**): $N_{\rm mult}(\alpha_j)=1$ and $N_{\rm sum}(\alpha_j)=1 \implies {\rm TNAO}(\alpha_j)=2$.
Step 4 (**Line 9**): $N_{\rm sub}(\alpha_j)=1$, $N_{\rm mult}(\alpha_j)=1$ and $N_{\rm div}(\alpha_j)=1 \Longrightarrow {\rm TNAO}(\alpha_j)=3$.
Step 5 (**Lines 11 and 15**): Let $u_k^+ := u_k+\alpha_j d_k$. Then
$N_{\rm sum}(u_k^+)=n_1$ and $N_{\rm mult}(u_k^+)=n_1 \implies {\rm TNAO}(u_k^+)=2n_1$.
Step 6 (**Line 11**): To compute $\psi(\alpha_j)$, let $z_1 := (u_k^+)_{i,j}-y_{m,n}$ and $z_2 := (u_k^+)_{i,j}-u_{m,n}$ then we compute ${\rm TNAO}(\Psi_{i,j})$ as follows:

(a) $N_{\rm sub}(z_1)=n_2 \implies {\rm TNAO}(z_1)=n_2$;

(b) $N_{\rm sub}(z_2)=n_3 \implies {\rm TNAO}(z_2)=n_3$;

(c) $N_{\rm sum}(\varphi_\alpha)=N_{\rm mult}(\varphi_\alpha)=n_2+n_3 \implies {\rm TNAO}(\varphi_\alpha)=2(n_2+n_3)$;

(d) $N_{\rm sum}(2\sum_{(m,n)\in\mathcal{V}_{i,j}\setminus\mathcal{N}}\varphi_\alpha)=n_2-1$ and $N_{\rm mult}(2\sum_{(m,n)\in\mathcal{V}_{i,j}\setminus\mathcal{N}}\varphi_\alpha)=1$
$\implies {\rm TNAO}(2\sum_{(m,n)\in\mathcal{V}_{i,j}\setminus\mathcal{N}}\varphi_\alpha)=n_2$;

(e) $N_{\rm sum}(\sum_{(m,n)\in\mathcal{V}_{i,j}\cap\mathcal{N}}\varphi_\alpha)=n_3-1 \implies {\rm TNAO}(\sum_{(m,n)\in\mathcal{V}_{i,j}\cap\mathcal{N}}\varphi_\alpha)=n_3-1$;

(f) $N_{\rm sum}(+)=1$
$\implies {\rm TNAO}(+)=1$.

Hence

$$ {\rm TNAO}(\Psi_{i,j})=n_2+n_3+2(n_2+n_3)+n_2+n_3-1+1=4(n_2+n_3), $$

and therefore

$$ \begin{aligned} {\rm TNAO}(\psi(\alpha_j)) &:= {\rm TNAO}(u_k^+)+n_1({\rm TNAO}(\Psi_{i,j})) \\ &= 2n_1+n_1(4(n_2+n_3)) \\ &= 2n_1+4n_1(n_2+n_3). \end{aligned} $$

Step 7 (**Line 12**): $N_{\rm sum}(\textbf{Armijo})=1$ and $N_{\rm mult}(\textbf{Armijo})=2 \Longrightarrow {\rm TNAO}(\textbf{Armijo})=3$.
Step 8 (**Line 15**): To compute $\psi'(\alpha_j)$,

(a) $N_{\rm sum}(\varphi'_\alpha)=N_{\rm mult}(\varphi'_\alpha)=N_{\rm div}(\varphi'_\alpha)=n_2+n_3 \implies {\rm TNAO}(\varphi'_\alpha)=3(n_2+n_3)$;

(b) $N_{\rm sum}(\sum_{(m,n)\in\mathcal{V}_{i,j}\setminus\mathcal{N}}\varphi'_\alpha)=n_2-1 \implies {\rm TNAO}(\sum_{(m,n)\in\mathcal{V}_{i,j}\setminus\mathcal{N}}\varphi_\alpha)=n_2-1$;

(c) $N_{\rm sum}(\sum_{(m,n)\in\mathcal{V}_{i,j}\cap\mathcal{N}}\varphi'_\alpha)=n_3-1 \implies {\rm TNAO}(\sum_{(m,n)\in\mathcal{V}_{i,j}\cap\mathcal{N}}\varphi'_\alpha)=n_3-1$;

(d) Let $z_3 := 2(\sum_{(m,n)\in\mathcal{V}_{i,j}\setminus\mathcal{N}}\varphi'_\alpha+\sum_{(m,n)\in\mathcal{V}_{i,j}\cap\mathcal{N}}\varphi'_\alpha)$ then we have
$N_{\rm mult}(z_3)=1$ and $N_{\rm sum}(z_3)=1 \implies {\rm TNAO}(z_3)=2$;

(e) $N_{\rm sum}(g(u_k^+)^Td_k)=n_1-1$ and $N_{\rm mult}(g(u_k^+)^Td_k)=n_1 \implies {\rm TNAO}(g(u_k^+)^Td_k)=2n_1-1$.

Hence

$$\text{TNAO}(\psi'(\alpha_j)) = 3(n_2+n_3)+n_2-1+n_3-1+2+2n_1-1 = 4(n_2+n_3)+2n_1-1.$$

Step 9 (**Line 16**): $N_{\text{mult}}(\textbf{curvature}) = 1 \implies \text{TNAO}(\textbf{curvature}) = 1.$
Step 10 (**Line 20**): $N_{\text{mult}}(c_3) = 1$ and $N_{\text{sub}}(c_3) = 1 \implies \text{TNAO}(c_3) = 2.$

Let us define $j_{\max} := j_1 + j_2$, in which j_1 is the number of iterations of Algorithm 1 with $c_2 \le 0$ and j_2 is the number of iterations of Algorithm 1 with $c_2 > 0$. Now, by procedure 1, we compute TNAO of Algorithm 1, for all iterations, as

$$\begin{aligned}
&\text{TNAO(ZAL)}\\
:= \;& \underbrace{(j_1-1)\Big(13+4(n_2+n_3)(n_1+1)+6n_1\Big)}_{\text{TNAO for } j=1,\cdots,j_{\max}-1 \text{ and } c_2\le 0} + \underbrace{11+4(n_2+n_3)(n_1+1)+6n_1}_{\text{TNAO for } j=j_{\max} \text{ and } c_2\le 0}\\
+ \;& \underbrace{(j_2-1)\Big(14+4(n_2+n_3)(n_1+1)+6n_1\Big)}_{\text{TNAO for } j=1,\cdots,j_{\max}-1 \text{ and } c_2> 0} + \underbrace{12+4(n_2+n_3)(n_1+1)+6n_1}_{\text{TNAO for } j=j_{\max} \text{ and } c_2> 0}\\
= \;& j_{\max}\Big(4(n_2+n_3)(n_1+1)+6n_1\Big)+13j_1+14j_2-4 = O(n_1\max\{n_2,n_3\}).
\end{aligned}$$

Here, we describe the line search algorithm that guarantees finding a step-size to satisfy the strong Wolfe conditions (13) and (14).
Algorithm 2 starts with a trial estimate α_1 and keeps increasing it (Line 18) until it finds either an acceptable step-size (Line 12) or an interval (Line 6 and Line 15) that brackets the desired step-sizes by calling Algorithm 1, which successively decreases the size of the interval until an acceptable step-size is identified.

TNAO for Algorithm 2 is computed by the following procedure:

Procedure 2: Calculation of TNAO for an iteration of Algorithm 2

Step 1 (**Line 4**): $\text{TNAO}(\psi(\alpha_i)) = 2n_1 + 4n_1(n_2+n_3)$.
Step 2 (**Line 5**): $\text{TNAO}(\textbf{Armijo}) = 3$.
Step 3 (**Line 6**): $\text{TNAO}(\alpha^*) := \text{TNAO(ZAL)} = O(n_1\max\{n_2,n_3\})$.
Step 4 (**Line 9**): $\text{TNAO}(\psi'(\alpha_i)) = 4(n_2+n_3)+2n_1-1$.
Step 5 (**Line 10**): $N_{\text{mult}}(\textbf{curvature}) = 1 \implies \text{TNAO}(\textbf{curvature}) = 1$.
Step 6 (**Line 15**): $\text{TNAO}(\alpha^*) := \text{TNAO(ZAL)} = O(n_1\max\{n_2,n_3\})$.
Step 7 (**Line 18**): $\text{TNAO}(\alpha_{i+1}) = 1$.

Note that, in Line 18 of Algorithm 2, we choose $\alpha_{i+1} = \min(2\alpha_i, \alpha_{\max})$. First, we compute the TNAO of Algorithm 2 in each iteration without stop condition as follows

$$\begin{aligned}
\text{TNAO}^1\text{(LSAL)} \;:=\; & \text{TNAO}(\psi(\alpha_i)) + \text{TNAO}(\textbf{Armijo}) + \text{TNAO}(\psi'(\alpha_i))\\
& + \text{TNAO}(\textbf{curvature}) + \text{TNAO}(\alpha_{i+1})\\
=\; & 2n_1 + 4n_1(n_2+n_3) + 3 + 4(n_2+n_3) + 2n_1 - 1 + 1 + 1\\
=\; & 4(n_2+n_3)(n_1+1) + 4n_1 + 4 = O(n_1\max\{n_2,n_3\}).
\end{aligned}$$

Algorithm 2: LSAL (Line search algorithm)

Input: An initial point $\alpha_0 = 0$, α_1, $\psi(0)$, $\psi'(0)$;
Output: α^* and $i_{\max}$;

```
begin
    i := 1;
    while not converged do
        compute ψ(α_i);
        if ψ(α_i) > ψ(0) + η_1 α_i ψ'(0) or ψ(α_i) ≥ ψ(α_{i-1}) and i > 1 then
            α* := ZAL(α_i, α_{i-1}, ψ(α_i), ψ(α_{i-1}), ψ'(α_{i-1}));
            stop 1;
        end
        compute ψ'(α_i);
        if |ψ'(α_i)| ≤ −η_2 ψ'(0) then
            α* := α_i;
            stop 2;
        end
        if ψ'(α_i) ≥ 0 then
            α* := ZAL(α_i, α_{i-1}, ψ(α_i), ψ(α_{i-1}), ψ'(α_{i-1}));
            stop 3;
        end
        α_{i+1} ∈ (α_i, α_max);
        i := i + 1;
    end
    i_max := i;
end
```

Also, the TNAO of Algorithm 2 with first stop condition, second stop condition and third stop condition are computed, respectively, by

$$\begin{aligned}\mathrm{TNAO}^{s_1}(\mathrm{LSAL}) :&= \mathrm{TNAO}(\psi(\alpha_i)) + \mathrm{TNAO}(\textbf{Armijo}) + \mathrm{TNAO}(\mathrm{ZAL})\\ &= 2n_1 + 4n_1(n_2+n_3) + 3 + O(n_1\max\{n_2,n_3\}) = O(n_1\max\{n_2,n_3\}),\end{aligned}$$

$$\begin{aligned}\mathrm{TNAO}^{s_2}(\mathrm{LSAL}) :&= \mathrm{TNAO}(\psi(\alpha_i)) + \mathrm{TNAO}(\textbf{Armijo}) + \mathrm{TNAO}(\psi'(\alpha_i)) + \mathrm{TNAO}(\textbf{curvature})\\ &= 2n_1 + 4n_1(n_2+n_3) + 3 + 4(n_2+n_3) + 2n_1 - 1 + 1 = O(n_1\max\{n_2,n_3\}),\end{aligned}$$

$$\begin{aligned}\mathrm{TNAO}^{s_3}(\mathrm{LSAL}) :&= \mathrm{TNAO}^{s_2}(\mathrm{LSAL}) + \mathrm{TNAO}(\mathrm{ZAL})\\ &= O(n_1\max\{n_2,n_3\}) + O(n_1\max\{n_2,n_3\}) = O(n_1\max\{n_2,n_3\}).\end{aligned}$$

Now, we compute TNAO of Algorithm 2, for all iterations, as

$$\mathrm{TNAO(LSAL)} := \begin{cases} (i_{\max}-1)(\mathrm{TNAO}^{1}(\mathrm{LSAL})) + \mathrm{TNAO}^{s_1}(\mathrm{LSAL}) & \text{if Stop=Stop 1;} \\ (i_{\max}-1)(\mathrm{TNAO}^{1}(\mathrm{LSAL})) + \mathrm{TNAO}^{s_2}(\mathrm{LSAL}) & \text{if Stop=Stop 2;} \\ (i_{\max}-1)(\mathrm{TNAO}^{1}(\mathrm{LSAL})) + \mathrm{TNAO}^{s_3}(\mathrm{LSAL}) & \text{if Stop=Stop 3.} \end{cases}$$

Therefore

$$\mathrm{TNAO(LSAL)} := O(n_1 \max\{n_2, n_3\}).$$

Finally, the generalized conjugate gradient algorithm is presented as follows:

Algorithm 3: GCG (generalized Conjugate Gradient)

Input: An initial point $u_0 \in \mathbb{R}^{n_1}$, $\alpha_{\max} > 0$, $\alpha_1 \in (0, \alpha_{\max})$, $0 < \eta_1 < \eta_2 < 1$, $0 < \lambda_{\min} < \lambda_{\max} < \infty$;
Output: u^*, F_α^*, $k_{\max}$;

```
1  begin
2      k := 0;
3      compute F_α(u_0);
4      compute g_0;
5      d_0 := −g_0;
6      while not converged do
7          [α_k, u_{k+1}, F_α(u_{k+1}), g_{k+1}] = LSAL(F_α(u_k), g_k, d_k, α_max, α_1);
8          compute β_{k+1} by one of the formulas (6)-(10);
9          obtain d_{k+1} by (5);
10         replace k + 1 by k;
11     end
12     u* := u_k;  F*_α := F_α(u_k); k_max := k;
13 end
```

By the following procedure, TNAO for Algorithm 3 computes.

