Investigation on Extended Conjugate Gradient Non-linear methods for solving Unconstrained Optimization

Khalil K. Abbo                          Osama M. T. Waiss
Department Of Mathematic
College of Computer Science and Mathematics
University of Mosul

Received 07 / 04 / 2011
Accepted 10 / 07 / 2011

Abstract
In this paper we have generalized the extended Dai-Yuan conjugate Gradient method by Considering the parameter $\tau_{k+1}$ in the denominator of $\beta_{k+1}$ as a convex combination. Three values of $\tau_{k+1}$ are computed in three different ways namely by assuming descent property, Pure Conjugacy and using Newton direction.

The descent property and global convergence for the proposed algorithms are established. Our numerical experiments on some standard test functions show that there are considerable improvement on other classical methods in this field.

1- Introduction:
Consider the unconstrained optimization problem defined by:
\[
\text{Min} \ f(x), \quad x \in \mathbb{R}^n
\]
Where $f : \mathbb{R}^n \rightarrow \mathbb{R}$ is continuously differentiable. The line search algorithm for (1) often generates a sequence of iterates $\{x_k\}$ by letting
\begin{equation}
    x_{k+1} = x_k + \alpha_k d_k \quad k = 0, 1, 2, \ldots
\end{equation}

where \( x_k \) is the current iterate point, \( d_k \) is a descent search direction i.e. \( g_k^T d_k < 0 \) and \( \alpha_k > 0 \) is a step length.

Different choices of \( d_k \) and \( \alpha_k \) will determine different line search methods [8-10]. These methods are divided into two stages at each iteration:

a) Choose a descent search direction \( d_k \).

b) Choose a step-size \( \alpha_k \) along the search direction \( d_k \).

Throughout this paper, we denote \( f(x_k) \) by \( f_k \), \( \nabla f(x_k) \) by \( g_k \), and \( \nabla f(x_{k+1}) \) by \( g_{k+1} \) respectively. \( \| \| \) denotes the Euclidian norm of vectors.

One simple line search method is the steepest descent method if we take \( d_k = -g_k \) as a search direction at every iteration, which has wide applications in solving large-scale minimization problems [11]. However, the steepest descent method often yields zig-zag phenomena in solving practical problems. Which makes the algorithm converge to an optimal solution very slowly or even fail to converge [6]. then the steepest descent (SD) not recommended for practical use.

If \( d_k = -H_k g_k \) is the search direction at each iteration in the algorithm, Where \( H_k \) is an \( n \times n \) matrix approximation to the \( [\nabla^2 f_k]^{-1} = G_k \), then the corresponding line search method is called Newton like method such as quasi-Newton or variable metric etc, on the other hand if \( H_k = G_k \) the method is called Newton method, Which is one of the more the successful algorithm for unconstrained optimization if \( G_k^{-1} \) is symmetric and positive definite and satisfies quasi-Newton condition given by

\begin{equation}
    G_{k+1}^{-1} y_k = s_k
\end{equation}

Where \( y_k = g_{k+1} - g_k \) and \( s_k = x_{k+1} - x_k \)

For the general non-liner objective function the convergence of the Newton Algorithm to a solution cannot be guaranteed from an arbitrary initial point \( x_k \). In general if initial point is not sufficiently close to the solution then the algorithm may not posses the descent property, the other drawback of the Newton or quasi-Newton method is required to store and compute matrix \( H_k \) at each iteration and these adds cost of storage and computation. Accordingly these methods is not suitable to solve large scale optimization problems in many cases [7].

The conjugate gradient method is very useful for solving (1) especially when \( n \) is large and has the following form:

\begin{equation}
    d_k = -g_k \quad k = 1
\end{equation}

\begin{equation}
    d_{k+1} = -g_{k+1} + \beta_k d_k
\end{equation}

Where \( \beta_k \) is a parameter, in the case when \( f \) is a convex quadratic function and \( \alpha_k = \text{arg min}_{\alpha \geq 0} f(x_k + \alpha d_k) \)
The conjugate gradient method is such that the conjugacy condition holds [2], namely
\[ d_i^T H d_j = 0 \quad \forall i \neq j \]  
(5)
For general non-liner function Dai and Liao in [2] replaced the conjugacy condition (5) to the following form
\[ d_{k+1}^T y_k = 0 \]  
(6)
Which is called pure conjugacy conditions additionally if inexact line search is used also see [2], the condition in (6) can be written as:
\[ d_{k+1}^T = -t g_{k+1} s_k \]  
(7)
When \( t \geq 0 \) is scalar.

