Applications and Applied Mathematics: An International Journal (AAM)

Volume 7 | Issue 1 Article 16

6-2012

A New CG-Algorithm with Self-Scaling VM-Update for Unconstraint Optimization

Abbas Y. Al-Bayati
University of Mosul

Ivan S. Latif
University of Salahaddin

Available at
http://pvamu.edu/aam
Appl. Appl. Math.
ISSN: 1932-9466

Applications and Applied Mathematics:
An International Journal
(AAM)

Vol. 7, Issue 1 (June 2012), pp. 226 – 247

A New CG-Algorithm with Self-Scaling VM-Update for Unconstraint Optimization

Abbas Y. Al-Bayati
College of Basic Education Telafer
University of Mosul, Mosul-Iraq
profabbaslbayati@yahoo.com

Ivan S. Latif
Department of Mathematics
University of Salahaddin, Erbil-Iraq
ivansubhi2001@yahoo.com

Received: February 09, 2011; Accepted: February 23, 2012

Abstract

In this paper, a new combined extended Conjugate-Gradient (CG) and Variable-Metric (VM) methods is proposed for solving unconstrained large-scale numerical optimization problems. The basic idea is to choose a combination of the current gradient and some pervious search directions as a new search direction updated by Al-Bayati's SCVM-method to fit a new step-size parameter using Armijo Inexact Line Searches (ILS). This method is based on the ILS and its numerical properties are discussed using different non-linear test functions with various dimensions. The global convergence property of the new algorithm is investigated under few weak conditions. Numerical experiments show that the new algorithm seems to converge faster and is superior to some other similar methods in many situations.

Keywords: Unconstrained Optimization, Gradient Related Method, Self-Scaling VM-Method, Inexact Line Searches

MSC 2010: 49M07, 49M10, 90C06, 65K

1. Introduction

Consider an unconstrained optimization problem:

$$\min f(x),\ x \in \Re^{n}\ , \tag{1}$$

where $f:\Re^{n}\to\Re^{1}$ is a continuously differentiable function in $\Re^{n}$ an n-dimensional Euclidean space; n may be very large in some sense. Most of the well-known iterative algorithms for solving (1) take the form:

$$x_{k+1}=x_k+\alpha_k d_k\ , \tag{2}$$

where d_k is a search direction and α_k is a positive step-size along the search direction. This class of methods is called a line search gradient method. If x_k is the current iterative point, then we denote $\nabla f(x_k)$ by g_k, $f(x_k)$ by f_k and $f(x^*)$ by f^*, respectively. If we take $d_k=-g_k$, then the corresponding method is called the Steepest Descent (SD) method; one of the simpler gradient methods. That has wide applications in large scale optimization; see Nocedal and Wright (1999). Generally the CG-method is a useful technique for solving large-scale nonlinear problems because it avoids the computation and storage of some matrices associated with the Hessian of objective functions. The CG-method has the form:

$$d_k=\begin{cases}-g_k, & \text{if}\quad k=0,\\ -g_k+\beta_{k-1}d_{k-1}, & \text{if}\quad k>0,\end{cases} \tag{3}$$

where β_k is a parameter that determines the different CG-methods; see for example the following references: Crowder & Wolfe (1972); Dai & Yuan (1996, 1999) and Fletcher-Reeves (1964). Well known choices of β_k satisfy:

$$\beta_k^{FR}=\frac{\|g_k\|^2}{\|g_{k-1}\|^2},\qquad \beta_{\mathrm{k}}^{\mathrm{PR}}=\frac{g_k^{\mathrm{T}}(g_k-g_{k-1})}{\|g_{k-1}\|^2}\ ,\qquad \beta_{\mathrm{k}}^{\mathrm{HS}}=\frac{g_k^{\mathrm{T}}(g_k-g_{k-1})}{d_{k-1}^{\mathrm{T}}g_{k-1}}, \tag{4}$$

which respectively, correspond to the FR (Fletcher-Reeves, 1964), PR (Polak-Ribiere, 1969) and HS (Hestenes Stiefel, 1952). CG-method with Exact Line Search (ELS) has finite convergence when they are used to minimize strictly convex quadratic function; see for example Al-Bayati and Al-Assady (1986). However, if the objective function is not quadratic or ELS is not used then a CG-method has no finite convergence. Also a CG-method has no global convergence if the objective function is non- quadratic. Similarly, Miele and Cantrell (1969) studied the memory gradient method for (1); namely, if x_0 is an initial point and $d_0=g_0$, the method can be stated as follows:

$$x_{k+1}=x_k+v_k\ ,\qquad v_{\mathrm{k}}=-\alpha g_k+\beta v_{k-1}\ , \tag{5}$$

where α and β are scalars chosen at each step so as to yield the greatest decrease in the function f .Cantrell (1969) showed that the memory gradient method and the FR-CG method are identical in the particular case of a quadratic function. Cragg and Levy (1969) proposed a super-memory gradient method whose search direction is defined by:

$$d_k = -\alpha g_k + \sum_{i=1}^{k} \beta_i d_{i-1} \ , \tag{6}$$

where x_0 is an initial point and $d_0 = g_0$. Wolfe and Viazminsky (1976) also investigated a super-memory descent method for (1) in which the objective takes the form:

$$f(x_k - \alpha_k p_k + \sum_{i=1}^{m} \beta_i^{(k)} v_{k-i}) = \min_{x,\beta_1,\ldots,\beta_k} f(x_k - \alpha\, p_k + \sum_{i=1}^{m} \beta_i^{(k)} v_{k-i}) \ , \tag{7a}$$

where

$$v_k = -\alpha_k p_k + \sum_{i=1}^{m} \beta_i^{(k)} \delta_{k-i} \, , \tag{7b}$$

m is a fixed positive integer; with

$$p_k^{\mathrm{T}} g_k \neq 0 \, . \tag{8}$$

Both the memory and super-memory gradient methods are more efficient than the CG and SD methods by considering the amount of computation and storages required in the latter. Shi-Shen (2004) combined the CG-method and supper-memory descent method to form a new gradient method that may be more effective than the standard CG-method for solving large scale optimization problems as follows:

$$\min\{g_k^{\mathrm{T}} d_k(\beta_{k-m-1}^{(k)},\ldots,1-\sum_{i=2}^{m}\beta_{k-i+1}^{(k)}) \,\Big|\, \beta_{k-i+1}^{(k)} \in [\frac{\delta_k^i}{2}, \delta_k^i](i=2,\cdots,m)\} \, , \tag{9a}$$

where

$$x_{k+1} = x_k + \alpha_k d_k(\beta_{k-m+1}^{(k)},\ldots,\beta_k^{(k)}) \, . \tag{9b}$$

We denote $d_k(\beta_{k-m+1}^{(k)},\ldots,\beta_k^{(k)})$ by d_k throughout this paper, m is a fixed positive integer and α_k is a scalar chosen by a line search procedure. The theoretical and practical merits of the Quasi Newton (QN) family of methods for unconstrained optimization have been systematically explored since the classic paper of Fletcher and Powell analyzed by Davidon' VM method. In 1970 the self-scaling VM algorithms were introduced, showing significant improvement in efficiency over earlier methods.