Procedure 3: Calculation of TNAO for an iteration of Algorithm 3

Step 1 (**Line 3**): $\mathrm{TNAO}(F_\alpha(u_0)) = 2n_1 + 4n_1(n_2 + n_3)$.
Step 2 (**Line 4**): $\mathrm{TNAO}(g_0) = 4(n_2 + n_3) + 2n_1 - 1$.
Step 3 (**Line 5**): $N_{\mathrm{sub}}(d_0) = n_1 \implies \mathrm{TNAO}(d_0) = n_1$.
Step 4 (**Line 7**): $\mathrm{TNAO}(\alpha_k) := \mathrm{TNAO(LSAL)} = O(n_1 \max\{n_2, n_3\})$.
Step 5 (**Line 8**): For computing $\beta_{k+1} = \beta_{k+1}^{\mathrm{HS}}, \beta_{k+1}^{\mathrm{FR}}, \beta_{k+1}^{\mathrm{PR}}, \beta_{k+1}^{\mathrm{DY}}, \beta_{k+1}^{\mathrm{HZ}}$,

(a) $N_{\mathrm{sum}}(\beta_{k+1}^{\mathrm{HS}}) = 2(n_1 - 1)$, $N_{\mathrm{mult}}(\beta_{k+1}^{\mathrm{HS}}) = 2n_1$, $N_{\mathrm{sub}}(\beta_{k+1}^{\mathrm{HS}}) = n_1$ and $N_{\mathrm{div}}(\beta_{k+1}^{\mathrm{HS}}) = 1 \implies \mathrm{TNAO}(\beta_{k+1}^{\mathrm{HS}}) = 5n_1 - 1$.

(b) $N_{\mathrm{sum}}(\beta_{k+1}^{\mathrm{FR}}) = 2(n_1 - 1)$, $N_{\mathrm{mult}}(\beta_{k+1}^{\mathrm{FR}}) = 2n_1$ and $N_{\mathrm{div}}(\beta_{k+1}^{\mathrm{FR}}) = 1 \implies \mathrm{TNAO}(\beta_{k+1}^{\mathrm{FR}}) = 4n_1 - 1$.

(c) $N_{\rm sum}(\beta_{k+1}^{\rm PR}) = 2(n_1-1)$, $N_{\rm mult}(\beta_{k+1}^{\rm PR}) = 2n_1$, $N_{\rm sub}(\beta_{k+1}^{\rm PR}) = n_1$ and $N_{\rm div}(\beta_{k+1}^{\rm PR}) = 1 \implies {\rm TNAO}(\beta_{k+1}^{\rm PR}) = 5n_1 - 1$.

(d) $N_{\rm sum}(\beta_{k+1}^{\rm DY}) = 2(n_1-1)$, $N_{\rm mult}(\beta_{k+1}^{\rm DY}) = 2n_1$, $N_{\rm sub}(\beta_{k+1}^{\rm DY}) = n_1$ and $N_{\rm div}(\beta_{k+1}^{\rm DY}) = 1 \implies {\rm TNAO}(\beta_{k+1}^{\rm DY}) = 5n_1 - 1$.

(e) $N_{\rm sum}(\beta_{k+1}^{\rm HZ}) = 3(n_1-1)$, $N_{\rm mult}(\beta_{k+1}^{\rm HZ}) = 4n_1+1$, $N_{\rm sub}(\beta_{k+1}^{\rm HZ}) = 2n_1$ and $N_{\rm div}(\beta_{k+1}^{\rm HZ}) = 2 \implies {\rm TNAO}(\beta_{k+1}^{\rm HZ}) = 9n_1$.

Step 6 (**Line 9**): $N_{\rm mult}(d_{k+1}) = n_1$ and $N_{\rm sub}(d_{k+1}) = n_1 \implies {\rm TNAO}(d_{k+1}) = 2n_1$.

As a result of Procedure 3, we can determine TNAO for Algorithms of CGHS, CGFR, CGPR, CGDY and CGHZ, which are presented by

(a) ${\rm TNAO(CGHS)} = \Big(4(n_1+1)(n_2+n_3) + 12n_1 - 2 + O(n_1 \max\{n_2, n_3\})\Big)k_{\max}$
$= O(n_1 \max\{n_2, n_3\})$;

(b) ${\rm TNAO(CGFR)} = \Big(4(n_1+1)(n_2+n_3) + 11n_1 - 2 + O(n_1 \max\{n_2, n_3\})\Big)k_{\max}$
$= O(n_1 \max\{n_2, n_3\})$;

(c) ${\rm TNAO(CGPR)} = \Big(4(n_1+1)(n_2+n_3) + 12n_1 - 2 + O(n_1 \max\{n_2, n_3\})\Big)k_{\max}$
$= O(n_1 \max\{n_2, n_3\})$;

(d) ${\rm TNAO(CGDY)} = \Big(4(n_1+1)(n_2+n_3) + 12n_1 - 2 + O(n_1 \max\{n_2, n_3\})\Big)k_{\max}$
$= O(n_1 \max\{n_2, n_3\})$;

(e) ${\rm TNAO(CGHZ)} = \Big(4(n_1+1)(n_2+n_3) + 16n_1 - 1 + O(n_1 \max\{n_2, n_3\})\Big)k_{\max}$
$= O(n_1 \max\{n_2, n_3\})$.

We see that TNAO of all algorithms is $O(n_1 \max\{n_2, n_3\})$ operations. In addition, CGHZ has the greatest amount of TNAO to compute β_{k+1} among others while CGFR has the smallest amount for it among others and TNAO to compute β_{k+1} of CGHS, CGPR and CGDY are equivalent.

Contribution. In this paper, we present a HCGN method that is a combination of both the CGDY and CGHZ methods. The parameter of this combination takes advantages of Barzilai-Borwein (BB) method [1]. The descent property of proposed method is proved with using the strong Wolfe conditions and numerical results show that our method has a low computational cost.

Organization. The rest of this paper is organized as follows. In Section 2, we describe a HCGN algorithm for smooth functional (2). In next section, descent property and global convergence of new algorithm will be investigated. In Section 4, preliminary numerical results are reported. Finally, some conclusions are given in Section 5.

2. NEW HYBRID CONJUGATE GRADIENT ALGORITHM

The traditional gradient method for solving unconstrained minimization problems is the steepest descent method whose exact step-size is computed by

$$\frac{1}{\lambda_k} := \arg\min_{\lambda>0} F_\alpha\Big(u_k - \frac{1}{\lambda} g_k\Big).$$

The steepest descent direction do not attain the fast convergence of CG methods because it use the direction $-g_k$ and may not produce the very small step-size whenever iterates are near the optimizer. Hence, to overcome this drawback, several authors dealt with various step-sizes. One of these methods is the ***BB method*** with few storage locations and inexpensive computations whose step-sizes λ_k are given by

$$\lambda_k^1 := \frac{s_{k-1}^T y_{k-1}}{s_{k-1}^T s_{k-1}} \quad \text{and} \quad \lambda_k^2 := \frac{y_{k-1}^T y_{k-1}}{s_{k-1}^T y_{k-1}}, \tag{15}$$

which $y_{k-1} := g_k - g_{k-1}$ and $s_{k-1} := u_k - u_{k-1}$. In fact, λ_k^1 is the solution of the least-squares problem

$$\begin{array}{ll} \min & \|\lambda s_{k-1} - y_{k-1}\|^2 \\ \text{s.t.} & \lambda \in \mathbb{R}, \end{array}$$

and λ_k^2 is obtained by solving

$$\begin{array}{ll} \min & \|\lambda y_{k-1} - s_{k-1}\|^2 \\ \text{s.t.} & \lambda \in \mathbb{R}. \end{array}$$

At each iteration, in order to take advantages of both λ_k^1 and λ_k^2, we introduce

$$\lambda_k := \max\Big\{\lambda_k^1, \lambda_k^2\Big\}. \tag{16}$$

Since both the CGDY and CGHZ methods have good numerical results for large-scale unconstrained optimization problems (see [17]), we use a strategy which takes advantages of both of them with the help of the parameters of the BB method. Hence, we introduce an efficient method for solving problem (4), which takes advantages of the following descent direction

$$\widetilde{d}_k := \begin{cases} -g_k & \text{if } \ k = 0, \\ -\widehat{\lambda}_k g_k + \beta_k^{\text{New}} \widetilde{d}_{k-1} & \text{if } \ k \geq 1, \end{cases} \tag{17}$$

where

$$\widehat{\lambda}_k := P_{[\lambda_{\min}, \lambda_{\max}]}\Big(\frac{1}{\lambda_k}\Big), \tag{18}$$

and the parameter β_k^{New}, the convex combination of both the β_k^{DY} and β_k^{HZ}, presents as follows:

$$\beta_k^{\text{New}} := \widehat{\lambda}_k \beta_k^{\text{HZ}} + (1 - \widehat{\lambda}_k)\beta_k^{\text{DY}}. \tag{19}$$

Let us choose the safeguard parameters as $\lambda_{\min} := 8\eta_2/(7(1+\eta_2)) + 0.01$ and $\lambda_{\max} := 1$ to guarantee the production of descent direction by (17). Hence the formula (18) implies that $\widehat{\lambda}_k \in [8\eta_2/(7(1+\eta_2)) + 0.01, 1]$.

In finally, the new hybrid conjugate gradient algorithm can be written as follows:

Algorithm 4: HCGN (Hybrid Conjugate Gradient)

Input: An initial point $u_0 \in \mathbb{R}^{n_1}$, $0 < \eta_1 < \eta_2 < 1$, $\alpha_{\max} > 0$, $\alpha_1 \in (0, \alpha_{\max})$, $\lambda_{\min} := 8\eta_2/(7(1+\eta_2)) + 0.01$ and $\lambda_{\max} := 1$;

Output: u^*, F^*_α;

```
begin
    k := 0;
    compute F_α(u_0);
    compute g_0;
    d̃_0 := −g_0;
    while not converged do
        [α_k, u_{k+1}, F_α(u_{k+1}), g_{k+1}] = LSAL(u_k, F_α(u_k), g_k, α_max, α_1);
        compute the parameters λ^1_{k+1} and λ^2_{k+1} using (15);
        calculate λ_{k+1} by formula (16);
        compute β^HZ_{k+1} by (10);
        compute β^DY_{k+1} by (9);
        choose λ̂_{k+1} satisfying (18);
        compute β^New_{k+1} by formula (19);
        obtain d̃_{k+1} by (17);
        replace k + 1 by k;
    end
    u* := u_k;  F*_α := F_α(u_k);
end
```

Since the CGHZ method has good numerical results for solving very nonlinear unconstrained optimization problems and the CGDY method has strong convergence properties, cf. [17], the parameter β_k^{New}, the convex combination of both the β_k^{HZ} and β_k^{DY}, takes advantages of the BB method. On one hand, Algorithm 4 prevents the production of very short steps whenever iterations are far away from the optimizer. On the other hand, for all iterations of near the optimizer, it prevents the generation of very large steps, which will lead to huge improvements in the object function.

By Procedures 1 – 3, we present the following procedure for computing TNAO of Algorithm 4:

Procedure 4: Calculation of TNAO for an iteration of Algorithm 4

Step 1 (**Line 3**): $\text{TNAO}(F_\alpha(u_0)) = 2n_1 + 4n_1(n_2 + n_3)$.