Several kinds of formulas for \( \beta_k \) has been proposed. For example Fletcher–Reeves (FR), Polak–Ribiere (PR). and Hestenes–Stiegel (HS). formulas are well Known and they are given by:
\[ \beta_k^{FR} = \frac{g_k^T s_k}{g_k^T d_k} \]  
(8)
\[ \beta_k^{PR} = \frac{g_k^T s_k}{g_{k-1}^T s_{k-1}} \]  
(9)
\[ \beta_k^{HS} = \frac{g_k^T s_k}{d_k^T s_k} \]  
(10)
The global convergence properties of the FR, PR and HS methods without regular restarts have been studied by many researchers [1-3]. The conjugate gradient methods with regular restart was also found in [4].

To establish convergence properties of these methods it is usually required that the step size \( \alpha_k \) should satisfy the strong Wolfe conditions (SWC):
\[ f(x_k) - f(x_k + \alpha_k d_k) \geq -\delta \alpha_k g_k^T d_k \]  
(11a)
\[ |g(x_k + \alpha_k d_k)| \leq \sigma g_k^T d_k \]  
(11b)
Where \( 0 < \delta < \sigma < \frac{1}{2} \). On the other hand, many other numerical methods (e.g. the steepest descent methods and quasi-Newton method) for unconstrained optimization are proved to be convergence under the standard Wolfe conditions (SDWC). which are weaker than the (SWC):
\[ f(x_k) - f(x_k + x_k^T d_k) \geq -\delta x_k^T g_k^T d_k \]  
(12a)
\[ g(x_k + x_k^T d_k)^T d_k \geq \sigma g_k^T d_k \]  
(12b)
Line search strategies require the descent condition
\[ g_k^T d_k < 0 \quad \forall k \]  
(13)
However most of conjugate gradient methods don't always generate a descent search direction [5], so condition (13) is usually assumed in the analysis and implementation. Some strategies have been studied which produce a descent search direction within the framework of conjugate gradient methods for example:
Hiroshi and Naoki in [5] generalized the Dai and Yuan (DY) [3], which is defined as follows:
\[ d_{k+1} = -g_{k+1} + \beta^{gy} g_k \]
Investigation on Extended Conjugate Gradient Non-linear methods for …

\[ \beta_{DY}^{k+1} = \frac{\eta_k y_k}{d_k^T g_k} \]  

(14)

Their generalization of (14) as follows: they are assumed that

\[ d_k^T g_k + \beta_k d_k^T g_k = \eta_k y_k \]  

(15)

Where \( \beta_k = \frac{\eta_k y_k}{d_k^T g_k} \)

(16)

The equation (15) is equivalent to

\[ \tau_{k+1} = \frac{\eta_k y_k}{d_k^T g_k} \]  

(17)

And they are suggested three different values for \( \tau_{k+1} \):

1) \( \tau_{k+1} = \frac{\eta_k y_k}{d_k^T g_k} \) therefore

\[ \beta_k = \frac{\eta_k y_k}{d_k^T g_k} \]  

(18)

2) \( \tau_{k+1} = \frac{\eta_k y_k + \tau_k \max \{ \eta_k, 0 \} }{d_k^T g_k} \)

\[ \beta_k = \frac{\eta_k y_k + \tau_k \max \{ \eta_k, 0 \} }{d_k^T g_k} \]  

(19)

Where \( \tau_k \geq 0 \) and \( \phi_k = 3 (f_k - f_{k+1}) + 3 (g_k + g_{k+1})^T s_k \) and \( u_k \) any vector with \( s_k^T u_k = 0 \)

3) \( \tau_{k+1} = \frac{\eta_k y_k + \tau_k \max \{ \phi_k, 0 \} }{d_k^T g_k} \)

\[ \beta_k = \frac{\eta_k y_k + \tau_k \max \{ \phi_k, 0 \} }{d_k^T g_k} \]  

(20)

The algorithms defined in equation (4) with \( \beta_k \) is defined in (18) or (19) or (20) is called Extension of the Dai-Yuan (DY) method and the search direction generated by the above algorithms generates descent direction whenever the condition (17) satisfied for more detail see [5].