Recently, Al-Bayati and Latif (2008) proposed a new three terms preconditioned gradient memory method. Their method subsumes some other families of nonlinear preconditioned gradient memory methods as its subfamilies with Powell's restart criterion and inexact Armijo line searches. Their search direction was defined by:

$$d_k^{B\&L} = \begin{cases} -H_k g_k, & \text{if } k = 0 \\ -g_k + \beta H_k d_k - \alpha H_{k-1} d_{k-1}, & \text{if } k > 0, \end{cases} \tag{10}$$

where α is a step-size defined by inexact Armijo line search procedure and β is the conjugacy parameter. Al-Bayati et al. (2009) introduced two versions CG-algorithm. Their search directions are defined by:

$$d_k^1 = \begin{cases} -g_k, & \text{if } k = 0 \\ -g_k + \beta_k^{v1} d_{k-1}, & \text{if } k > 0, \end{cases} \text{ and } \beta_k^{v1} = (1 - \frac{s_k^T y_k}{y_k^T y_k}) \frac{(g_{k+1}^T y_k)}{s_k^T y_k} \tag{11a}$$

$$d_k^2 = \begin{cases} -g_k, & \text{if } k = 0 \\ -g_k + \beta_k^{v2} d_{k-1}, & \text{if } k > 0, \end{cases} \text{ and } \beta_k^{v2} = (1 - \frac{s_k^T y_k}{y_k^T y_k}) \frac{(g_{k+1}^T y_k)}{d_k^T y_k} + \frac{s_k^T g_{k+1}}{d_k^T y_k}, \tag{11b}$$

where (11) has been proved to be a sufficiently descent directions.

Also, Zhang, et al. in (2009) had modified Dai-Liao DL-CG method with three terms search directions as follows:

$$d_k = \begin{cases} -g_0, & \text{if } k = 0, \\ -g_k + \beta_k^{DL} d_{k-1} - \xi_k \left(y_{k-1} - t s_{k-1} \right), & \text{if } k > 0, \end{cases} \tag{12a}$$

where $\xi_k = g_k^T d_{k-1} / d_{k-1}^T y_{k-1}$ and β_k^{DL} is defined by:

$$\beta_k^{DL} = \frac{g_k^T (y_{k-1} - t s_{k-1})}{d_{k-1}^T y_{k-1}}, \quad t \geq 0. \tag{12b}$$

They show that the sufficient descent condition also holds true if no line search is used, that is,

$$g_{k-1}^T d_k = -\|g_{k-1}\|^2. \tag{13}$$

In order to achieve the global convergence result, Grippo and Lucidi (1997) proposed the following new line search: for given constants $\tau > 0$, $\delta > 0$, and $\lambda \in (0,1)$, let

$$\alpha_k = \max\left\{ \lambda^j \frac{\tau\left|g_k^T d_k\right|}{\left\|d_k\right\|^2}; j = 0, 1, ... \right\}, \tag{14}$$

which satisfy

$$f(x_k) \le f(x_{k-1}) - \delta\alpha_k^2 \left\|d_{k-1}\right\|^2. \tag{15}$$

This line search will be preferred to the classical Armijo one for the sake of a greater reduction of objective function. Introducing this line search rule. This may be taken as an open problem.

In this paper, a new gradient related algorithm combined with VM-update used for solving large scale unconstrained optimization problems, is proposed. The new algorithm is a kind of ILS method modified with VM-algorithm. The basic idea is to choose a combination of the current gradient and some previous search directions with Al-Bayati self-scaling VM-update which is based on two-parameter family of rank-two updating formulae. The algorithm is compared with similar published algorithms, which may be more effective than the standard conjugate related algorithm; namely, Nazareth (1977) and other VM-algorithm. The global rate of convergence is investigated under a diverse weak condition. Numerical experiment shows that the new algorithm seems to converge more stably and is superior to other similar methods.

2. Shi-Shen Algorithm

Shi-Shen (2004) proposed the following assumptions:

S_1**:** The objective function f has lower bound on the level set $L_0 = \{x \in \Re^n \left| f(x) \le f(x_0)\right\}$, where x_0 is an available initial point.

S_2 : The gradient $g(x)$ of $f(x)$ is Lipschitz continuous in an open convex set B which contains L_0 i.e., there exist a constant $L > 0$ such that:

$$\left\|g(x) - g(y)\right\| \le L\left\|x - y\right\|, \quad \forall \mathrm{x}, \mathrm{y} \in \mathrm{B}.$$

S_3 : The gradient $g(x)$ is uniformly continuous in an open convex set B containing L_0. Obviously Assumption (S_2) implies (S_3).

As we know, a key to devise an algorithm for unconstrained optimization problems is to choose an available search direction d_k and a suitable step-size α_k at each iteration. Certainly if we choose a search direction d_k satisfying:

$$-g_k^{\mathrm{T}} d_k < 0, \tag{16}$$

then we can devise a descent direction generally, we demand that:

$$-g_k^{\mathrm{T}} d_k \geq \eta \|g_k\|^2, \tag{17a}$$

which is called sufficient descent condition where $\eta > 0$.

Furthermore, if

$$-g_k^{\mathrm{T}} d_k \geq \eta \|g_k\| . \|d_k\|, \tag{17b}$$

then many descent algorithms have their convergence under the above condition. It is called an angle condition or a gradient–related conception.