Step 2 (**Line 4**): $\mathrm{TNAO}(g_0) = 4(n_2 + n_3) + 2n_1 - 1$.
Step 3 (**Line 5**): $N_{\mathrm{sub}}(\widetilde{d}_0) = n_1 \implies \mathrm{TNAO}(\widetilde{d}_0) = n_1$.
Step 4 (**Line 7**): $\mathrm{TNAO}(\alpha_k) := \mathrm{TNAO}(\mathrm{LSAL}) = O(n_1 \max\{n_2, n_3\})$.
Step 5 (**Line 8**): By computing

(a) $N_{\mathrm{sub}}(y_k) = n_1$ and $N_{\mathrm{sub}}(s_k) = n_1$;

(b) $N_{\mathrm{sum}}(s_k^T s_k) = n_1 - 1$ and $N_{\mathrm{mult}}(s_k^T s_k) = n_1$;

(c) $N_{\mathrm{sum}}(y_k^T y_k) = n_1 - 1$ and $N_{\mathrm{mult}}(y_k^T y_k) = n_1$;

(d) $N_{\mathrm{sum}}(y_k^T \widetilde{d}_k) = N_{\mathrm{sum}}(s_k^T y_k) = n_1 - 1$ and $N_{\mathrm{mult}}(y_k^T \widetilde{d}_k) = N_{\mathrm{mult}}(s_k^T y_k) = n_1$;

(f) $N_{\mathrm{div}}(\lambda_k^1) = 1$ and $N_{\mathrm{div}}(\lambda_k^2) = 1$;

$\implies \mathrm{TNAO}(\lambda_k^{1,2}) := \mathrm{TNAO}(\lambda_k^1) + \mathrm{TNAO}(\lambda_k^2) = 10n_1 - 2$.
Step 6 (**Line 10**): $\mathrm{TNAO}(\beta_{k+1}^{HZ}) = 4n_1 + 2$.
Step 7 (**Line 11**): $\mathrm{TNAO}(\beta_{k+1}^{DY}) = 2n_1$.
Step 8 (**Line 12**): $N_{\mathrm{div}}(\widehat{\lambda}_{k+1}) = 1 \implies \mathrm{TNAO}(\widehat{\lambda}_{k+1}) = 1$.
Step 9 (**Line 13**): $N_{\mathrm{mult}}(\beta_{k+1}^{\mathrm{New}}) = 2$, $N_{\mathrm{sub}}(\beta_{k+1}^{\mathrm{New}}) = 1$ and $N_{\mathrm{sum}}(\beta_{k+1}^{\mathrm{New}}) = 1 \implies \mathrm{TNAO}(\beta_{k+1}^{\mathrm{New}}) = 4$.
Step 10 (**Line 14**): $N_{\mathrm{mult}}(\widetilde{d}_{k+1}) = 2n_1$ and $N_{\mathrm{sub}}(\widetilde{d}_{k+1}) = n_1 \implies \mathrm{TNAO}(\widetilde{d}_{k+1}) = 3n_1$.

Note that, in process of calculation TNAO of β_{k+1}^{DY}, the amount TNAO of y_k and $y_k^T \widetilde{d}_k$ are zero since they are considered by TNAO of λ_k^1 and λ_k^2. Also, the amount TNAO of y_k, $y_k^T \widetilde{d}_k$ and $y_k^T y_k$ are zero in process of calculation TNAO of β_{k+1}^{HZ} because they are considered by TNAO of λ_k^1 and λ_k^2. Hence, TNAO of Algorithm HCGN computes with the following formula

$$\mathrm{TNAO}(\mathrm{HCGN}) := \Big(4(n_1 + 1)(n_2 + n_3) + 24n_1 + 4 + O(n_1 \max\{n_2, n_3\})\Big)k_{\max}$$

$$= O(n_1 \max\{n_2, n_3\}),$$

which shows that it needs to $O(n_1 \max\{n_2, n_3\})$ operations.

Table 1 contains the amount of TNAO for each algorithm using the parameters β_k and d_k for CGHS, CGFR, CGPR, CGDY and CGHZ and the parameters β_k, $\widetilde{d}_k$, $\widehat{\lambda}_k$, λ_k^1 and λ_k^2 for HCGN. As a result from this table, in each iteration, HCGN has $8n_1+5$ operations more than CGHZ, $12n_1 + 6$ operations more than CGPR, CGHS and CGDY while it has $13n_1 + 6$ operations more than CGFR. Nevertheless HCGN produces the efficient direction by (17), in each iteration, which obtains the optimizer very fast than others because it produces very small steps or very large steps whenever iterations are near the optimizer or far away from it. Hence HCGN can truly decrease the total number of iterations and consequently it will lead to decrease the total number of function evaluations, gradient evaluations and CPU times while it will increase peak signal to noise ratio, see Tables 2 – 6 in Section 4.

	CGFR	CGPR	CGHS	CGDY	CGHZ	HCGN
TNAO(β_k)	$4n_1-1$	$5n_1-1$	$5n_1-1$	$5n_1-1$	$9n_1$	$6n_1+6$
TNAO(d_k)	$2n_1$	$2n_1$	$2n_1$	$2n_1$	$2n_1$	—
TNAO($\widetilde{d}_k$)	—	—	—	—	—	$3n_1$
TNAO($\lambda_k^{1,2}$)	—	—	—	—	—	$10n_1-2$
TNAO($\widehat{\lambda}_k$)	—	—	—	—	—	1
Total	$6n_1-1$	$7n_1-1$	$7n_1-1$	$7n_1-1$	$11n_1$	$19n_1+5$

Tab. 1. A comparison among TNAO of some parameters HCGN with β_k and d_k generated by CGHS, CGFR, CGPR, CGDY and CGHZ in an iteration.

3. DESCENT PROPERTY AND GLOBAL CONVERGENCE

In this section, we will investigate descent property and global convergence results of Algorithm 4. For these goals, the following assumptions are required.

(H1) For any $u_0 \in \mathbb{R}^{n_1}$, the level set $L(u_0) := \{u \in \mathbb{R}^{n_1} | F_\alpha(u) \leq F_\alpha(u_0)\}$ is bounded.

(H2) The gradient of $F_\alpha(u)$ is Lipschitz continuous over a neighborhood Ω of $L(u_0)$, i. e., there exists constant $L_g > 0$ such that

$$\|g(x) - g(y)\| \leq L_g \|x - y\|,$$

for any $x, y \in \Omega$.

(H3) The function F_α is uniformly convex, i. e., there exists constant $\gamma > 0$ such that

$$\gamma \|x - y\|^2 \leq (g(x) - g(y))^T (x - y),$$

for any $x, y \in \Omega$.

Lemma 3.1. Suppose that the direction $\widetilde{d}_{k-1}$ generated by Algorithm 4 is a descent direction. Then, $\widetilde{d}_k$ is a descent direction, i. e.,

$$g_k^T \widetilde{d}_k < 0.$$

P r o o f. By **(H3)**, there exists a constant $\gamma > 0$ such that

$$y_{k-1}^T \widetilde{d}_{k-1} \geq \gamma \|\widetilde{d}_{k-1}\|^2. \tag{20}$$

The proof follows in the following two cases:

Case 1. If $g_k^T\widetilde{d}_{k-1} < 0$, then

$$\begin{aligned} g_k^T\widetilde{d}_k &= -\widehat{\lambda}_k\|g_k\|^2 + \Big(\widehat{\lambda}_k\beta_k^{\text{HZ}} + (1-\widehat{\lambda}_k)\beta_k^{\text{DY}}\Big)g_k^T\widetilde{d}_{k-1} \\ &= \widehat{\lambda}_k\Big(\underbrace{-\|g_k\|^2 + \beta_k^{\text{HZ}}g_k^T\widetilde{d}_{k-1}}_{:=g_k^T\widetilde{d}_k^{\text{HZ}}}\Big) + \underbrace{(1-\widehat{\lambda}_k)\beta_k^{\text{DY}}g_k^T\widetilde{d}_{k-1}}_{<0} < \widehat{\lambda}_k g_k^T\widetilde{d}_k^{\text{HZ}}. \end{aligned}$$

By choosing

$$t_k := \frac{1}{2}\Big(y_{k-1}^T\widetilde{d}_{k-1}\Big)g_k \quad \text{and} \quad u_k := 2\Big(g_k^T\widetilde{d}_{k-1}\Big)y_{k-1},$$

and this fact that

$$t_k^T u_k \le \frac{1}{2}\Big(\|t_k\|^2 + \|u_k\|^2\Big),$$

we have

$$\begin{aligned} \frac{g_k^T y_{k-1}\Big(g_k^T\widetilde{d}_{k-1}\Big)}{y_{k-1}^T\widetilde{d}_{k-1}} &= \frac{g_k^T y_{k-1}\Big(y_{k-1}^T\widetilde{d}_{k-1}\Big)\Big(g_k^T\widetilde{d}_{k-1}\Big)}{\Big(y_{k-1}^T\widetilde{d}_{k-1}\Big)^2} \\ &\le \frac{1}{8}\|g_k\|^2 + 2\frac{\|y_{k-1}\|^2\Big(g_k^T\widetilde{d}_{k-1}\Big)^2}{\Big(y_{k-1}^T\widetilde{d}_{k-1}\Big)^2}. \end{aligned} \tag{21}$$

The formula (21) gives

$$g_k^T\widetilde{d}_k^{\text{HZ}} \overset{(10)}{=} -\|g_k\|^2 + \frac{g_k^T y_{k-1}\Big(g_k^T\widetilde{d}_{k-1}\Big)}{y_{k-1}^T\widetilde{d}_{k-1}} - 2\frac{\|y_{k-1}\|^2\Big(g_k^T\widetilde{d}_{k-1}\Big)^2}{\Big(y_{k-1}^T\widetilde{d}_{k-1}\Big)^2} \le -\frac{7}{8}\|g_k\|^2 < 0. \tag{22}$$

Setting (22) in (21) leads

$$g_k^T\widetilde{d}_k < -\frac{7}{8}\widehat{\lambda}_k\|g_k\|^2 < 0.$$

Case 2: If $g_k^T\widetilde{d}_{k-1} > 0$, then

$$\begin{aligned} g_k^T\widetilde{d}_k &= -\widehat{\lambda}_k\|g_k\|^2 + \Big(\widehat{\lambda}_k\beta_k^{\text{HZ}} + (1-\widehat{\lambda}_k)\beta_k^{\text{DY}}\Big)g_k^T\widetilde{d}_{k-1} \\ &= \widehat{\lambda}_k\Big(\underbrace{-\|g_k\|^2 + \beta_k^{\text{HZ}}g_k^T\widetilde{d}_{k-1}}_{:=g_k^T\widetilde{d}_k^{\text{HZ}}}\Big) + \underbrace{\beta_k^{\text{DY}}g_k^T\widetilde{d}_{k-1}}_{>0} - \underbrace{\widehat{\lambda}_k\beta_k^{\text{DY}}g_k^T\widetilde{d}_{k-1}}_{>0} \\ &\overset{(9),(22)}{<} -\frac{7}{8}\widehat{\lambda}_k\|g_k\|^2 + \frac{\|g_k\|^2}{y_{k-1}^T\widetilde{d}_{k-1}}g_k^T\widetilde{d}_{k-1} \\ &= \frac{-\frac{7}{8}\widehat{\lambda}_k\|g_k\|^2 y_{k-1}^T\widetilde{d}_{k-1} + \|g_k\|^2 g_k^T\widetilde{d}_{k-1}}{y_{k-1}^T\widetilde{d}_{k-1}} \end{aligned}$$

$$
\begin{aligned}
&= \|g_k\|^2 \frac{-\frac{7}{8}\widehat{\lambda}_k y_{k-1}^T \widetilde{d}_{k-1} + g_k^T \widetilde{d}_{k-1}}{y_{k-1}^T \widetilde{d}_{k-1}} \\
&= \|g_k\|^2 \frac{-\frac{7}{8}\widehat{\lambda}_k \Big(g_k^T \widetilde{d}_{k-1} - g_{k-1}^T \widetilde{d}_{k-1}\Big) + g_k^T \widetilde{d}_{k-1}}{y_{k-1}^T \widetilde{d}_{k-1}} \\
&= \|g_k\|^2 \frac{\Big(1 - \frac{7}{8}\widehat{\lambda}_k\Big) g_k^T \widetilde{d}_{k-1} + \frac{7}{8}\widehat{\lambda}_k g_{k-1}^T \widetilde{d}_{k-1}}{y_{k-1}^T \widetilde{d}_{k-1}} \\
&= \|g_k\|^2 \frac{\Big(1 - \frac{7}{8}\widehat{\lambda}_k\Big) \Big|g_k^T \widetilde{d}_{k-1}\Big| + \frac{7}{8}\widehat{\lambda}_k g_{k-1}^T \widetilde{d}_{k-1}}{y_{k-1}^T \widetilde{d}_{k-1}} \\
&\overset{(12)}{\leq} \|g_k\|^2 \frac{\Big(1 - \frac{7}{8}\widehat{\lambda}_k\Big)\Big(-\eta_2 g_{k-1}^T \widetilde{d}_{k-1}\Big) + \frac{7}{8}\widehat{\lambda}_k g_{k-1}^T \widetilde{d}_{k-1}}{y_{k-1}^T \widetilde{d}_{k-1}} \\
&\overset{(18),(20)}{=} \|g_k\|^2 \frac{\Big(\underbrace{(1 - \frac{7}{8}\widehat{\lambda}_k)(-\eta_2) + \frac{7}{8}\widehat{\lambda}_k}_{>0}\Big) \underbrace{g_{k-1}^T \widetilde{d}_{k-1}}_{<0}}{\underbrace{y_{k-1}^T \widetilde{d}_{k-1}}_{>0}} < 0.
\end{aligned}
$$

□

Dai and Ni [11] established that for any conjugate gradient method with the strong Wolfe line search (11) and (12) under Assumptions **(H1)** and **(H2)** there exists a constant $\varepsilon \geq 0$ such that $\|g_k\| \leq \varepsilon$ for all $u_k \in L(u_0)$. Hence we have the following result.