This paper is organized as follows: In section 2 we deal with an extension of the DY method and we give another three different values for \( \tau_{k+1} \) this values are based to the descent property, pure conjugacy condition and Newton direction. In section 3 the convergence analysis studied and in section 4 the numerical experiments are reported.

2- New proposed algorithms

In this section, we try to find new values for \( \tau_{k+1} \) that satisfies the condition given in equation (17), using three different methods:

2.1 Descent property

Hiroshi and Naoki in [5] show that if the condition (17) is satisfied then the related conjugate gradient (CG) method will be generates always descent directions for all \( k \). Now consider

\[ \tau_{k+1} = \lambda_{k+1}^{(i)} d_k^T y_k + \left( 1 - \lambda_{k+1}^{(i)} \right) g_k^T g_k \]

\[ \lambda_{k+1}^{(i)} \in [0,1] \]

Then the search direction can be defined as:
\[
d_{k+1} = -g_{k+1} + \frac{(g^T_{k+2} d_{k+2})}{\lambda_k^T d_{k+2} + (1-\lambda_k^T) g_k^T g_k} d_k
g_{k+1}
\]
If
\[
d_{k+1} = -g_{k+1} + \frac{(g^T_{k+2} d_{k+2})}{\lambda_k^T d_{k+2} + (1-\lambda_k^T) g_k^T g_k} d_k^T g_{k+2} < 0
\]

With simple algebra
\[
\lambda_k^{(3)} = \frac{g_k^T g_{k+2}}{d_k^T g_k}
\]

Therefore our first new algorithm say (MH1-CG) can be define as MH1
\[
d_{k+1} = -g_{k+1} + \lambda_k^{(3)} \beta^{MH1} d_k
\]

Where
\[
\beta^{MH1} = \frac{g_k^T g_{k+2}}{\lambda_k^T d_{k+2} + (1-\lambda_k^T) g_k^T g_k}
\]

and
\[
\lambda_k^{(3)}
\]
is defined in equation (22)
with the condition if \(\lambda_k^{(3)} \leq 0\) set \(\lambda_k^{(3)} = 0\) and if \(\lambda_k^{(3)} > 0\) set \(\lambda_k^{(3)} = 1\).

In equation (23), we multiply \(\beta_k\) by \(\lambda_k^{(3)}\) for the purpose of the global convergence.

2.2 Pure Conjugacy property

The second method to evaluate the value of \(\tau_{k+1}\) is the pure conjugacy condition defined in equation (6), we assume that the following search direction generates conjugate directions
\[
d_{k+1} y_k = -g_{k+1} y_k + \frac{g_k^T g_{k+2}}{\lambda_k^T d_{k+2} + (1-\lambda_k^T) g_k^T g_k} d_k^T y_k = 0
\]

Solve the above equation for \(\lambda_k^{(2)}\) where \(\lambda_k^{(2)} \in [0,1]\) to get
\[
\lambda_k^{(2)} = \frac{g_k d_k - g_{k+2} d_{k+2}}{g_k d_k - g_{k+2} d_{k+2}}
\]

Then the second new algorithm (MH2-CG) say can be defined as
\[
d_{k+1} = g_{k+1} + \lambda_k^{(2)} \beta^{MH2} d_k
\]

When
\[
\beta^{MH2} = \frac{g_k^T g_{k+2}}{\lambda_k^T d_{k+2} + (1-\lambda_k^T) g_k^T g_k}
\]

and \(\lambda_k^{(2)}\) in equation (24) and if \(\lambda_k^{(2)} < 0\) or \(\lambda_k^{(2)} > 1\) set \(\lambda_k^{(2)} = 1\)

In equation (25), we multiply \(\beta_k\) by \(\lambda_k^{(2)}\) for the purpose of the global convergence.