Definition 2.1. Berstsekas (1982)

Let $\{x_k\}$ be a sequence generated by the gradient method (2). We say that the sequence $\{d_k\}$ is uniformly gradient related to $\{x_k\}$ if for every convergent subsequence $\{x_k\}$ for which

$$\lim_{k \in K \ \mathrm{k} \to \infty} g_k \neq 0 \ , \tag{18a}$$

we have

$$0 < \liminf_{k \in K \ \mathrm{k} \to \infty} \left| \mathrm{g_k^T} d_k \right| \quad , \limsup_{\mathrm{k} \in \mathrm{K} \ \mathrm{k} \to \infty} \|\mathrm{d_k}\| < +\infty . \tag{18b}$$

Equivalently, $\{d_k\}$ is uniformly gradient related if whenever a subsequence $\{g_k\}$ tends to a non-zero vector, the corresponding subsequence of direction d_k is bounded and does not tend to be orthogonal to g_k. Moreover, we must choose a line search rule to find the step-size a long search direction at each iteration.

Lemma 2.1. Berstsekas (1982)

Let $\{x_k\}$ be a sequence generated by a gradient method and assume that $\{d_k\}$ is uniformly gradient related and α_k is chosen by the minimization rule or the limited minimization rule, then every limited point of $\{x_k\}$ is a critical point x^*, i.e. $g(x^*) = 0$. As to the parameters in the algorithm, we seem to choose $\beta_{k-i+1}^{(k)} \in [\frac{\delta_k^i}{2}, \delta_k^i](i = 2,...,m)$ for solving large scale optimization problems as defined in (9). To get the algorithm to converge more quickly, we take:

$$\beta_{k-i+1}^{(k)} = \begin{cases} \dfrac{\delta_k^i}{2}, & \text{if} \quad \|g_k\|^2 \geq g_k^T d_{k-i+1}, \\ \delta_k^i, & \text{if} \quad \|g_k\|^2 < g_k^T d_{k-i+1}. \end{cases} \qquad \text{for } i = 1,2,...,m. \tag{19}$$

Now, it is easy to prove that:

$$0 < \sum_{i=2}^{m} \beta_{k-i+1}^{(k)} \leq \sum_{i=2}^{m} \delta_k^i \leq \rho < 1$$

and

$$1 > \beta_k^{(k)} = 1 - \sum_{i=2}^{m} \beta_{k-i+1}^{(k)} \geq 1 - \sum_{i=2}^{m} \delta_k^i \geq 1 - \rho > 0.$$

The details may be found in Crowder & Wolfe (1972).

Algorithm 2.1. Shi-Shen

Let $0 < \rho < 1$, $0 < \mu_1 < ½ < \mu_2 < 1$, a fixed integer $m \geq 2$, $x_1 \in \Re^n$, $k = 1$ and ε is a small parameter, then:

Step 1. If $\|g_k\| < \varepsilon$, then stop**.**

Step 2. Set $x_{k+1} = x_k + \alpha_k d_k(\beta_{k-m+1}^{(k)},...,\beta_k^{(k)})$, where

$$d_k(\beta_{k-m+1}^{(k)},...,\beta_k^{(k)}) = \begin{cases} -g_k, & \text{if} \quad k = m-1, \\ -\beta_k^{(k)} g_k - \sum_{i=2}^{m} \beta_{k-i+1}^{(k)} d_{k-i+1}, & \text{if} \quad k \geq m, \end{cases}$$

$$\beta_{k-i+1}^{(k)} \in [\frac{\delta_k^i}{2}, \delta_k^i] \quad , i = 2,...,m,$$

$$\beta_{k-i+1}^{(k)} = \begin{cases} \dfrac{\delta_k^i}{2}, & \text{if} \quad \|g_k\|^2 \geq g_k^T d_{k-i+1}, \\ \delta_k^i, & \text{if} \quad \|g_k\|^2 < g_k^T d_{k-i+1}, \end{cases} \qquad \text{for } i = 1,2,...,m,$$

and

$$\delta_{\mathrm{k}}^{\mathrm{i}} = \frac{\rho}{m - \dfrac{\|g_k\|^2}{\|g_k\|^2 + \left|g_k^{\mathrm{T}} d_{k-i+1}\right|}}, \quad (\mathrm{i} = 2,....,\mathrm{m}),\ \beta_{\mathrm{k}}^{(\mathrm{k})} = 1 - \sum_{i=2}^{m} \beta_{k-i+1}^{(k)},$$

scalar α_k in Step (2) is chosen by a cubic line search; Bunday (1984).

Step 3. If the available storage is exceeded, then employ a restart option [Zoutendijk (1970)] either with

$$k = n \text{ or } g_{k+1}^{\mathrm{T}} g_{k+1} > g_{k+1}^{\mathrm{T}} g_k .$$

Step 4. Set $k = k + 1$, go to Step1**.**

3. A New Proposed Algorithm for Solving Problem (1)

In this section we want to choose a line search rule to find the best step-size parameter along the search direction at each iteration. In fact, we can use the generalized Armijo' line search rule implemented in Luenberger (1989):

Set scalar S, β and μ_1 with $\mu_1 \in (0,1)$, $\beta \in (0,1)$ and $S > 0$. Let α_k be the largest α in $\{S,\ S\beta,\ S\beta^2,...\}$ such that:

$$f_k - f(x_k + \alpha d_k) \geq -\mu_1 \alpha g_k^{\mathrm{T}} d_k \ . \tag{20}$$

Choosing the parameter β is important for the implementation of the line search method. If β is too large then the line search process may be too slow while if β is too small then the line search process may be too fast so as to lose the available step size we should choose a suitable step size at each iteration.

4. New Method

In order to increase the efficiency of Algorithm 2.1, an extended Armijo line search rule given in Cantrell (1969) is used to find the best value of the step-size in order to locate the new hybrid line search which combines the search direction of Shi-Shen Algorithm 2.1 with Al-Bayati (1991) self-scaling update and as shown below:

Let $0 < \rho < 1$, $\frac{1}{2} < \mu_1 < 1$, a fixed integer $m \geq 2$, $\mathrm{x}_1 \in \Re^n$, $\mathrm{k} = 1$ and H_1 is any positive definite matrix usually $H_1 = I$ and ε a small parameter.

Algorithm 4.1.

Step 1. If $\|g_k\| = 0$, then stop.