Lemma 3.2. Suppose that Assumptions **(H1)** and **(H2)** hold and $\{u_k\}$ is the sequence generated by Algorithm 4. If

$$\sum_{k\geq 1} \frac{1}{\|\widetilde{d}_k\|^2} = \infty,$$

then

$$\liminf_{k\to\infty} \|g_k\| = 0.$$

Theorem 3.3. Suppose that Assumptions **(H1)** – **(H3)** hold and $\{u_k\}$ is the sequence generated by Algorithm 4. Also, suppose that there exists $\zeta > 0$ such that $\|g_k\|^2 \leq \zeta \|\widetilde{d}_{k-1}\|$. Then

$$\lim_{k\to\infty} g_k = 0$$

P r o o f . Assumptions **(H1)** and **(H2)** imply that there exists a constant $\varepsilon \geq 0$ such that

$$\|g_k\| \leq \varepsilon.$$

Since β_k^{New} is the convex combination of β_k^{DY} and β_k^{HZ}, our proof is divided into the following two cases:

i) If $\beta_k^{\text{DY}} \leq \beta_k^{\text{HZ}}$, then

$$\begin{aligned}
\|\widetilde{d}_k\| &\leq \widehat{\lambda}_k\|g_k\| + |\beta_k^{\text{HZ}}|\|\widetilde{d}_{k-1}\| \\
&\leq \|g_k\| + \Big(\Big|g_k^T y_{k-1}\Big|\frac{1}{y_{k-1}^T\widetilde{d}_{k-1}} + (2\|y_{k-1}\|^2)\Big(\Big|g_k^T\widetilde{d}_{k-1}\Big|\frac{1}{(y_{k-1}^T\widetilde{d}_{k-1})^2}\Big)\Big)\|\widetilde{d}_{k-1}\| \\
&\overset{(20)}{\leq} \|g_k\| + \Big(\frac{\|g_k\|\|y_{k-1}\|}{\gamma\|\widetilde{d}_{k-1}\|^2} + (2\|y_{k-1}\|^2)\frac{\|g_k\|\|\widetilde{d}_{k-1}\|}{\gamma^2\|\widetilde{d}_{k-1}\|^4}\Big)\|\widetilde{d}_{k-1}\| \\
&\overset{\textbf{(H2)}}{\leq} \|g_k\| + \Big(\frac{\|g_k\|L_g\|\widetilde{d}_{k-1}\|}{\gamma\|\widetilde{d}_{k-1}\|^2} + (2L_g^2\|\widetilde{d}_{k-1}\|^2)\frac{\|g_k\|\|\widetilde{d}_{k-1}\|}{\gamma^2\|\widetilde{d}_{k-1}\|^4}\Big)\|\widetilde{d}_{k-1}\| \\
&= \|g_k\| + \frac{\|g_k\|L_g}{\gamma} + \frac{2\|g_k\|L_g^2}{\gamma^2} \leq \delta_1,
\end{aligned}$$

which $\delta_1 := \varepsilon(1 + \frac{L_g}{\gamma} + \frac{2L_g^2}{\gamma^2})$.

ii) If $\beta_k^{\text{HZ}} < \beta_k^{\text{DY}}$, then

$$\begin{aligned}
\|\widetilde{d}_k\| &< \widehat{\lambda}_k\|g_k\| + |\beta_k^{\text{DY}}|\|\widetilde{d}_{k-1}\| \\
&\leq \|g_k\| + \frac{\|g_k\|^2}{y_{k-1}^T\widetilde{d}_{k-1}}\|\widetilde{d}_{k-1}\| \\
&\leq \|g_k\| + \frac{\zeta\|\widetilde{d}_{k-1}\|}{\gamma\|\widetilde{d}_{k-1}\|^2}\|\widetilde{d}_{k-1}\| \\
&= \|g_k\| + \frac{\zeta}{\gamma} \leq \delta_2,
\end{aligned}$$

which $\delta_2 := \varepsilon + \frac{\zeta}{\gamma}$.
By taking $\delta := \max\{\delta_1, \delta_2\}$, both cases give

$$\|\widetilde{d}_k\| \leq \delta.$$

This fact implies that

$$\sum_{k\geq 1}\frac{1}{\|\widetilde{d}_k\|^2} = \infty,$$

for which Lemma 3.2 leads to

$$\liminf_{k\to\infty}\|g_k\| = 0,$$

which for uniformly convex functions is equivalent to

$$\lim_{k\to\infty} g_k = 0.$$

□

Some properties of the edge-preserving regularization functional (4), which have been established in [4], are as follows:

Theorem 3.4. If φ_α is second order Lipschitz continuous, continuously differentiable, convex, strictly convex, or coercive, then the functional F_α is respectively second order Lipschitz continuous, continuously differentiable, convex, strictly convex, or coercive.

Theorem 3.5. If $\varphi_\alpha(t)$ is even, continuous and strictly increasing $w.r.t.|t|$, then the global minimum of F_α exists, and any global minimizer u^* is in the dynamic range, i. e., $u^*_{i,j} \in [s_{\min}, s_{\max}]$ for all $(i,j) \in \mathcal{N}$.

4. PRELIMINARY NUMERICAL EXPERIMENTS

In this section, we present some numerical results to demonstrate the performance of HCGN for salt-and-pepper impulse noise removal. In our experiments, we first compare HCGN with several versions of CG methods which their details are as

- CGHZ: conjugate gradient method proposed by Hager and Zhang [16]
- CGDY: conjugate gradient method proposed by Dai and Yuan [12]
- CGHS: conjugate gradient method proposed by Hestenes and Stiefel [18]
- CGPR: conjugate gradient method proposed by Polak and Ribière [25]
- CGFR: conjugate gradient method proposed by Fletcher and Reeves [14].

The simulations are preformed in Matlab 2015 on a laptop Asus with a 1.7 GHz Intel Core i3-4010U CPU and 4 GB of memory (2 GB is used) under ubuntu 10.04 Linux on Oracle VM VirtualBox. The test images are Lena, House, Cameraman and Elaine. Similar to [4, 29], we use the PSNR (peak signal to noise ratio) in order to assess the restoration performance qualitatively which is defined as

$$\mathrm{PSNR} := 20\log_{10}\frac{255^2}{\frac{1}{MN}\Sigma_{i,j}(u^r_{i,j}-u^*_{i,j})^2},$$

where $u^r_{i,j}$ and $u^*_{i,j}$ denote the pixel values of the restored image and the original image, respectively. The stopping criterions of all algorithms are

$$\frac{F_\alpha(u_k)-F_\alpha(u_{k-1})}{F_\alpha(u_k)} < 10^{-4} \quad \text{and} \quad \|\nabla F_\alpha(u_k)\| \le 10^{-4}(1+|F_\alpha(u_k)|).$$

The parameters of Wolfe conditions are chosen $\eta_1 = 0.0001$ and $\eta_2 = 0.5$ in all algorithms. For HCGN, we choose the parameters $\lambda_{\min} = 8\eta_2/(7(1+\eta_2)) + 0.01 \simeq 0.39$ and $\lambda_{\max} = 1$. It should be emphasized that in this paper, we are mainly concerned with the speed of solving the minimization of the edge-preserving regularization function (4), in which the potential function is $\varphi_\alpha(t) = \sqrt{t^2+\alpha}$.

In the first run, we perform all algorithms for 5 different noise samples of each image in order to test the speed of the algorithm more fairly, for

$$\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5, 1, 2, 3, 4, 5, 10, 20, 30, 40, 50\}.$$

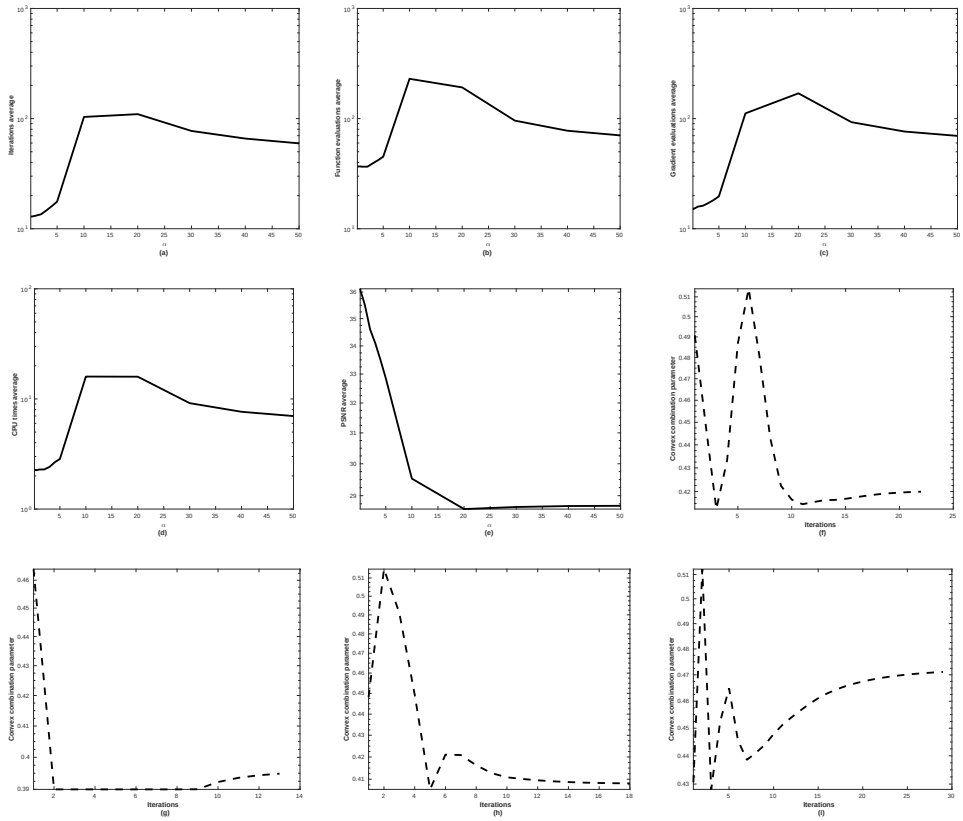

Fig. 1. Comparisons for HCGN: (a) Diagram of N_i versus α; (b) Diagram of N_f versus α; (c) Diagram of N_g versus α; (d) Diagram of C_t versus α; (e) Diagram of PSNR versus α; (f) – (i) Diagrams of $\widehat{\lambda}_k$ versus iterations for Lena, House, Cameraman 256×256 and Cameraman 512×512, respectively, with noise ratio 90%.

Then, the average of the total number of iterates (N_i), function evaluations (N_f), gradient evaluations (N_g), CPU times required (C_t) and PSNR have reported in the Tables 2 – 6. In Tables 2 – 6, we see that HCGN is better than other methods in N_i, N_f, N_g, C_t and PSNR, respectively. Efficiency comparisons of all codes have been made using the performance profile introduced by Dolan and Moré in [13] based on N_i, N_f, N_g, C_t and PSNR in Figures 1 – 7. In these figures, P designates the percentage of problems which are solved within a factor τ of the best solver. Subfigures (a) – (e) of Figure 1 show that with increasing the amount of α, the average of the total number of iterates, function evaluations, gradient evaluations and CPU times will increase while PSNR of them decreases. Therefore, with increasing amount of α the efficiency of HCGN will decrease. Hence, based on the theory of the performance profile above and to give a clear view of the behaviour of all algorithms, we depict the contour plot of the obtained N_i, N_f, N_g, C_t and PSNR for

$$\alpha \in \{0.1, 0.2, 0.3, 0.4, 0.5\},$$

in Figures 2 and 3.