2.3 Assuming parallel to the Newton direction

As we know when initial \(x_1\) is close enough to a local minimum point \(x^*\) then the best direction to be followed in the current point \(x_{k+1}\) is the Newton direction \(-G_{k+1}^{-1} g_{k+1}\). Therefore our motivation is to choose the parameter \(\beta_{k+1}\) in (4) so that for every \(k \geq 1\) the direction \(d_{k+1}\) given by
\[
d_{k+1} = -g_{k+1} + \frac{g_{k+2} g_{k+2}}{\lambda_k^T d_{k+2} + (1-\lambda_k^T) g_k^T g_k} d_k
\]

Can be best direction. Hence using the direction from the equality
Investigation on Extended Conjugate Gradient Non-linear methods for …

\[-g_{k+1} = -g_{k+1} + \frac{\alpha g_{k+1} - \beta g_{k}}{\lambda_k \beta - \alpha} \theta_k d_k \quad (26)\]

When $G^{-1}$ is inverse Hessian which is symmetric and positive definite. Multiply equation (26) by $y_k$ noting that $G^{-1}y_k = s_k$ and using equation (3) with simple computations we obtain

\[\lambda_k^{(3)} = \frac{(\alpha s_k + \beta s_{k+1})(s_k^T x_k) - \beta s_k (s_k^T x_k + s_{k+1}^T x_{k+1})}{(\beta s_k^T x_k + s_{k+1}^T x_{k+1})(s_k^T x_k - s_{k+1}^T x_{k+1})} \quad (27)\]

Then the third algorithm (MH3-CG) say is given by

\[d_{k+1} = -g_{k+1} + \lambda_k^{(3)} \beta_{MH3} d_k \quad (28)\]

Where \[\beta_{MH3} = \frac{\alpha}{\lambda_k^{(3)}} \frac{\theta_k^T s_k}{\beta s_k + s_{k+1} - \lambda_k^{(3)} \beta s_k} \quad (29)\]

Where $\lambda_k^{(3)}$ computed from (27) with the condition if $\lambda_k^{(3)} < 0$ or $\lambda_k^{(3)} > 1$ set $\lambda_k^{(3)} = 1$

In equation (28), we multiply $\beta_k$ by $\lambda_k^{(3)}$ for the purpose of the global convergence.

3- Convergence analysis

In this section we have proved the global convergence property of the algorithm MH1. Our proof are based to the theorem given in the paper proposed by Gilbert and Nocedal (Gilbert and Nocedal, 1992). They show that any non-linear conjugate gradient algorithm that satisfies the assumption (3.1) below will be globally convergent according to the theorem (1) and theorem (2) (given later on).

Assumption (3.1):

(a)

1- The level set $L = \{x; f(x) \leq f(x_k)\}$ is bounded below, where $x_k$ is initial estimate for the minimizer.

2- In some neighborhood $N$ of $L$ the objective function $f$ is continuously differentiable and its gradient is Lipchitz continuous.

3- The step size $\alpha_k$ satisfies the Wolfe conditions

(b) The parameter $\beta_k$ satisfies the following inequality

\[0 < \beta_k \leq \beta_k^{(3)} \quad , \quad \forall k \geq 0\]

Theorem (1):

suppose that assumption (3.1) hold. Consider any method of the form (2) and (4), with $0 < \alpha < \frac{1}{2}$ in SWC, then the method generator descent directions $d_k$ satisfying

\[-\frac{1}{1-\rho} \leq \frac{\theta_k^T d_k}{\|g_k\|} \leq \frac{2\rho-1}{1-\rho} \quad k = 1,2,3...\]