Step 2. Set $x_{k+1} = x_k + \alpha_k d_k(\beta_{k-m+1}^{(k)},...,\beta_k^{(k)})$, where

$$d_k(\beta_{k-m+1}^{(k)},...,\beta_k^{(k)}) = \begin{cases} -Hg_k, & \text{if} \quad k \le m\text{-}1, \\ \text{-H}\{\beta_{\text{k}}^{(\text{k})} g_k - \sum_{i=2}^{m} \beta_{k-i+1}^{(k)} d_{k-i+1}\}, & \text{if} \quad k \ge m, \end{cases}$$

$$\beta_{k-i+1}^{(k)} \in [\frac{\delta_k^i}{2}, \delta_k^i], \qquad i = 2,\cdots,m,$$

$$\beta_{k-i+1}^{(k)} = \begin{cases} \dfrac{\delta_k^i}{2}, & \text{if} \quad \|\text{g}_{\text{k}}\|^2 \ge g_k^{\text{T}} d_{k-i+1}, \\ \delta_k^i, & \text{if} \quad \|\text{g}_{\text{k}}\|^2 < g_k^{\text{T}} d_{k-i+1}, \end{cases} \qquad \text{for } i = 1,2,...,m,$$

and

$$\delta_{\text{k}}^{\text{i}} = \frac{\rho}{m - \dfrac{\|g_k\|^2}{\|g_k\|^2 + |g_k^{\text{T}} d_{k-i+1}|}}, \quad (\text{i} = 2,....,\text{m}), \quad \beta_{\text{k}}^{(\text{k})} = 1 - \sum_{i=2}^{m} \beta_{k-i+1}^{(k)}.$$

Scalar α_k in Step (2) is chosen by Armijo line search rule defined in (20) and H_k is defined by Al-Bayati (1991) VM-update defined in Step (3).

Step 3. Update H_k by $H_{k+1} = \left(H_k - \dfrac{H_k y_k y_k^{\text{T}} H_k}{y_k^{\text{T}} H_k y_k} + w_k w_k^{\text{T}} \right) + \mu_k (v_k v_k^{\text{T}} / v_k^{\text{T}} y_k)$

with

$$v_k = x_{k+1} - x_k, \quad \text{y}_{\text{k}} = g_{k+1} - g_k,$$

$$w_k = (y_k^{\text{T}} H_k y_k)^{1/2} [v_k / v_k^{\text{T}} y_k - H_k y_k / y_k^{\text{T}} H_k y_k],$$

$$\mu_k = y_k^{\text{T}} H_k y_k / v_k^{\text{T}} y_k .$$

Step 4: If available storage is exceeded, then employ a restart option either with $k = n$ or

$$g_{k+1}^{\text{T}} g_{k+1} > g_{k+1}^{\text{T}} g_k .$$

Step 5. Set $k = k + 1$ and go to Step 2.

Now to ensure that the new algorithm has a super-linear convergence, let us consider the following theorems:

Theorem 4.1.

If (S_1) and (S_2) hold and if the new Algorithm 4.1 generates an infinite sequence $\{x_k\}$, then

$$\sum_{k=m}^{\infty} \frac{\|g_k\|^4}{\gamma_k} < +\infty, \tag{21a}$$

where

$$\gamma_k = \max_{2\le i\le m}(\|g_k\|^2, \|d_{k-i+1}\|^2). \tag{21b}$$

Proof:

Since $\{f_k\}$ is a decreasing sequence and has a lower bound on the level set L_0, it is a convergent sequence. Moreover, since H_{k+1} is also has a global rate of convergence, see Al-Bayati (1991) for the details; therefore Lemma 2.1 shows that (21) holds and hence the new algorithm has a super linear convergence and hence the proof is complete.

Theorem 4.2.

If conditions in Theorem 4.1 hold, then either $\lim_{k\to\infty} \|g_k\| = 0$ or $\{x_k\}$ has no bound.

Proof:

If $\lim_{k\to\infty} \|g_k\| \neq 0$, then there exists an infinite subset $K_0 \subset \{m, m+1,...\}$ and $\varepsilon > 0$ such that:

$$\|g_k\| > \varepsilon \quad , \mathrm{k} \in \mathrm{K}_0 \tag{22}$$

Thus

$$\frac{\varepsilon^4}{\gamma_k} \le \frac{\|g_k\|^4}{\gamma_k} \qquad \forall\ k \in K_0 \tag{23}$$

By Theorem 4.1 and for $k \ge 1$, we obtain

$$\|d_k\|^2 \le \max_{\le 1 i \le} \{\|g_i\|^2\} \tag{24}$$

Now if $k \le m$, then the conclusion is obvious. However, if $k > m$, then by induction process we obtain the conclusion; we have:

$$\sum_{k\in K_0}\frac{\varepsilon^4}{\gamma_k}\le\sum_{k=m}^{+\infty}\frac{\|g_k\|^4}{\gamma_k}<+\infty\ . \tag{25}$$

Then there exists at least one $i_0 : 2\le i_0\le m$ such that:

$$\lim_{k\in K_0,k\to\infty}\|d_{k-i+1}\|=+\infty \tag{26}$$

and hence $\{x_k\}$ has no bound.

5. Numerical Results.

Comparative tests are performed with seventy eight well-known test functions (Twenty six with three different versions) which are specified in the Appendix. All the results shown in Table 1 are obtained with newly-programmed Fortran routines which employ double precision. The Comparative performances of the algorithms are in the usual way by considering both the total number of function evaluations (NOF) and the total number of iterations (NOI). In each case the convergence criterion is that the value of $\|g_k\|<1\times10^{-5}$. The cubic fitting technique, published in its original form by Bunday (1984) is used as the common linear search subprogram for Shi-Shen algorithm while Armijo line search procedure defined in (21) is used for our new proposed algorithm.

Each test function was solved using the following two algorithms:

(1) The original algorithm published by Shi-Shen call it (**Shi-Shen, Algorithm 2.1**).

(2) The new proposed algorithm call it (**New algorithm, Algorithm 4.1**).

The numerical results in Table 1 give the comparison between the New and the Shi-Shen algorithms for different dimensions of test functions, while Table 2 gives the total overall the tools. The details of the percentage of improvements of NOI and NOF was given in Table 3. The important thing is that the new algorithm needs less iteration, fewer evaluations of $f(x)$ and $g(x)$ than the standard Shi-Shen algorithm in many situations especially for large-scale unconstrained optimization problems, when the iterative process reaches the same precision. The new proposed algorithm uses less CPU time than Shi-Shen method even we have not mention it. However, we can see that Shi-Shen algorithm may fail in some cases while the new method always converges for the minimum points. Moreover, the new algorithm seems to be suitable to solve ill-conditioned problems and suitable to solve large-scale minimization problems.

Table 1. Comparison between the **New (4.1) and Shi-Shen Algorithms (3.1)** using three different cases for n and m.