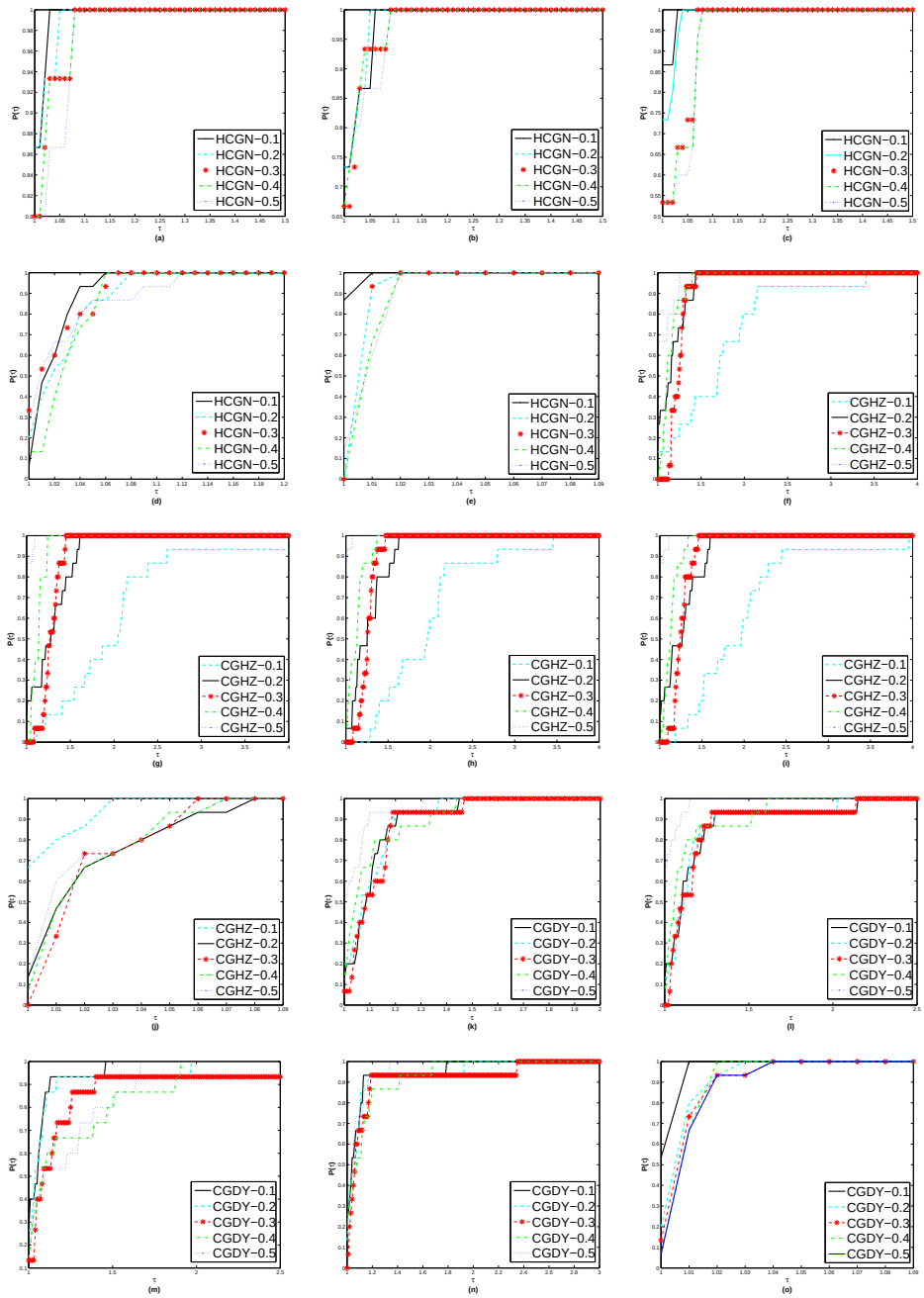

Fig. 2. Comparisons for versions of HCGN, CGHZ and CGDY by performance profiles with the measures N_i, N_f, N_g, C_t and PSNR: (a) – (e) displays the performance profile for HCGN-0.1, HCGN-0.2, HCGN-0.3, HCGN-0.4 and HCGN-0.5; (f) – (j) displays the performance profile for CGHZ-0.1, CGHZ-0.2, CGHZ-0.3, CGHZ-0.4 and CGHZ-0.5; (k) – (o) displays the performance profile for CGDY-0.1, CGDY-0.2, CGDY-0.3, CGDY-0.4 and CGDY-0.5.

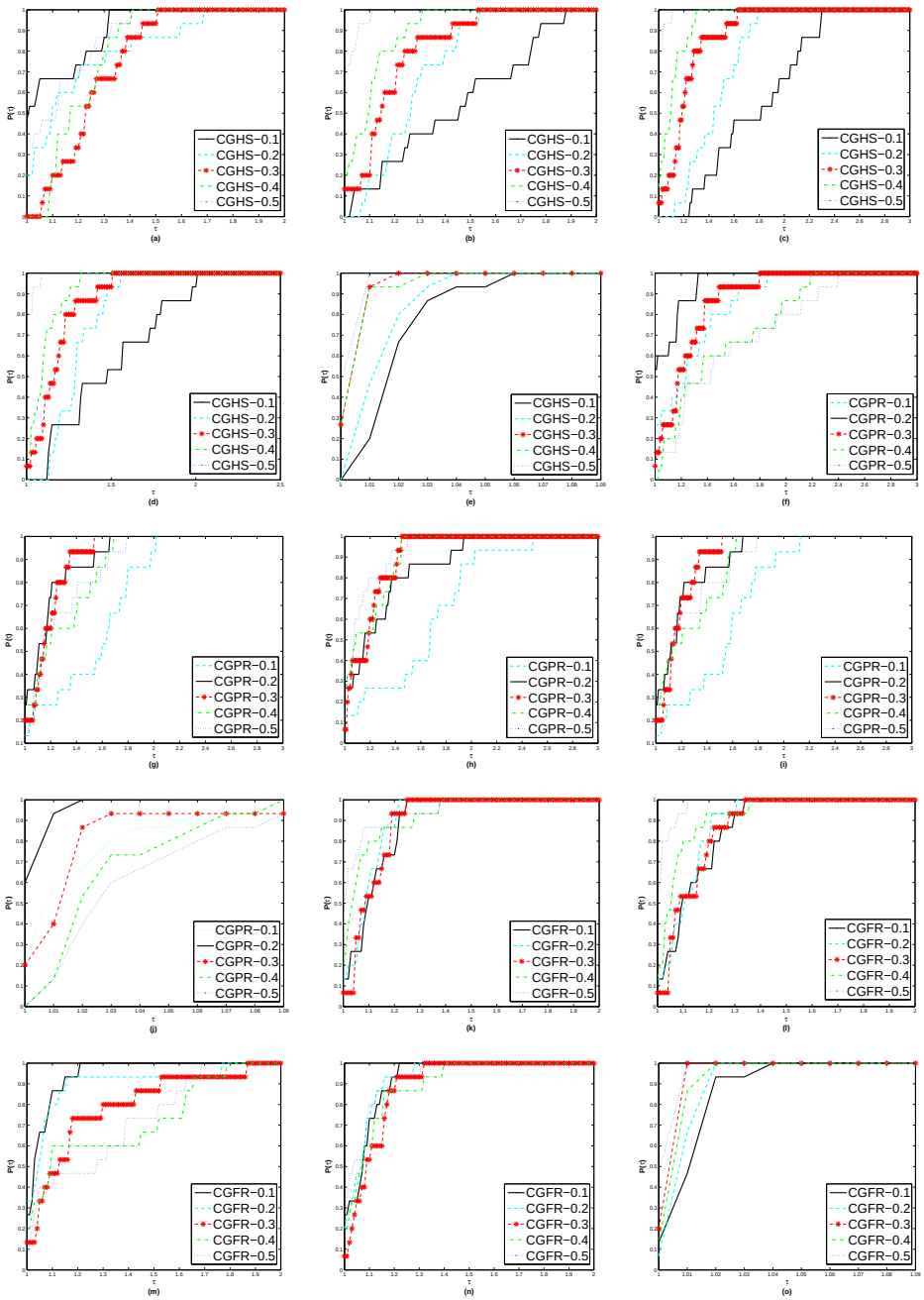

Fig. 3. Comparisons for versions of CGHS, CGPR and CGFR by performance profiles with the measures N_i, N_f, N_g, C_t and PSNR: (a) – (e) displays the performance profile for CGHS-0.1, CGHS-0.2, CGHS-0.3, CGHS-0.4 and CGHS-0.5; (f) – (j) displays the performance profile for CGPR-0.1, CGPR-0.2, CGPR-0.3, CGPR-0.4 and CGPR-0.5; (k) – (o) displays the performance profile for CGFR-0.1, CGFR-0.2, CGFR-0.3, CGFR-0.4 and CGFR-0.5.

Subfigures (a) – (e) of Figure 2 show that the results of HCGN for $\alpha = 0.1$ are considerably better than results of this algorithm for other considered values, especially, Subfigure (e) of Figure 2 confirms the sensible excellence of HCGN in PSNR for this value. Although, Subfigures (f) – (i) of Figure 2 demonstrate that the results of CGHZ method for $\alpha = 0.5$ are competitive with the other values of this parameter in N_i, N_f, N_g and C_t, while from Subfigure (j), it is clear that PSNR has the best performance for $\alpha = 0.1$. Also, the performance profiles of CGDY method for different values of α are compared in the sense of N_i, N_f, N_g, C_t and PSNR in Subfigures (k) – (o) of Figure 2, respectively. In this method, the optimal parameter α for PSNR is similar to the two earlier methods. In Figure 3, we display the results of CGHS, CGPR and CGFR methods by using performance profiles for the measures N_i, N_f, N_g, C_t and PSNR. From Subfigures (e) and (o) of Figure 3, it is quite evident that the best results for PSNR are achieved by $\alpha = 0.5$ in CGHS and CGFR methods while the CGPR method has best performance for PSNR in $\alpha = 0.2$. Subfigures (a) – (e) of Figure 4 show that HCGN has best results among others for $\alpha = 0.1$. For N_i, N_f, N_g, C_t and PSNR, it wins 100%, 100%, 100%, 100% and 88% for most of the test pictures, respectively.

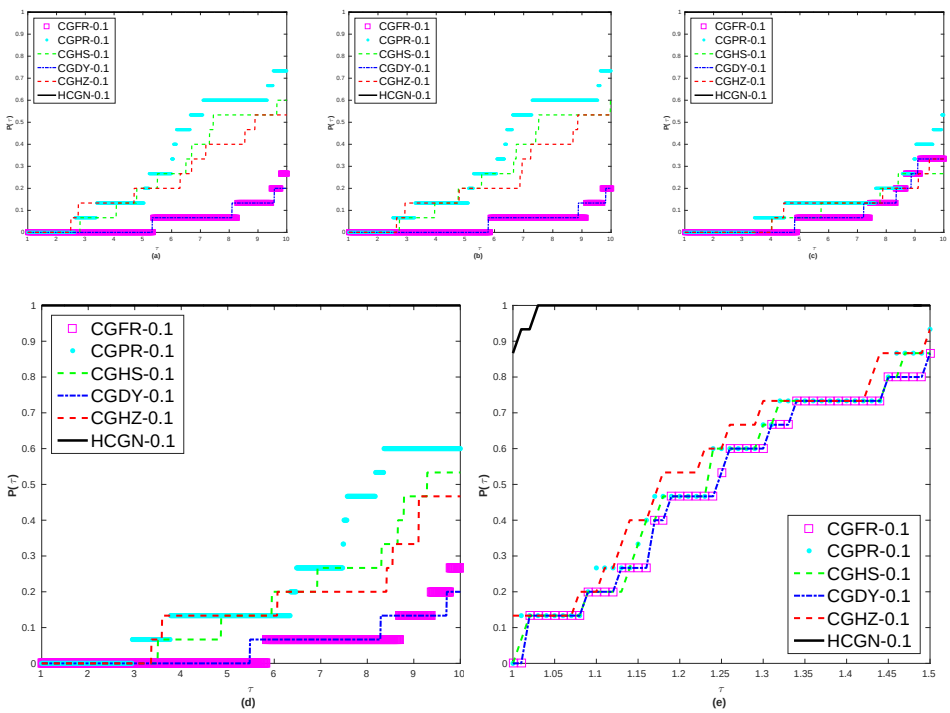

Fig. 4. Comparisons for all algorithms by performance profiles with the measures N_i, N_f, N_g, C_t and PSNR: (a) – (e) displays the performance profile for $\alpha = 0.1$.