Proof (see Gilbert and Nocedal, 1992)
**Theorem (2):**
Suppose that assumption (3.1) hold. consider any method of the form (2) and (4) with \(0 < \sigma < \frac{1}{2}\), then
\[
\lim_{k \to \infty} \inf \|g_{k+1}\| = 0
\]
Proof (see Gilbert and Nocedal).
Theorem (1) and Theorem (2) shows that for conjugate gradient methods, for which \(0 < \beta_k \leq \beta_{FR}\) and \(\alpha_k\) satisfies strong Wolfe conditions then the methods generates descent direction and they are globally convergent. Therefore to prove descent property and global convergent to the MH1 or (MH2,MH3)-conjugate gradient methods we need only to show
\[
0 < \beta_{MH1} \leq \beta_{FR}\tag{30}
\]
To prove the inequality (30). Since \(\lambda_k^{(1)}, \alpha_k\) are positive scalars \((\lambda_k, \alpha_k \in [0,1])\) and \(d_k^T y_k \geq 0\) by second Wolfe condition then
\[
\lambda_k \frac{\lambda_k^{(1)}(g_{k+1}^T g_{k+1})}{d_k^T y_k + (1 - \lambda_k)g_k^T g_k} > 0
\]
To establish the second part of the inequality (30) from (31) we have
\[
\frac{\lambda_k^{(1)}d_k^T y_k + g_k^T g_k - \lambda_k^{(1)}g_k^T g_k}{\lambda_k^{(1)}d_k^T y_k + (1 - \lambda_k^{(1)})g_k^T g_k} \leq \frac{\lambda_k^{(1)}d_k^T y_{k+1} + g_{k+1}^T g_{k+1}}{\lambda_k^{(1)}d_k^T y_{k+1} + (1 - \lambda_k^{(1)})g_{k+1}^T g_{k+1}}\tag{32}
\]
Multiply equation (32) by \(\lambda_k^{(1)}g_{k+1}^T g_{k+1}\) to set
\[
\frac{\lambda_k^{(1)}d_k^T y_{k+1} + g_{k+1}^T g_{k+1}}{\lambda_k^{(1)}d_k^T y_{k+1} + (1 - \lambda_k^{(1)})g_{k+1}^T g_{k+1}} \leq \frac{\lambda_k^{(1)}d_k^T y_{k+1} + g_{k+1}^T g_{k+1}}{\lambda_k^{(1)}d_k^T y_{k+1} + (1 - \lambda_k^{(1)})g_{k+1}^T g_{k+1}} \leq \beta_{FR}\nabla
\]

4- **Numerical Experiments**
This section presents the performance of FORTRAN implementation of our new conjugate gradient algorithms (MH1,MH2 and MH3) on a set of unconstrained optimization test problems taken from (Andrei, 2008). We select (15) large scale test problems in extended or generalized form (see Appendix), for each function we have considered numerical experiments with number of variables \(n = 100\) and \(n = 1000\).

We have compared the performance of these algorithms versus \(\beta_k\) given in equation (18) [which is better than from the \(\beta_k\) given in equation (19) or (20) see (Heroshi and Naoki,2005)].

All these algorithms are implemented with the standard Wolfe line search conditions with \(\delta = 0.001\) and \(\sigma = 0.9\), where the initial step-size \(\alpha_k = \frac{1}{\|g_0\|}\) and guess for other iterations i.e. \(k > 1; \alpha_k = \alpha_{k-1} \times \frac{\|g_{k-1}\|}{\|g_k\|}\).

In the all cases the stopping criteria is the \(\|g_{k+1}\| < 10^{-5} \times \max[1,\|f_{k+1}\|]\) and the maximum number of iterations is 2000. Our comparisons includes the following:
Investigation on Extended Conjugate Gradient Non-linear methods for …

1- NOI: Number of iterations
2- FGN: Number of function and gradient evaluations which are same in these algorithms
3- Lins: number of calling subroutine \( t = \text{compute step-size} \)

Table (1) and (2) illustrate the details of the results for and \( n = 100 \) and \( n = 1000 \).

Table (3) presents the performance of all algorithms in terms of percentage where DY method considered as 100%. From table we see that all algorithms improves DY-CG method about \( 2\%-20\% \) in terms of NOI.