N0. OF TEST	TEST FUNCTION		SHI-SHEN (3.1) NOI(NOF)					NEW (4.1) NOI(NOF)				
		n	12	36	360	1080	4320	12	36	360	1080	4320
1	EX-Tridiagonal-1	$m<n$	57 142	54 118	58 118	58 118	59 120	16 31	16 19	16 20	15 19	16 20
		$m=n$	53 113	54 112	54 112	58 118	58 118	16 28	16 19	16 20	15 19	16 20
		$m>n$	53 109	54 111	58 118	58 118	58 118	15 19	16 19	16 20	15 19	16 20
2	EX-Three exponential	$m<n$	26 61	22 59	22 45	22 45	21 43	4 7	5 8	5 8	5 8	5 8
		$m=n$	26 55	21 43	22 45	22 45	21 43	4 7	5 8	5 8	5 8	5 8
		$m>n$	26 53	21 43	22 45	22 45	21 43	4 7	5 8	5 8	5 8	5 8
3	Matrix Rom	$m<n$	2 7	2 7	2 7	2 7	2 7	3 5	4 6	4 6	4 6	4 6
		$m=n$	2 7	2 7	2 7	2 7	2 7	3 5	4 6	4 6	4 6	4 6
		$m>n$	2 7	2 7	2 7	2 7	2 7	3 5	4 6	4 6	4 6	4 6
4	EX-Freud & Roth	$m<n$	6645 13735	311 707	1940 5341	1956 5745	2040 6086	20 26	21 27	21 27	21 27	21 27
		$m=n$	512 1455	9265 18877	10504 21132	10882 21869	11324 22743	20 26	21 27	21 27	21 27	21 27
		$m>n$	9440 718975	9792 19697	10528 21151	10880 21855	11324 22743	20 26	21 27	21 27	21 27	21 27
5	GEN-Tridiagonal-1	$m<n$	58 145	71 230	58 127	58 139	52 135	16 39	17 29	16 20	15 19	16 20
		$m=n$	58 123	56 115	58 127	58 127	52 135	16 28	16 19	16 20	15 19	16 20
		$m>n$	48 99	26 75	58 127	58 127	52 135	15 19	16 19	16 20	15 19	16 20
6	Diagonal4	$m<n$	7 15	7 15	9 19	9 19	9 19	8 12	8 12	8 12	8 12	8 12
		$m=n$	7 15	7 15	9 19	9 19	9 19	8 12	8 12	8 12	8 12	8 12
		$m>n$	7 15	7 15	9 19	9 19	9 19	8 12	8 12	8 12	8 12	8 12
7	Dqdrtic	$m<n$	1260 3036	1371 3807	925 2509	1025 2987	1305 3827	20 25	16 21	12 17	12 17	13 18
		$m=n$	1260 2626	1371 2781	925 1853	1025 2051	1305 2611	20 25	16 21	12 17	12 17	13 18
		$m>n$	2521 1260	1371 2743	925 1851	1025 2051	1305 2611	20 25	16 21	12 17	12 17	13 18
8	Denschnb	$m<n$	11 28	12 31	13 29	13 29	14 31	8 20	9 18	8 11	8 11	8 11
		$m=n$	11 25	12 27	13 29	13 29	14 31	7 10	8 11	8 11	8 11	8 11
		$m>n$	11 25	12 27	13 29	13 29	14 31	7 10	8 11	8 11	8 11	8 11

N0. OF TEST	TEST FUNCTION		SHI-SHEN (3.1) NOI(NOF)					NEW (4.1) NOI(NOF)				
		n	12	36	360	1080	4320	12	36	360	1080	4320
9	GEN-Quartic GQ1	*m*<*n*	11 27	11 27	11 24	11 24	11 24	9 12	12 15	14 17	14 17	14 17
		m=*n*	11 24	11 27	11 24	11 24	11 24	9 12	12 15	14 17	14 17	14 17
		m>*n*	11 24	11 27	11 24	11 24	11 24	9 12	12 15	14 17	14 17	14 17
10	Diagonal 7	*m*<*n*	4 10	4 10	4 10	4 10	4 10	4 7	4 7	5 8	5 8	5 8
		m=*n*	4 10	4 10	4 10	4 10	4 10	4 7	4 7	5 8	5 8	5 8
		m>*n*	4 10	4 10	4 10	4 10	4 10	4 7	4 7	5 8	5 8	5 8
11	Full Hessian	*m*<*n*	25 76	31 160	17 78	15 75	21 121	4 8	3 7	3 8	3 9	3 9
		m=*n*	25 68	31 93	17 78	15 75	21 121	4 8	3 7	3 8	3 9	3 9
		m>*n*	25 64	31 93	17 78	15 75	21 121	4 8	3 7	3 8	3 9	3 9
12	GEN-Powell	*m*<*n*	85 208	88 252	104 307	119 239	125 251	50 56	50 56	50 56	50 56	50 56
		m=*n*	109 229	107 217	115 231	119 239	125 521	50 56	50 56	50 56	50 56	50 56
		m>*n*	101 203	107 215	115 231	119 239	125 251	50 56	50 56	50 56	50 56	50 56
13	GEN-Rosen	*m*<*n*	48 117	37 130	39 138	39 138	41 145	52 66	53 67	54 68	54 68	54 68
		m=*n*	830 1757	864 1779	954 1937	998 2023	1052 2131	52 66	53 67	54 68	54 68	54 68
		m>*n*	820 1667	864 1755	954 1935	998 2023	1052 2131	52 66	53 67	54 68	54 68	54 68
14	Non-Diagonal	*m*<*n*	23 63	44 138	73 222	23 63	23 63	11 15	11 15	11 15	11 15	11 15
		m=*n*	469 1024	129 302	373 803	385 826	385 826	11 15	11 15	11 15	11 15	11 15
		m>*n*	407 902	359 777	373 802	385 826	385 826	11 15	11 15	11 15	11 15	11 15
15	GEN-Wolf	*m*<*n*	102 440	197 544	206 530	212 540	216 560	14 17	16 26	15 18	15 18	16 19
		m=*n*	182 388	197 400	206 413	212 425	216 433	14 17	16 26	15 18	15 18	16 19
		m>*n*	102 365	197 395	206 413	212 425	216 433	14 17	16 26	15 18	15 18	16 19
16	GEN-Strait	*m*<*n*	9 20	9 20	11 23	11 23	11 23	10 15	10 15	10 15	10 15	10 15
		m=*n*	9 19	9 19	11 23	11 23	11 23	10 15	10 15	10 15	10 15	10 15
		m>*n*	9 19	9 19	11 23	11 23	11 23	10 15	10 15	10 15	10 15	10 15