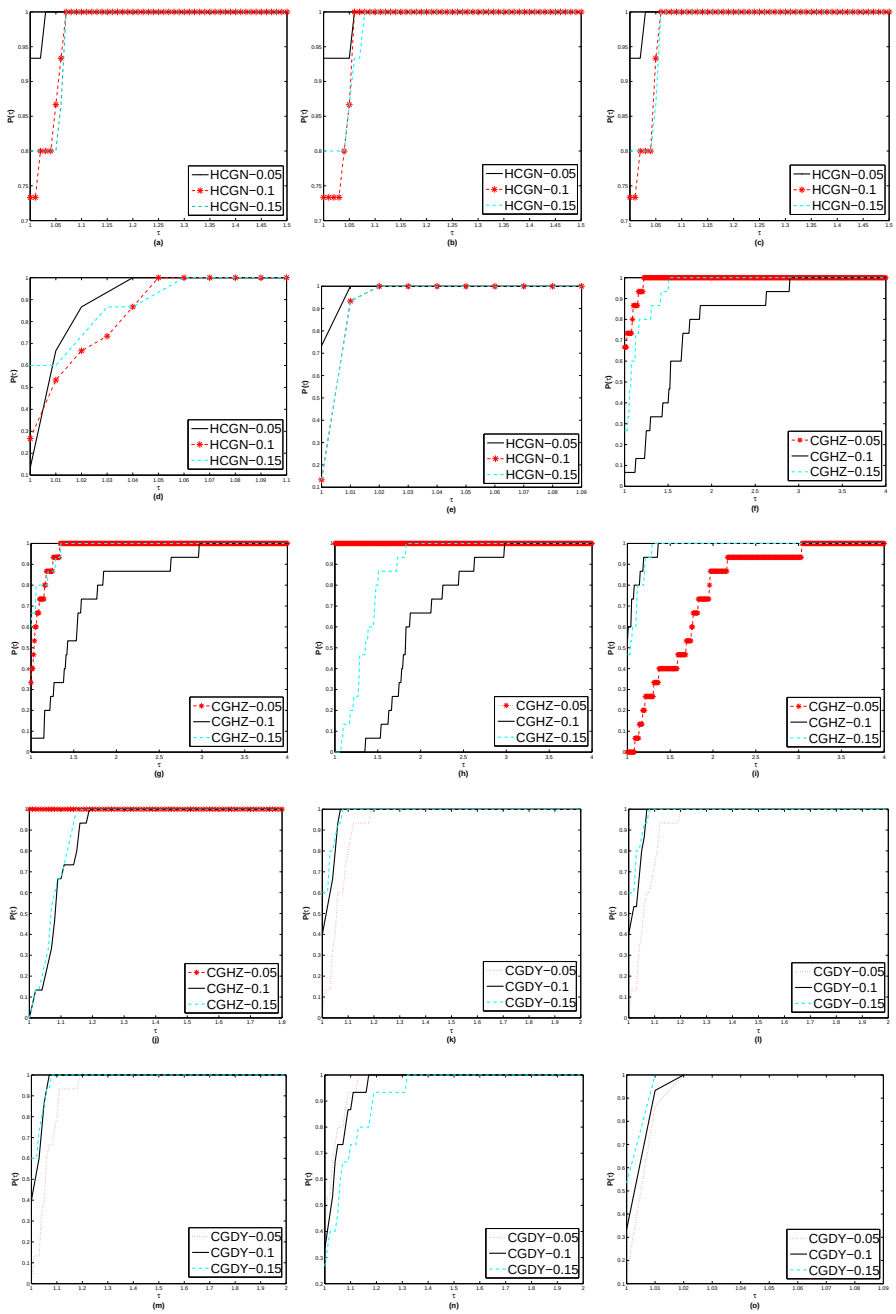

Fig. 5. Comparisons for versions of HCGN, CGHZ and CGDY by performance profiles with the measures N_i, N_f, N_g, C_t and PSNR: (a) – (e) displays the performance profile for HCGN-0.05, HCGN-0.1 and HCGN-0.15; (f) – (j) displays the performance profile for CGHZ-0.05, CGHZ-0.1 and CGHZ-0.15; (k) – (o) displays the performance profile for CGDY-0.05, CGDY-0.1 and CGDY-0.15.

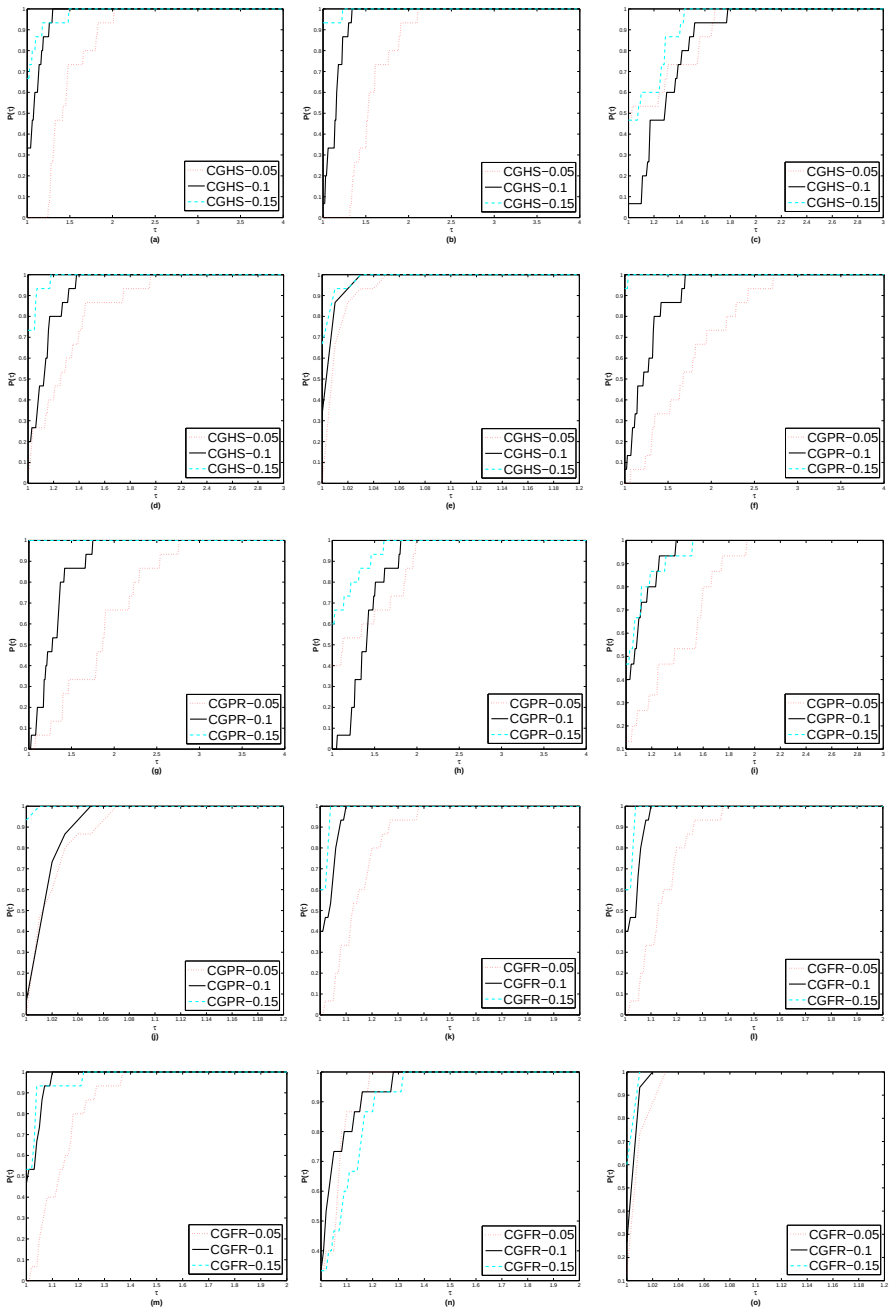

Fig. 6. Comparisons for versions of CGHS, CGPR and CGFR by performance profiles with the measures N_i, N_f, N_g, C_t and PSNR: (a) – (e) displays the performance profile for CGHS-0.05, CGHS-0.1 and CGHS-0.15; (f) – (j) displays the performance profile for CGPR-0.05, CGPR-0.1 and CGPR-0.15; (k) – (o) displays the performance profile for CGFR-0.05, CGFR-0.1 and CGFR-0.15.

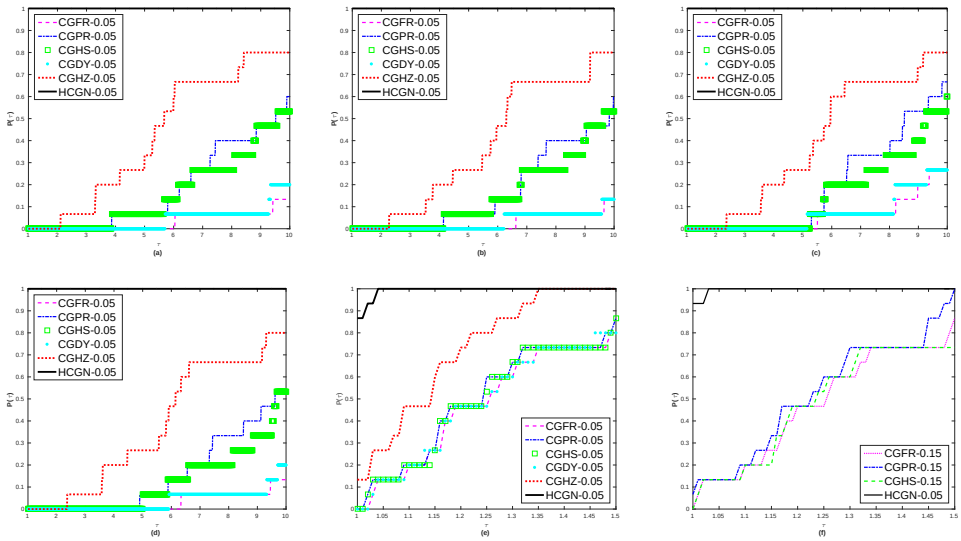

Fig. 7. Comparisons for all algorithms by performance profiles with the measures N_i, N_f, N_g, C_t and PSNR: (a) – (e) displays the performance profile for $\alpha = 0.05$; (f) display the performance profile of PSNR for $\alpha = 0.05$ and $\alpha = 0.15$.

Note that, if convex combination parameter $\widehat{\lambda}_k$ in (19) tends to 1, then β_k^{New} inherits only the advantage of CGHZ method. However, Subfigures (f) – (i) of Figure 1 demonstrate that the convex combination parameter takes values between 0.39 and 1 in different iterations and therefore the parameter β_k^{New} inherits advantages of both β_k^{HZ} and β_k^{DY}.

Image	Size	CGFR	CGPR	CGHS	CGDY	CGHZ	HCGN
Lena	256×256	226.3822	109.4800	118.1156	219.6000	97.2711	45.8756
House	256×256	172.4978	84.1467	91.8311	168.5067	76.3733	29.2444
Cameraman	256×256	284.3644	118.7067	128.9387	277.3689	116.4933	46.4311
Cameraman	512×512	112.3689	62.2933	69.1511	111.1022	54.3733	24.5289
Elaine	512×512	170.4756	96.6844	100.4667	163.0311	89.4000	39.0667
Average		193.21778	94.26222	101.70064	187.92178	86.78220	37.02934

Tab. 2. Iterations.

Image	Size	CGFR	CGPR	CGHS	CGDY	CGHZ	HCGN
Lena	256×256	570.0711	222.3111	236.1911	544.7422	202.5289	88.4400
House	256×256	442.0267	173.6711	185.7156	419.9644	162.6622	55.5289
Cameraman	256×256	747.7911	246.3022	262.6400	725.1467	247.9111	86.9200
Cameraman	512×512	275.1600	123.2978	135.2800	264.3867	110.2178	44.6756
Elaine	512×512	427.5511	200.8444	205.5333	402.6444	192.0267	73.9689
Average		492.52000	193.28532	205.07200	471.37688	183.06934	69.85870

Tab. 3. Function evaluations.

Image	Size	CGFR	CGPR	CGHS	CGDY	CGHZ	HCGN
Lena	256×256	252.4133	148.0400	156.9022	236.5733	130.9244	56.0356
House	256×256	196.6133	110.8978	118.1644	178.1733	101.8978	36.7778
Cameraman	256×256	316.2978	160.7022	171.9422	297.5111	151.2400	57.6089
Cameraman	512×512	130.7244	79.7022	86.9778	121.1289	73.4311	29.8356
Elaine	512×512	194.3022	128.7244	132.4756	177.9556	119.8400	47.4089
Average		218.07014	125.61332	133.29244	202.26844	115.46666	45.53336

Tab. 4. Gradient evaluations.

Image	Size	CGFR	CGPR	CGHS	CGDY	CGHZ	HCGN
Lena	256×256	17.6342	8.4763	9.0299	16.8709	7.6500	3.3616
House	256×256	13.5885	6.4361	6.8989	12.9039	6.0144	2.1361
Cameraman	256×256	22.5252	9.2437	9.8946	21.6945	9.1000	3.3935
Cameraman	512×512	35.4139	17.7439	20.3977	32.8135	16.7664	6.7043
Elaine	512×512	52.5127	29.5980	30.4606	49.5570	28.3075	10.8813
Average		28.33490	14.29960	15.33634	26.76796	13.56766	5.29536

Tab. 5. CPU times.

Image	Size	CGFR	CGPR	CGHS	CGDY	CGHZ	HCGN
Lena	256×256	26.9977	27.1827	27.0526	27.0002	27.5396	31.3699
House	256×256	30.5509	30.7321	30.5247	30.5468	31.0894	34.9959
Cameraman	256×256	24.7873	25.0162	24.8775	24.7871	25.3490	29.8770
Cameraman	512×512	31.3882	31.4705	31.3481	31.3896	31.6652	32.4783
Elaine	512×512	30.4517	30.6491	30.4956	30.4518	31.0334	36.1198
Average		28.83516	29.01012	28.8597	28.8351	29.33532	32.96818

Tab. 6. PSNR.