Table (1): comparison of algorithms w.r. to percentage of NOI, FGN & Lins

<table>
<thead>
<tr>
<th>N=100</th>
<th>FR</th>
<th>DY</th>
<th>EXDY</th>
<th>MH1</th>
<th>MH2</th>
<th>MH3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NOI / FGN / Lins</td>
<td>NOI / FGN / Lins</td>
<td>NOI / FGN / Lins</td>
<td>NOI / FGN / Lins</td>
<td>NOI / FGN / Lins</td>
<td>NOI / FGN / Lins</td>
</tr>
<tr>
<td>1</td>
<td>659/16198/594</td>
<td>13 / 29 / 13</td>
<td>22 / 40 / 14</td>
<td>13 / 28 / 13</td>
<td>47 / 395 / 34</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>19 / 35 / 13</td>
<td>18 / 33 / 12</td>
<td>19 / 34 / 13</td>
<td>18 / 34 / 13</td>
<td>45 / 89 / 39</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>47 / 93 / 42</td>
<td>34 / 67 / 26</td>
<td>34 / 72 / 28</td>
<td>34 / 72 / 28</td>
<td>34 / 72 / 28</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>43 / 88 / 33</td>
<td>34 / 65 / 25</td>
<td>38 / 81 / 32</td>
<td>38 / 81 / 32</td>
<td>38 / 81 / 32</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>2001/2025/20</td>
<td>61 / 103 / 42</td>
<td>61 / 102 / 38</td>
<td>61 / 102 / 38</td>
<td>61 / 102 / 38</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>25 / 43 / 15</td>
<td>21 / 42 / 18</td>
<td>21 / 42 / 18</td>
<td>21 / 42 / 18</td>
<td>21 / 42 / 18</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>15 / 25/ 9</td>
<td>14 / 20 / 5</td>
<td>14 / 20 / 5</td>
<td>14 / 20 / 5</td>
<td>14 / 20 / 5</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>37 / 67 / 29</td>
<td>34 / 66 / 26</td>
<td>34 / 66 / 26</td>
<td>34 / 66 / 26</td>
<td>34 / 66 / 26</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>180/313/12</td>
<td>18 / 33 / 12</td>
<td>18 / 33 / 12</td>
<td>18 / 33 / 12</td>
<td>18 / 33 / 12</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>63 / 93 / 42</td>
<td>55 / 86 / 29</td>
<td>55 / 86 / 29</td>
<td>55 / 86 / 29</td>
<td>55 / 86 / 29</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>43 / 88 / 33</td>
<td>34 / 65 / 25</td>
<td>38 / 81 / 32</td>
<td>38 / 81 / 32</td>
<td>38 / 81 / 32</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>25 / 43 / 15</td>
<td>21 / 42 / 18</td>
<td>21 / 42 / 18</td>
<td>21 / 42 / 18</td>
<td>21 / 42 / 18</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>15 / 25 / 9</td>
<td>14 / 20 / 5</td>
<td>14 / 20 / 5</td>
<td>14 / 20 / 5</td>
<td>14 / 20 / 5</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>37 / 67 / 29</td>
<td>34 / 66 / 26</td>
<td>34 / 66 / 26</td>
<td>34 / 66 / 26</td>
<td>34 / 66 / 26</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>180/313/12</td>
<td>18 / 33 / 12</td>
<td>18 / 33 / 12</td>
<td>18 / 33 / 12</td>
<td>18 / 33 / 12</td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>673 / 2255 / 456</td>
<td>541 / 943 / 353</td>
<td>592 / 1018 / 387</td>
<td>548 / 946 / 354</td>
<td>639 / 1439 / 449</td>
<td></td>
</tr>
</tbody>
</table>

Table(2): comparison of algorithms w.r. to percentage of NOI, FGN & Lins

<table>
<thead>
<tr>
<th>N=1000</th>
<th>FR</th>
<th>DY</th>
<th>EXDY</th>
<th>MH1</th>
<th>MH2</th>
<th>MH3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NOI / FGN / Lins</td>
<td>NOI / FGN / Lins</td>
<td>NOI / FGN / Lins</td>
<td>NOI / FGN / Lins</td>
<td>NOI / FGN / Lins</td>
<td>NOI / FGN / Lins</td>
</tr>
<tr>
<td>1</td>
<td>1585/44127/1566</td>
<td>13 / 27 / 13</td>
<td>12 / 26 / 12</td>
<td>13 / 27 / 13</td>
<td>17 / 33 / 15</td>
<td>66 / 1515 / 66</td>
</tr>
<tr>
<td>2</td>
<td>38 / 65 / 22</td>
<td>38 / 65 / 22</td>
<td>29 / 54 / 20</td>
<td>27 / 50 / 18</td>
<td>27 / 50 / 18</td>
<td>28 / 53 / 20</td>
</tr>
<tr>
<td>3</td>
<td>38 / 65 / 22</td>
<td>38 / 65 / 22</td>
<td>29 / 54 / 20</td>
<td>27 / 50 / 18</td>
<td>27 / 50 / 18</td>
<td>28 / 53 / 20</td>
</tr>
<tr>
<td>4</td>
<td>46 / 92 / 38</td>
<td>32 / 46 / 24</td>
<td>32 / 69 / 27</td>
<td>32 / 69 / 27</td>
<td>32 / 69 / 27</td>
<td>43 / 91 / 40</td>
</tr>
<tr>
<td>5</td>
<td>2001 / 2005 / 3</td>
<td>201 / 329 / 120</td>
<td>191 / 308 / 109</td>
<td>198 / 311 / 114</td>
<td>204 / 332 / 120</td>
<td>174 / 287 / 105</td>
</tr>
<tr>
<td>6</td>
<td>46 / 74 / 46</td>
<td>26 / 56 / 25</td>
<td>32 / 51 / 23</td>
<td>32 / 51 / 23</td>
<td>32 / 51 / 23</td>
<td>21 / 47 / 21</td>
</tr>
<tr>
<td>7</td>
<td>127 / 3531 / 124</td>
<td>11 / 19 / 7</td>
<td>9 / 16 / 9</td>
<td>10 / 17 / 6</td>
<td>10 / 17 / 6</td>
<td>10 / 17 / 6</td>
</tr>
</tbody>
</table>
| 8 | 73 / 115 / 40 | 64 / 101 / 35 | 46 / 72 / 24 | 63 / 99 / 34 | 55 / 89 / 32 | 67 / 110 /
Khalil K. Abbo & Osama M. T. Waiss