N0. OF TEST	TEST FUNCTION		SHI-SHEN (3.1) NOI(NOF)					NEW (4.1) NOI(NOF)				
		n	12	36	360	1080	4320	12	36	360	1080	4320
17	GEN-Recipe	*m*<*n*	69 204	73 267	81 203	85 213	89 223	11 14	11 14	12 15	12 15	13 16
		m=*n*	69 183	73 187	81 203	85 213	89 223	11 14	11 14	12 15	12 15	13 16
		m>*n*	69 173	73 183	81 203	85 213	89 223	11 14	11 14	12 15	12 15	13 16
18	Non-diagonal (Shanno-78)	*m*<*n*	30 71	10 24	102 219	292 789	946 2751	34 44	13 19	17 24	17 25	17 25
		m=*n*	30 65	10 22	102 206	292 586	946 2751	34 44	13 19	17 24	17 25	17 25
		m>*n*	30 65	10 22	102 206	292 586	946 2751	34 44	13 19	17 24	17 25	17 25
19	EX-Tridigonal-2	*m*<*n*	83 194	102 203	102 218	103 221	99 212	13 26	19 22	19 22	19 22	19 22
		m=*n*	83 173	102 207	102 205	105 211	98 198	13 25	19 22	19 22	19 22	19 22
		m>*n*	83 167	102 205	102 205	105 211	98 198	13 16	19 22	19 22	19 22	19 22
20	GEN-Beale	*m*<*n*	656 1597	586 1625	509 1350	535 1517	565 1607	20 23	20 23	22 25	22 25	22 25
		m=*n*	503 1049	459 931	509 1020	535 1071	565 1131	20 23	20 23	22 25	22 25	22 25
		m>*n*	433 867	457 915	509 1019	535 1071	565 1131	20 23	20 23	22 25	22 25	22 25
21	EX-Block-Diagonal BD2	*m*<*n*	16 39	18 48	18 38	20 42	20 42	37 120	69 79	72 76	75 79	78 82
		m=*n*	16 35	18 48	18 38	20 42	20 42	90 145	69 79	72 76	75 79	78 82
		m>*n*	16 34	18 38	18 38	20 42	20 42	64 68	66 70	72 76	75 79	78 82
22	Diagonal 7	*m*<*n*	4 11	5 13	5 13	5 13	5 13	8 23	7 12	8 13	8 13	8 13
		m=*n*	4 11	5 13	5 13	5 13	5 13	7 12	7 12	8 13	8 13	8 13
		m>*n*	4 11	5 13	5 13	5 13	5 13	7 12	7 12	8 13	8 13	8 13
23	Cosine (cute)	*m*<*n*	9 21	10 24	10 27	11 28	13 31	22 46	13 25	22 27	75 87	12 16
		m=*n*	9 20	10 22	10 27	11 28	13 31	20 26	13 17	22 27	75 87	12 16
		m>*n*	9 20	10 22	10 27	11 28	13 31	20 26	13 17	22 27	75 87	12 16
24	EX-Himmelblau	*m*<*n*	3 7	4 9	4 9	4 9	4 9	8 17	9 18	9 18	9 18	9 18
		m=*n*	3 7	4 9	4 9	4 9	4 9	8 17	9 18	9 18	9 18	9 18
		m>*n*	3 7	4 9	4 9	4 9	4 9	8 17	9 18	9 18	9 18	9 18

N0. OF TEST	TEST FUNCTION		SHI-SHEN (3.1) NOI(NOF)					NEW (4.1) NOI(NOF)				
		n	12	36	360	1080	4320	12	36	360	1080	4320
25	Raydan2	*m<n*	2 5	2 5	2 5	2 5	2 5	5 7	5 7	6 8	6 8	6 8
		m=n	2 5	2 5	2 5	2 5	2 5	5 7	5 7	6 8	6 8	6 8
		m>n	2 5	2 5	2 5	2 5	2 5	5 7	5 7	6 8	6 8	6 8
26	Diagonal6	m<n	2 7	2 7	2 7	2 7	2 7	5 7	5 7	6 8	6 8	6 8
		m=n	2 7	2 7	2 7	2 7	2 7	5 7	5 7	6 8	6 8	6 8
		m>n	2 7	2 7	2 7	2 7	2 7	5 7	5 7	6 8	6 8	6 8

Table 2. Comparison between the **New (4.1) and Shi-Shen (3.1) algorithms** using the total of tools for each test function.

	n	SHI-SHEN NOI(NOF)					NEW NOI(NOF)				
		12	36	360	1080	4320	12	36	360	1080	4320
Total of each function	m<n	9243 20286	3083 8480	4327 11616	4636 13045	5699 16365	412 686	426 563	447 573	499 626	444 562
	m=n	4289 9493	12825 26275	14113 28576	14885 30095	16354 34206	461 657	426 549	447 562	499 625	444 562
	m>n	14238 725153	13550 27428	14141 28595	14883 30081	16354 33936	433 553	423 540	447 562	499 633	444 562
Total of 26x3=78 test functions		27770 754932	29458 62183	32581 68787	34404 73221	38407 84507	1306 1896	1275 1663	1341 1686	1497 1875	1332 1686

Table 3. Percentage performance of the new (4.1) proposed algorithm against Shi-Shen (3.1) algorithm for 100% in both NOI and NOF.

n	Cost		NEW
12	NOI NOF	*m*<*n*	95.5 96.62
	NOI NOF	*m*=*n*	89.25 93.08
	NOI NOF	*m*>*n*	96.96 99.92
	NOI NOF	**Total**	95.3 99.75
36	NOI NOF	*m*<*n*	86.18 93.36
	NOI NOF	*m*=*n*	96.68 97.91
	NOI NOF	*m*>*n*	96.88 98.03
	NOI NOF	**Total**	95.67 97.33
360	NOI NOF	*m*<*n*	89.67 95.07
	NOI NOF	*m*=*n*	96.83 98.03
	NOI NOF	*m*>*n*	96.83 98.03
	NOI NOF	**Total**	95.88 97.55

n	Cost		NEW
1080	NOI NOF	m<n	89.24 95.2
	NOI NOF	m=n	96.65 97.92
	NOI NOF	m>n	96.65 97.89
	NOI NOF	**Total**	95.65 97.44
4320	NOI NOF	m<n	92.21 96.57
	NOI NOF	m=n	97.29 98.36
	NOI NOF	m>n	97.29 98.34
	NOI NOF	**Total**	96.53 98.01