Let us denote ϵ-machine with ϵ_m. According to obtained results form Figures 1 – 4 and Tables 2 – 6, we choose $(\epsilon_m, 0.15]$ as the interval including the optimal α and run again all algorithms similar to conditions of the first run for $\alpha \in \{0.05, 0.1, 0.15\}$ of this interval. In Figure 5, Subfigures (a) – (e) illustrate the performance profiles of the HCGN for N_i, N_f, N_g, C_t and PSNR with different values of α in optimal interval. From these subfigures, our method for $\alpha = 0.05$ has the best results in competitive with the others values of this parameter. Also, Subfigure (o) of Figure 5 and Subfigures (e), (j) and (o) of Figure 6 show that CGDY, CGHS, CGPR and CGFR algorithms for $\alpha = 0.15$ have the best results for PSNR. Hence, Subfigure (f) of Figure 7 shows that HCGN Algorithm for $\alpha = 0.05$ has the best results in competitive with CGFR-0.15, CGPR-0.15 and CGHS-0.15 for PSNR. Finally, from Subfigures (a) – (e) of Figure 7, HCGN Algorithm for $\alpha = 0.05$ wins 100%, 100%, 100%, 100% and 87% in N_i, N_f, N_g, C_t and PSNR for most of the test pictures in competitive with others.

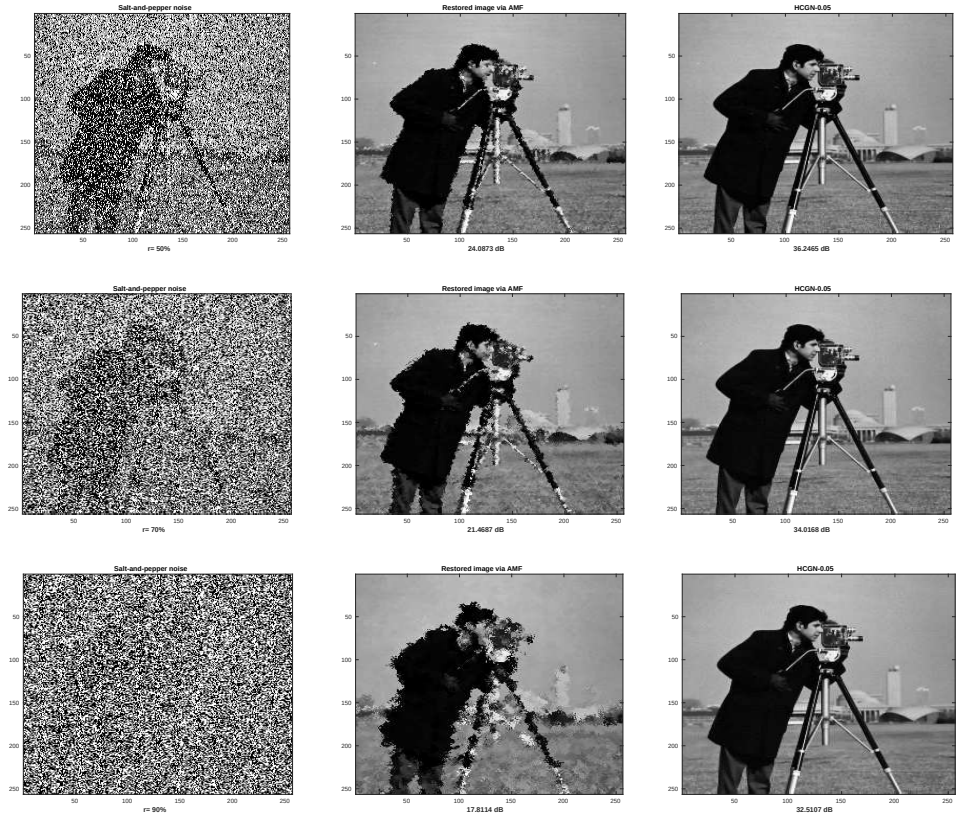

Fig. 8. The noisy images\the restored images via AMF\the restored images via HCGN-0.05 for 256×256 Cameraman image.

Figures 8 – 12 show the obtained results by AMF and the two-phase schemes solved by the HCGN method. These results show that HCGN with $\alpha = 0.05$ can restore corrupted image quite well in an efficient manner.

Finally, we compare HCGN ($\alpha = 0.05$) with the following considered algorithms

- BB1: This is a version of Algorithm 3, using $d_k := -\frac{1}{\lambda_k^1}\nabla F_\alpha(u_k)$ instead of $\widetilde{d}_k$ generated by (5) (Line 9) and expect Line 8.
- BB2: This is a version of Algorithm 3, using $d_k := -\frac{1}{\lambda_k^2}\nabla F_\alpha(u_k)$ instead of $\widetilde{d}_k$ generated by (5) (Line 9) and expect Line 8.
- LBFGS: This is Algorithm 7.5 in Nocedal and Wright [23] (Chapter 7, Page 198), using $d_k := -H_k\nabla F_\alpha(u_k)$, in which H_k is a quasi-Newton approximation of the inverse matrix B_k^{-1} generated by the well-known LBFGS approach developed by Liu and Nocedal in [20] and Nocedal in [22]. Note that B_k is an approximation of $\nabla^2 F_\alpha(u_k)$, which it satisfies in the following condition, known as the secant

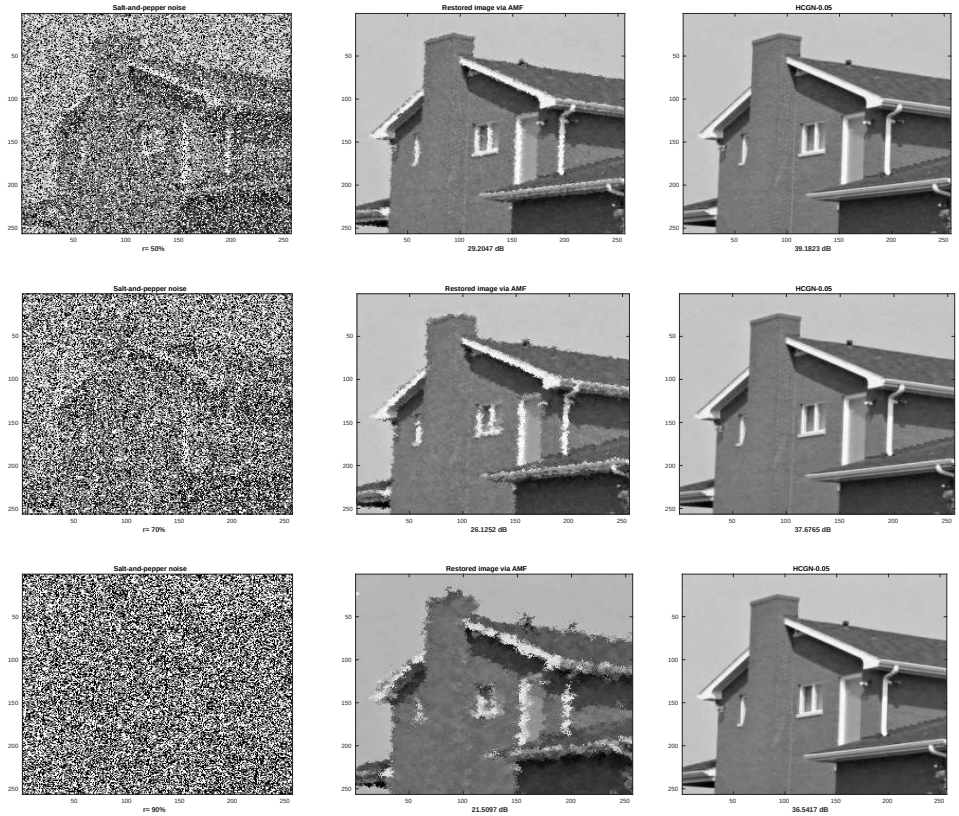

Fig. 9. The noisy images\the restored images via AMF\the restored images via HCGN-0.05 for 256×256 House image.

equation,

$$B_{k+1}s_k = y_k,$$

where $s_k := u_{k+1} - u_k$ and $y_k := \nabla F_\alpha(u_{k+1}) - \nabla F_\alpha(u_k)$. In the LBFGS method, matrix H_{k+1} is defined by

$$\begin{aligned} H_{k+1} := & \left(V_k^T \cdots V_{k-m}^T\right) H_0 (V_{k-m} \cdots V_k) \\ & + \theta_{k-m} \left(V_k^T \cdots V_{k-m+1}^T\right) s_{k-m} s_{k-m}^T (V_{k-m+1} \cdots V_k) \\ & + \theta_{k-m+1} \left(V_k^T \cdots V_{k-m+2}^T\right) s_{k-m+1} s_{k-m+1}^T (V_{k-m+2} \cdots V_k) \\ & \vdots \\ & + \theta_k s_k s_k^T, \end{aligned} \tag{23}$$

where H_0 is a symmetric and positive-definite starting matrix, $\theta_k := \frac{1}{y_k^T s_k}$, $V_k := I - \theta_k y_k s_k^T$ and $m := \min\{k, 5\}$.

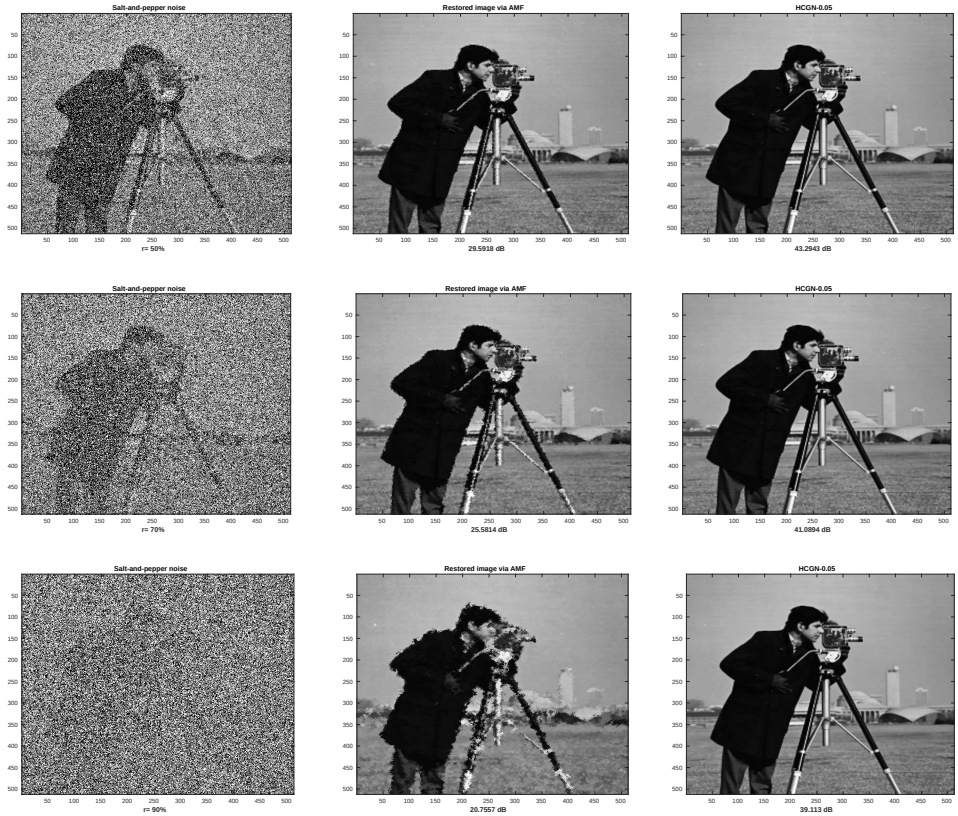

Fig. 10. The noisy images\the restored images via AMF\the restored images via HCGN-0.05 for 512×512 Cameraman image.

Note that BB1, BB2 and LBFGS are suitable to solve the smooth problem (2) since they have low computational costs. In Subfigures of Figure 13, it can be seen that the HCGN ($\alpha = 0.05$) is the best solver, in terms of N_i, N_f, N_g and C_t on 100% of the problems and PSNR on 88% of the problems, while LBFGS, BB1 and BB2 have approximately the same manner.

5. CONCLUDING REMARKS

In this paper, we introduce an extended CG method to minimize the smooth regularization functional for impulse noise removal. The parameter β_k^{New} of new method is the convex combination of both β_k^{HZ} and β_k^{DY} such that it takes advantages of BB method. Numerical results show that the new method has low computational cost and is efficient to solve signal processing problems.