<table>
<thead>
<tr>
<th></th>
<th>2001 / 2110 / 108</th>
<th>85 / 156 / 70</th>
<th>79 / 148 / 68</th>
<th>76 / 139 / 62</th>
<th>66 / 125 / 58</th>
<th>41</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>61 / 96 / 32</td>
<td>64 / 102 / 35</td>
<td>55 / 87 / 29</td>
<td>52 / 84 / 29</td>
<td>52 / 83 / 28</td>
<td>52 / 84 / 29</td>
</tr>
<tr>
<td>10</td>
<td>2001 / 2025 / 11</td>
<td>31 / 65 / 12</td>
<td>9 / 20 / 5</td>
<td>18 / 19 / 0</td>
<td>18 / 19 / 0</td>
<td>18 / 19 / 0</td>
</tr>
<tr>
<td>11</td>
<td>77 / 129 / 51</td>
<td>15 / 29 / 13</td>
<td>12 / 24 / 11</td>
<td>12 / 23 / 10</td>
<td>11 / 22 / 10</td>
<td>10 / 21 / 10</td>
</tr>
<tr>
<td>14</td>
<td>98 / 1967 / 86</td>
<td>37 / 349 / 27</td>
<td>34 / 59 / 20</td>
<td>26 / 46 / 16</td>
<td>28 / 51 / 19</td>
<td>24 / 47 / 19</td>
</tr>
<tr>
<td>total</td>
<td>1204 / 2347 / 785</td>
<td>1099 / 1853 / 697</td>
<td>1092 / 1851 / 704</td>
<td>1145 / 1929 / 722</td>
<td>1190 / 3399 / 771</td>
<td></td>
</tr>
</tbody>
</table>

Table(3): comparison of algorithms w.r. to percentage of NOI

<table>
<thead>
<tr>
<th>N</th>
<th>Measure</th>
<th>DY</th>
<th>EXDY</th>
<th>MH1</th>
<th>MH2</th>
<th>MH3</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>NOI</td>
<td>100%</td>
<td>80.7%</td>
<td>88.3%</td>
<td>81.8%</td>
<td>91.7%</td>
</tr>
<tr>
<td>1000</td>
<td>NOI</td>
<td>100%</td>
<td>90.8%</td>
<td>90.5%</td>
<td>95%</td>
<td>98.7%</td>
</tr>
</tbody>
</table>

Appendix

1- Extended Freudenstein & Roth Function

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

\[x_0 = \left[ 0.5, -2, 0.5, -2, \ldots, 0.5, -2 \right] \]

2- Extended Trigonometric Function

\[
f(x) = \sum_{i=1}^{n} \left( n - \frac{\sum_{j=1}^{n} \cos x_j}{(1 - \cos x_j - \sin x_j)} \right)^2,
\]

\[x_0 = \left[ 0.2, 0.2, \ldots, 0.2 \right] \]

3- Extended Rosenbrock Function

\[
f(x) = \sum_{i=1}^{n} c \left( x_{2i} - x_{2i-1}^2 \right)^2 + \left( 1 - x_{2i-1} \right)^2,
\]

\[x_0 = [-1, 2, 1, \ldots, -1, 2, 1]. \quad c = 100
\]