5. Conclusions and Discussions.

In this Paper, a new combined gradient related and VM-algorithm for solving large-scale unconstrained optimization problems is proposed. The new algorithm is a kind of Armijo line search method. The basic idea is to choose a combination of the current gradient and some previous search directions which are updated by Al-Bayati's self scaling (1991) VM as a new search direction and to find a step-size by using Armijo ILS. Using more information at the current iterative step may improve the performance of the algorithm and accelerate the gradient relates which need a few iterations. The new algorithm concept is useful to analyze its global convergence property. Numerical experiments show that the new algorithm converges faster and is more efficient than the standard Shi-Shen algorithm in many situations. The new algorithm is expected to solve ill-conditioned problems. Clearly there are large ranges of the improving percentages against the standard Shi-Shen algorithm; namely, the new algorithm has about (53)% NOI and (75)%NOF improvements against Shi-Shen algorithm taking n=12. These

improvements are very clear for n=36; n = 360, 1080 and finally, for n = 4320 the new algorithm has about (64)% NOI and (80)% NOF improvements against Shi-Shen (2004) CG-algorithm.

REFERENCES

Al-Bayati, A. (1991). A new family of self-scaling VM-algorithms, Journal of Education and Science, Mosul University, Iraq, **12**, 25-54.

Al-Bayati, A. and Al-Assady, N. (1986). Conjugate Gradient Methods, Technical Research Report, No. (1/86), School of Computer Studies, Leeds University, U.K.

Al-Bayati, A. Y. and Latif, I. S. (2008). A new three-term preconditioned gradient memory algorithm for nonlinear optimization problems, American Journal of Applied Science, Science Publication, New York, **4**(2), 81-87.

Al-Bayati A. Y., Salim, A. J. and Abbo, K.K. (2009). Two versions of CG-algorithms based on conjugacy conditions for unconstrained optimization, American Journal of Economic and Business administration, Science Publication, USA**, 1**(2), 97-104.

Berstsekas, D. (1982). Constrained Optimization and Lagrange Multiplier Methods, Academic Press, New York, USA.

Bunday, B. (1984). Basic Optimization Methods, Edward Arnold. Bedford square, London, U.K.

Cantrell, J. (1969). On the relation between the memory gradient and the Fletcher-Reeves method, JOTA, **4**(1), 67-71.

Crowder, H. and Wolfe, P. (1972). Liner convergence of the conjugate gradient methods, IBM Journal of Research and Development, **16**, 431-433.

Cragg, E. and Levy, A. (1969). Study on a super memory gradient method for the minimization of function, JOTA, **4**(3), 191-205.

Dai Y. and Yuan, Y. (1999). Nonlinear Conjugate Gradient Methods, Shanghai Science-17 and Technology press, Shanghai.

Dai, Y. and Yuan, Y. (1996). Convergence properties of the Fletcher-Reeves method, IMA. J. of Numerical Analysis, **16**, 155-164.

Fletcher, R. and Reeves, C. (1964). Function minimization by conjugate gradients, J. Computer, **7**, 149-154.

L. Grippo, L. and Lucidi, S. (1997). A globally convergent version of the Polak-Ribiere conjugate gradient method, Mathematical Programming, **78** (3), 375–391.

Hestenes, M. and Stiefel, E. (1952). Methods of conjugate gradients for solving linear Systems, J. Res. Nat Bureau stand., **29**, 409-430.

Luenberger, D. (1989). Linear and Nonlinear Programming, 2nd Edition. Addition-Wesley, Reading, U.K.

Miele, A. and Cantrell, J. (1969). Study on a memory gradient method for the minimization of function, JOTA, **3**(6), 459-470.

Nazareth, L. (1977). A Conjugate direction algorithm for unconstrained minimization without line searches, JOTA, **23**, 373-387.

Nocedal, J. (1980). Updating quasi-Newton matrices with limited storage, Mathematics of Computation, **35**, 773-782.

Nocedal, J. and Wright, S. (1999). Numerical Optimization, Spring Series in Operations Research, Springer Verlag, New York, USA.

Nocedal, J. (2005). Unconstrained Optimization Test Functions. Research Institute for Informatics, Center for advanced Modeling and Optimization, Bucharest1, Romania.

Polak, E. and Ribiere, G. (1969). Note sur la Convergence des methods de direction conjuguees. revue ferue francaised, Inform. Rec. Oper., **16**, 35-43.

Shi, Z. and Shen, J. (2004). A gradient-related algorithm with inexact line searcher, J. Computational and Applied Mathematics, **170**, 349-370.

Wolfe, M. and Viazminsky, C. (1976). Super memory descent methods for unconstrained minimization, JOTA., **18**(4), 455-468.

Zhang J., Xiao, Y. and Wei, Z. (2009). Nonlinear conjugate gradient methods with sufficient descent condition for large scale unconstrained optimization, Mathematical Problems in Engineering, DOI: 1155/2009/243290.

Zoutendijk, G. (1970). Nonlinear Programming: Computational Methods in Integer and Nonlinear Programming,, J. Abadie. ed., North-Holland, Amsterdam.

APPENDIX

All the test functions used in this paper are from general literature Nocedal (1980 , 2005).

1. Extended Tridigonal-1Function:

$$f(x) = \sum_{i=1}^{n/2} (x_{2i-1} + x_{2i} - 3)^2 + (x_{2i-1} - x_{2i} + 1)^4 ,$$

$$x_0 = [2,2,...,2].$$

2. Extended Three Exponential Terms Function:

$$f(x) = \sum_{i=1}^{n/2} \left(\exp(x_{2i-1} + 3x_{2i} - 0.1) + \exp(x_{2i-1} - 3x_{2i} - 0.1) + \exp(-x_{2i-1} - 0.1)\right),$$

$$x_0 = [0.1,0.1,...,0.1].$$

3. Diagonal 5 Function (Matrix Rom):

$$f(x) = \sum_{i=1}^{n} \log(\exp(x_i) + \exp(-x_i)),$$

$$x_0 = [1.1,...,1.1] .$$

4. Extended Freud & Roth Function:

$$f(x)=\sum_{i=1}^{n/2}\left(-13+x_{2i-1}+((5-x_{2i})x_{2i}-2)x_{2i}\right)^2+\left(-29+x_{2i-1}+((x_{2i}+1)x_{2i}-14)x_{2i}\right)^2,$$