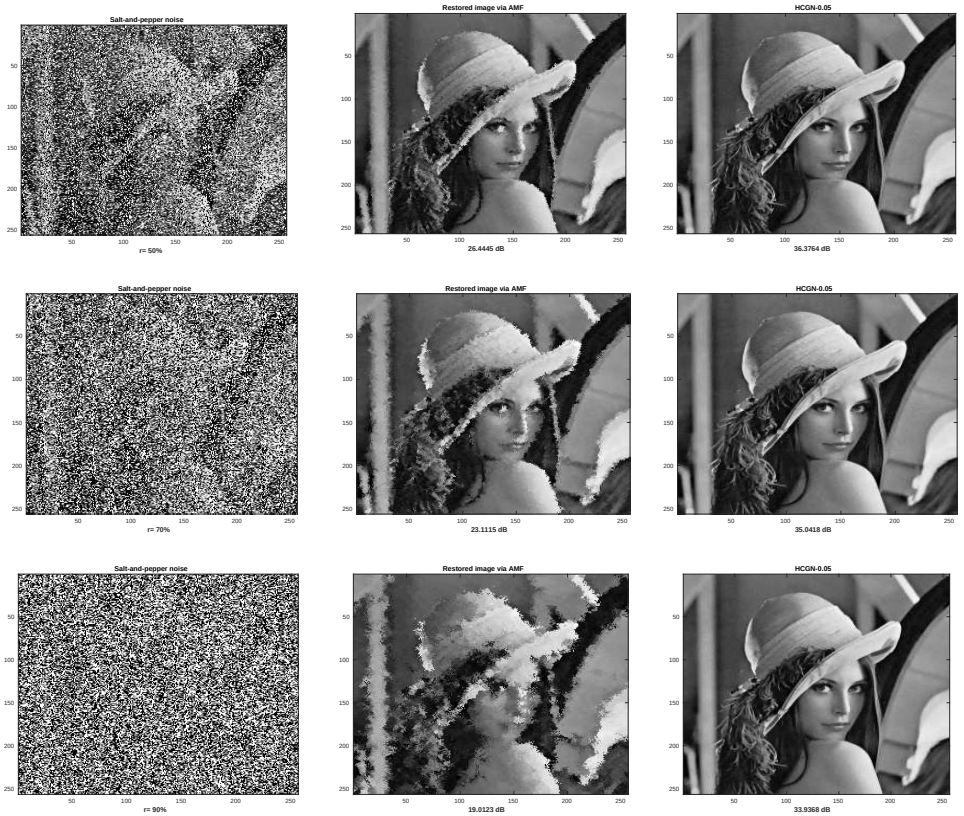

Fig. 11. The noisy images\the restored images via AMF\the restored images via HCGN-0.05 for 256×256 Lena image.

ACKNOWLEDGEMENT

The authors are grateful to anonymous referees for their valuable comments and suggestions that improve the paper.

(Received August 17, 2015)

REFERENCES

[1] J. Barzilai and J. M. Borwein: Two point step size gradient method. IMA J. Numer. Anal. *8* (1988), 141–148. DOI:10.1093/imanum/8.1.141

[2] M. Bertalmio, L. A. Vese, G. Sapiro, and S. Osher: Simultaneous structure and texture image inpainting. IEEE Trans. Image Processing. *12* (2003), 8, 882–889. DOI:10.1109/tip.2003.815261

[3] J. F. Cai, R. H. Chan, and C. D. Fiore: Minimization of a detail-preserving regularization functional for impulse noise removal. J. Math. Imaging Vision. *27* (2007), 79–91. DOI:10.1007/s10851-007-0027-4

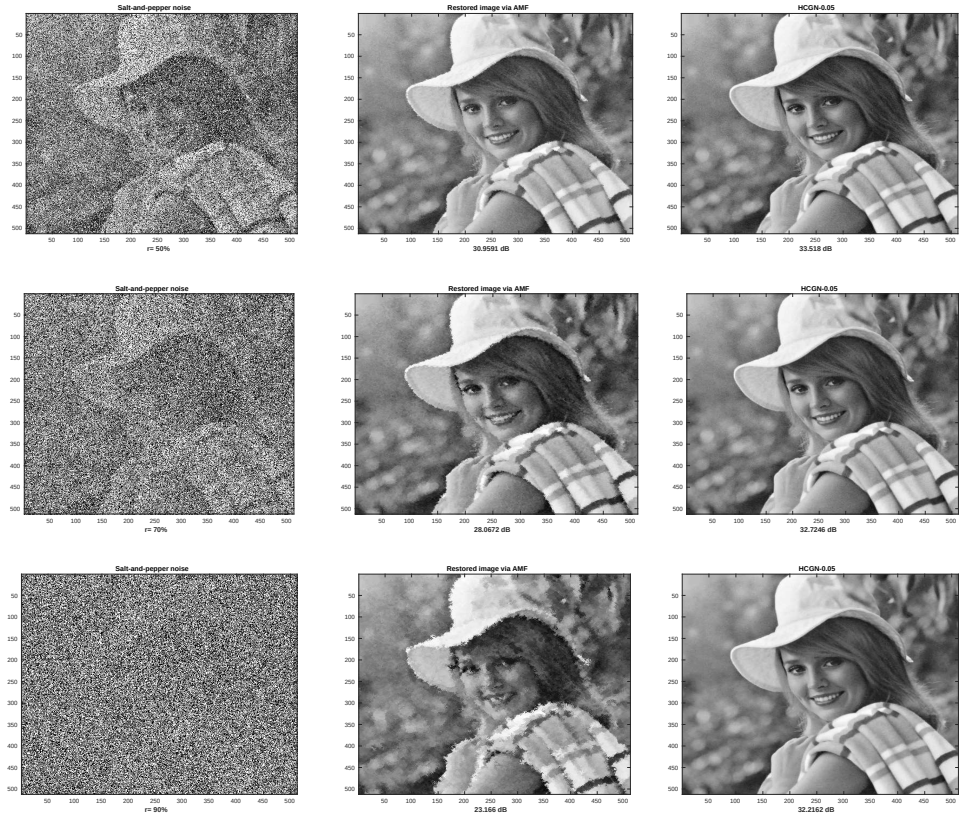

Fig. 12. The noisy images\the restored images via AMF\the restored images via HCGN-0.05 for 512×512 Elaine image.

[4] J. F. Cai, R. H. Chan, and B. Morini: Minimization of an edge-preserving regularization functional by conjugate gradient type methods, image processing based on partial differential equations. In: Mathematics and Visualization, Springer, Berlin Heidelberg 2007, pp. 109–122. DOI:10.1007/978-3-540-33267-1_7

[5] J. F. Cai, R. H. Chan, and M. Nikolova: Two-phase approach for deblurring images corrupted by impulse plus Gaussian noise. Inverse Problem and Imaging. *2* (2008), 187–204. DOI:10.3934/ipi.2008.2.187

[6] J. F. Cai, R. H. Chan, and M. Nikolova: Fast two-phase image deblurring under impulse noise. J. Math. Imaging and Vision *36* (2010), 46–53. DOI:10.1007/s10851-009-0169-7

[7] R. Chan, C. Hu, and M. Nikolova: Iterative procedure for removing random-valued impulse noise. IEEE Signal Process. Lett. *11* (2004), 12, 921–924. DOI:10.1109/lsp.2004.838190

[8] R. H. Chan, C. W. Ho, and M. Nikolova: Salt-and-pepper noise removal by median-type noise detectors and detail-preserving regularization. IEEE Trans. Image Process. *14* (2005), 1479–1485. DOI:10.1109/tip.2005.852196

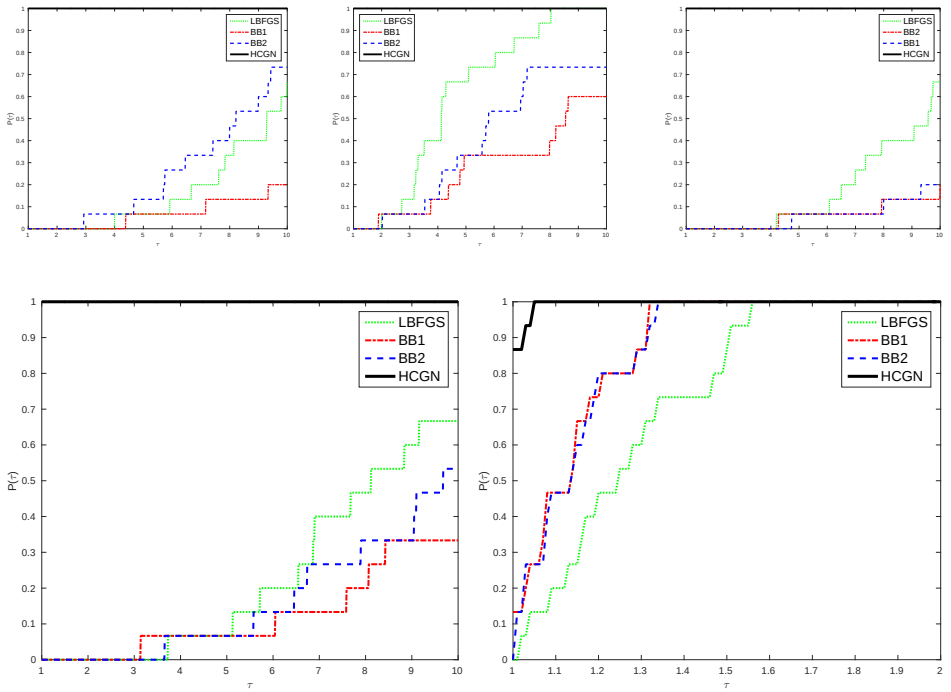

Fig. 13. Comparisons for LBFGS, BB1, BB2 and HCGN by performance profiles with the measures N_i, N_f, N_g, C_t and PSNR: (a) – (e) displays the performance profile for $\alpha = 0.05$.

[9] T. F. Chan, J. Shen, and H. Zhou: Total variation wavelet inpainting. J. Math. Imaging Vision *25* (2006), 107–125. DOI:10.1007/s10851-006-5257-3

[10] T. Chen and H. R. Wu: Adaptive impulse detection using center-weighted median filters. IEEE Signal Process. Lett. *8* (2001), 1–3. DOI:10.1109/97.889633

[11] Y. H. Dai and Q. Ni: Testing different conjugate gradient methods for large-scale unconstrained optimization. J. Comput. Math. *21* (2003), 311–320.

[12] Y. H. Dai and Y. Yuan: A nonlinear conjugate gradient method with a strong global convergence property. IEEE SIAM J. Optim. *10* (1999), 177–182. DOI:10.1137/s1052623497318992

[13] E. D. Dolan and J. J. Moré: Benchmarking optimization software with performance profiles. Math. Program. *91* (2002), 2, 201–213. DOI:10.1007/s101070100263

[14] R. Fletcher and C. Reeves: Function minimization by conjugate gradients. Comput. J. *7* (1964), 149–154. DOI:10.1093/comjnl/7.2.149

[15] J. C. Gilbert and J. Nocedal: Global convergence properties of conjugate gradient methods for optimization. SIAM J. Optim. *2* (1992), 21–42. DOI:10.1137/0802003

[16] W. W. Hager and H. Zhang: A new conjugate gradient method with guaranteed descent and an efficient line search. SIAM J. Optim. *16* (2005), 170–192. DOI:10.1137/030601880

[17] W. W. Hager and H. Zhang: A survey of nonlinear conjugate gradeint methods. http://www.math.u.edu/∼ hager, 2005.

[18] M. R. Hestenes and E. L. Stiefel: Methods of conjugate gradients for solving linear systems. J. Research Nat. Bur. Standards *49* (1952), 409–436. DOI:10.6028/jres.049.044

[19] H. Hwang and R. A. Haddad: Adaptive median filters: New algorithms and results. IEEE Trans. Image Process. *4* (1995), 499–502. DOI:10.1109/83.370679

[20] D. C. Liu and J. Nocedal: On the limited memory BFGS method for large scale optimization. Math. Program. *45* (1989), 503–528. DOI:10.1007/bf01589116

[21] M. Nikolova: A variational approach to remove outliers and impulse noise. J. Math. Imaging Vision *20* (2004), 1–2, 99–120. Special issue on mathematics and image analysis. DOI:10.1023/b:jmiv.0000011920.58935.9c

[22] J. Nocedal: Updating quasi-Newton matrices with limited storage. Math. Comput. *35* (1980), 773–782. DOI:10.1090/s0025-5718-1980-0572855-7

[23] J. Nocedal and S. J. Wright: Numerical Optimization. Springer, New York 2006. DOI:10.1007/978-0-387-40065-5

[24] B. T. Polyak: The conjugate gradient method in extreme problems. USSR Comp. Math. Math. Phys. *9* (1969), 94–112. DOI:10.1016/0041-5553(69)90035-4

[25] E. Polyak and G. Ribière: Note sur la convergence de directions conjugées. Francaise Informat Recherche Opertionelle, 3e Année *16* (1969), 35–43.

[26] M. J. D. Powell: Restart procedures of the conjugate gradient method. Math. Prog. *2* (1977), 241–254.

[27] M. J. D. Powell: Nonconvex minimization calculations and the conjugate gradient method. In: Numerical Analysis (Dundee, 1983), Lecture Notes in Mathematics, Springer-Verlag, Berlin *1066* (1984), pp. 122–141.

[28] G. Yua, J. Huanga, and Y. Zhou: A descent spectral conjugate gradient method for impulse noise removal. Appl. Math. Lett. *23* (2010), 555–560.

[29] G. Yu, L. Qi, Y. Sun, and Y. Zhou: Impulse noise removal by a nonmonotone adaptive gradient method. Signal Process. *90* (2010), 2891–2897.

[30] G. Zoutendijk: Nonlinear programming computational methods. In: Integer and Nonlinear Programming (J. Abadie, ed.), North-Holland, Amsterdam 1970, pp. 37–86.

Morteza Kimiaei, Department of Mathematics, Asadabad Branch, Islamic Azad University, Asadabad, Iran.
e-mail: morteza.kimiaei@gmail.com

Majid Rostami, Young Researchers and Elite Club, Hamedan Branch, Islamic Azad University, Hamedan, Iran.
e-mail: majid403rostami@yahoo.com