4- Extended White & Holst Function

\[
f(x) = \sum_{i=1}^{n} c \left( x_{2i} - x_{2i-1}^2 \right)^2 + \left( 1 - x_{2i-1} \right)^2,
\]

\[x_0 = [-1, 2, 1, \ldots, -1, 2, 1]. \quad c = 100
\]

5- Diagonal2 Function

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

\[x_0 = \left[ 1/1, 1/2, \ldots, 1/n \right].\]
6- Generalized Tridiagonal-1 Function
\[ f(x) = \sum_{i=1}^{n} (x_i + x_{i+2} - 3)^2 + (x_i - x_{i+2} + 1)^4, \]
\[ x_0 = [2, 2, ..., 2]. \]

7- Extended Three Exponential Terms Function
\[ f(x) = \sum_{i=1}^{n} \left( \exp(x_{2i-1} + 3x_{2i} - 0.1) + \exp(x_{2i-1} - 3x_{2i} - 0.1) + \exp(-x_{2i} - 0.1) \right), \]
\[ x_0 = [0.1, 0.1, ..., 0.1]. \]

8- Generalized Tridiagonal-2 Function
\[ f(x) = \left( (5 - 3x_i - x_i^2)x_i - 3x_i + 1 \right)^2 + \sum_{i=1}^{n-1} \left( (5 - 3x_i - x_i^2)x_i - x_{i-1} - 3x_{i+1} + 1 \right)^2 + \left( (5 - 3x_i - x_i^2)x_i - x_{i-1} + 1 \right)^2, \]
\[ x_0 = [-1, -1, ..., -1]. \]

9- Extended Powell Function
\[ f(x) = \sum_{i=1}^{n} \left( x_{4i-3} + 10x_{4i-2}^2 + 5(x_{4i-1} - x_{4i})^2 + (x_{4i-2} - 2x_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^4 \right), \]
\[ x_0 = [3, -1, 0, ..., 3, -1, 0]. \]

10- Extended Block Diagonal BD1 Function
\[ f(x) = \sum_{i=1}^{n} \left( x_{2i-1}^2 + x_{2i}^2 - 2 \right)^2 + (\exp(x_{2i-1} - 1) - x_{2i})^2, \]
\[ x_0 = [0, 1, 0, ..., 0, 1]. \]

11- Extended Cliff Function
\[ f(x) = \sum_{i=1}^{n} \left( \frac{x_{2i-1} - 3}{100} \right)^2 - (x_{2i-1} - x_{2i}) + \exp\left( 20(x_{2i-1} - x_{2i}) \right), \]
\[ x_0 = [0, -1, ..., 0, -1]. \]

12- Extended Tridiagonal-1 Function
\[ f(x) = \sum_{i=1}^{n} (x_{2i-1} + x_{2i} - 3)^2 + (x_{2i-1} - x_{2i} + 1)^4, \]
\[ x_0 = [2, 2, ..., 2]. \]

13- Partial Perturbed Quadratic
\[ f(x) = x_1^2 + \sum_{i=1}^{n} \left( i x_i^2 + \frac{1}{100} (x_1 + x_2 + \cdots + x_i)^2 \right), \]
\[ x_0 = [0.5, 0.5, ..., 0.5]. \]
14- Almost Perturbed Quadratic

\[ f(x) = \sum_{i=1}^{n} i x_i^2 + \frac{1}{100} (x_1 + x_n)^2, \]
\[ x_0 = [0.5, 0.5, ..., 0.5]. \]

15- VARDIM Function (Cut)

\[ f(x) = \sum_{i=1}^{n} (x_i - 1)^2 + \left( \sum_{i=1}^{n} i x_i - \frac{n(n+1)}{2} \right)^2 + \]
\[ \left( \sum_{i=1}^{n} i x_i - \frac{n(n+1)}{2} \right)^4, \]
\[ x_0 = [1 - \frac{1}{n}, 1 - \frac{2}{n}, ..., 1 - \frac{n}{n}]. \]

References


8) Yuan G. and La X. "A New line search method with Trust Region four Unconstrained optimization" communication on Applied Non-liner Analyses Vol. 15.No1, 2008.