$$x_0=[0.5.,-2.,0.5,-2.,...,0.5,-2.].$$

5. Generalized Tridiagonal-1 Function:

$$f(x)=\sum_{i=1}^{n-1}\left(x_{2i-1}+x_{2i}-3\right)^2+\left(x_{2i-1}-x_{2i}+1\right)^4,$$

$$x_0=[2,2,...,2].$$

6. Diagonal 4 Function:

$$f(x)=\sum_{i=1}^{n/2}\tfrac{1}{2}\left(x_{2i-1}^2+cx_{2i}^2\right),$$

$$x_0=[1,1,...,1]\ ,\ \mathrm{c}=100.$$

7. Dqudrtic Function (CUTE):

$$f(x)=\sum_{i=1}^{n-2}\left(x_i^2+cx_{i+1}^2+dx_{i+2}^2\right),$$

$$x_0=[3,3,...,3]\ ,\ \mathrm{c}=100,\mathrm{d}=100.$$

8. Extended Denschnb Function (CUTE):

$$f(x)=\sum_{i=1}^{n/2}(x_{2i-1}-2)^2+(x_{2i-1}-2)^2x_{2i}^2+(x_{2i}+1)^2,$$

$$x_0=[0.1,0.1,...,0.1].$$

9. Generalized quartic Function GQ1

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

$$x_0=[1.,1.,...,1.].$$

10. Diagonal 8 Function:

$$f(x) = \sum_{i=1}^{n} x_i \exp(x_i) - 2x_i - x_i^2 \ ,$$

$$x_0 = [1.,1.,...,1.,1.].$$

11. Full Hessian Function:

$$f(x) = \left(\sum_{i=1}^{n} x_i \right)^2 + \sum_{i=1}^{n} (x_i \exp(x_i) - 2x_i - x_i^2) \, ,$$

$$x_0 = [1.,1.,...,1.].$$

12. Generalized Powell function:

$$f(x) = \sum_{i=1}^{n/3} \{3 - [\tfrac{1}{1+(x_i - x_{2i})^2}] - \sin(\tfrac{\pi x_{2i} x_{3i}}{2}) - \exp[-(\tfrac{x_1 + x_{3i}}{x_{2i}} - 2)^2]\} \, ,$$

$$x_0 = [0.,1.,2.,...,0.,1.,2.] \, .$$

13. Generalized Rosen Brock Banana function:

$$f(x) = \sum_{i=1}^{n/2} 100(x_{2i} - x_{2i-1}^2)^2 + (1 - x_{2i-1})^2 \ ,$$

$$x_0 = [-1.2,1.,...,-1.2,1]$$

14. Generalized Non-diagonal function:

$$f(x) = \sum_{i=2}^{n} [100(x_1 - x_i^2)^2 + (1 - x_i)^2 \ ,$$

$$x_0 = [-1.,...,-1.].$$

15. Generalized Wolfe Function:

$$f(x) = (-x_1(3 - x_1/2) + 2x_2 - 1)^2 + \sum_{i=1}^{n-1} \left(x_{i-1} - x_i(3 - x_i/2 + 2x_{i+1} - 1)\right)^2 + \left(x_{n-1} - x_n(3 - x_n/2) - 1\right)^2 ,$$

$$x_0 = [-1.,...,-1.].$$

16. Generalized Strait Function:

$$f(x) = \sum_{i=1}^{n/2} (x_{2i-1}^2 - x_{2i})^2 + 100(1 - x_{2i-1})^2 ,$$

$$x_0 = [-2.,...,-2.].$$

17. Generalized Recipe Function:

$$f(x) = \sum_{i=1}^{n/3} \left[(x_{3i-1} - 5)^2 + x_{9i-1}^2 + \frac{x_{3i}^2}{(x_{3i-1} - x_{3i} - 2)^2} \right],$$

$$x_0 = [2.,5.,1.,...,2.,5.,1.].$$

18. Non-diagonal (Shanno-78) Function (Cute):

$$f(x) = (x_i - 1)^2 + \sum_{i=2}^{n} 100(x_1 - x_{i-1}^2)^2 ,$$

$$x_0 = [-1.,-1.,...,-1.].$$

19. Extended Tridiagonal-2 Function:

$$f(x) = \sum_{i=1}^{n-1} (x_i x_{i+1} - 1)^2 + c(x_i + 1)(x_{i+1} + 1),$$

$$x_0 = [1.,1.,...,1.] \ , \mathrm{c} = 0.1.$$

20. Generalized Beale Function:

$$f(x) = \sum_{i=1}^{n/2} [1.5 - x_{2i} + (1 - x_{2i})]^2 + [2.25 - x_{2i-1}(1 - x_{2i}^2)]^2 + [2.625 - x_{2i-1}(1 - x_{2i}^2]^2 ,$$

$$x_0 = [-1.,-1.,...,-1.,-1.].$$

21. Extended Block-Diagonal BD2 Function:

$$f(x) = \sum_{i=1}^{n/2} (x_{2i-1}^2 + x_{2i}^2 - 2.)^2 + (\exp(x_{2i-1} - 1.) + x_{2i}^3 - 2.)^2 ,$$

$$x_0 = [1.5,2.,...,1.5,2.].$$

22. Diagonal 7 Function:

$$f(x) = \sum_{i=1}^{n} (\exp(x_i) - 2x_i - x_i^2),$$

$$x_0 = [1.,1.,...,1.,1.].$$

23. Cosine Function (CUTE):

$$f(x) = \sum_{i=1}^{n-1} \cos(-0.5x_{i+1} + x_i^2),$$

$$x_0 = [1.,1.,...,1.,1.].$$

24. Extended Himmelblau Function:

$$f(x) = \sum_{i=1}^{n/2} \left(x_{2i-1}^2 + x_{2i} - 11\right)^2 + \left(x_{2i-1} + x_{2i}^2 - 7\right)^2,$$

$$x_0 = [1.1,1.1,...,1.1,1.1].$$

25. Raydan 2 Function:

$$f(x) = \sum_{i=1}^{n} \left(\exp(x_i) - x_i\right),$$

$$x_0 = [1.,1.,...,1.,1.].$$

26. Diagonal 6 Function:

$$f(x) = \sum_{i=1}^{n} (\exp(x_i) - (1 + x_i)),$$

$$x_0 = [1.,1.,...,1.,1.].$$