Abstract

The non-linear conjugate gradient method is a very useful technique for solving Large-Scale minimization problems and has wide applications in many fields. In this paper, we present a new spectral type, a non-linear conjugate gradient algorithm the derivatation of this algorithm is based on Fletcher – Reeves and Newton algorithm, the descent property for the suggested algorithm is proved provided that the step size $\alpha_k$ satisfies the Wolfe conditions. Numerical results show that the new algorithm is efficient in practical computation and superior to the Fletcher – Reervs algorithm in many situations.

* Department of Math /College of Computers Sciences and Mathematics/ University of Mosul

**Department of Math /College of Computers Sciences and Mathematics/ University of Mosul

Received: 8/12/2010    Accepted: 2/2/2011
1. **Introduction**

Consider the Unconstrained Optimization problem

\[ \min f(x), \quad x \in \mathbb{R}^n \]  

(1)

where \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is continuously differentiable function. The line search algorithm for (1) often generates a sequence of iterates \((x_k)\) by letting

\[ x_{k+1} = x_k + \alpha_k \ d_k \]  

(2)

Where \( x_k \) is the current iterate point, \( d_k \) is a search direction, and \( \alpha_k > 0 \) is step-length. Different choices of \( d_k \) and \( \alpha_k \) will determine different line search methods [1,2]. These methods are divided into two stages at each iteration [3]:

1. choose a descent search direction \( d_k \) i.e

\[ g_k^T d_k < 0 \]  

(3)

2. choose a step-length \( \alpha_k \) 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 \( g_{k+1} \), \( \| \| \) 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 application in solving large-scale minimization problems and training feed forward neural networks [4]. However, the steepest descent method often yields zig-zag phenomena in solving practical problems, which makes the algorithm converges to an optimal solution very slowly, or even fail to converge [5]. Then the steepest descent method is not the fastest one among the line search methods. If $d_k = -H_k g_k$ is the search direction at each iteration in the algorithm, where is an $n \times n$ matrix approximating $[\nabla^2 f(x)]^{-1}$, then the corresponding line search method is called Newton like method [5] such as Newton method, Quasi – Newton method, Variable metric method etc. Many papers [6] have been proposed by the method for optimization problems. However, one drawback of the Newton like methods is required to store and compute matrix $H_k$ at each iteration and thus adds the cost of storage and computation. Accordingly, this method is not suitable to solve large scale optimization problems in many cases. When $n$ is large the related problem is called large – scale minimization problem. In order to solve large–scale minimization problems, We need to design special algorithms that avoid the high storage and computation cost of some matrices.

The conjugate gradient method is a suitable approach for solving large – scale minimization problems. For strictly convex quadratic objective functions, the conjugate gradient method with
exact line searches has finite convergence property [9]. If the objective function is not quadratic or inexact line searches are used, the conjugate gradient method has no quadratic convergence property [7]. When the conjugate gradient method is used to minimize non–quadratic objective functions, the related algorithm is called non – linear conjugate gradient method. Meanwhile, some new non – linear conjugate gradient methods have appeared in [8].

The non – linear conjugate gradient (CG) method has the form

$$x_{k+1} = x_k + \alpha_k d_k$$  \hspace{1cm} (4)

Where $x_i$ is an initial point, $\alpha_k$ is a step – length and $d_i$ can be taken as $d_i = -g_i$ and

$$d_{k+1} = -g_{k+1} + \beta_{k+1} d_k$$  \hspace{1cm} (5)

Different $\beta_{k+1}$ will determine different CG method. Some famous formula for $\beta_{k+1}$ as follows :

$$\beta_{k+1}^{FR} = \frac{y_k^T g_{k+1}}{y_k^T y_k}$$  \hspace{1cm} (Fletcher-Reeves [9] ) \hspace{1cm} (6)

$$\beta_{k+1}^{HS} = \frac{y_k^T y_k}{y_k^T d_k}$$  \hspace{1cm} (Hestenes-Stiefel [10] ) \hspace{1cm} (7)

$$\beta_{k+1}^{DY} = \frac{y_k^T g_{k+1}}{y_k^T d_k}$$  \hspace{1cm} (Dai- Yuan [11] ) \hspace{1cm} (8)
Where \( y_k = g_{k+1} - g_k \). Although some conjugate gradient methods have good numerical performance in solving Large-Scale minimization problems, they have no global convergence in some situations such as Hestenens – Stiefiel conjugate gradient method [12], and others has global convergence theoretically but don't perform well in practice such as Fletcher – Reeves (\( \beta_{k+1}^{FR} \)) method. We often have two questions. Can we construct a conjugate gradient (CG) method that has both global convergence and good numerical performance in practical computation? or can we design a conjugate gradient method that is suitable to solve ill-conditioned minimization problems?

Yuan and Stoer in [13] studied the CG method on a subspace and they obtained a new conjugate gradient method. In their algorithm, the search direction was taken from the subspace \([ g_{k+1}, d_k ]\) at the \((k+1)\)th iteration \((k \geq 1)\) i.e.

\[
d_{k+1} = -y_{k+1} g_{k+1} + \beta_{k+1} d_k
\]  

(9)

Where \( y_{k+1} \) and \( \beta_{k+1} \) are two parameters. The other important aspect in the line search algorithms is the computing the step-length \( \alpha_k \), it has an important influence on the amount of calculations at each iteration. There are several line search rules for choosing step-length \( \alpha_k \) for example, the exact minimization rule, the step-length \( \alpha_k \) is chosen such that
\begin{align*}
    f(x_k + \alpha_k d_k) &= \arg\min_{\alpha > 0} f(x_k + \alpha d_k) 
\end{align*}

Armijo rule, Wolf rule, etc. In this paper we use the following two line search conditions:

1. The Wolfe – Powell (WWP) line search condition:
\begin{align*}
    f_{k+1} &\leq f_k + \rho \alpha_k g_k^T d_k 
    \tag{11a} \\
    g_{k+1}^T d_k &\geq \sigma g_k^T d_k 
    \tag{11b}
\end{align*}

2. The Strong Wolfe – Powell (SWP): The equation (11a) with
\begin{align*}
    |g_{k+1}^T d_k| &\leq -\sigma g_k^T d_k 
    \tag{11c}
\end{align*}

where \( \rho \in (0, 1) \) and \( \sigma \in (\rho, 1) \).

In general conjugate gradient methods usually implemented with restart since the rate of convergence of the algorithm is only linear unless the iterative procedure is restarted occasionally. It is usual to restart at every n or n+1 iterations but this not satisfactory since n is large, therefore other restarts are used such as Powell restarts [15] defined by
\begin{align*}
    |g_{k+1}^T g_k| &\geq 0.2 \|g_k\| 
    \tag{12}
\end{align*}

2. New Proposed Algorithm (SFR – CG say)

The search direction for the Fletcher – Reeves conjugate gradient (FR – CG) method is obtained by \( d_i = -g_i \) and
\[ d_{k+1} = -g_{k+1} + \frac{g_{T}T_{k+1}g_{k+1}}{g_{k}Tg_{k}} d_{k} \]  

(13)

Dai and Yuan in [14] established global convergence results for FR – CG method for any starting point with exact and inexact line searches, therefore it has better convergence property theoretically but it is not recommended for practical use since it has poor performance in practice see [16]. In order to accelerate the FR – CG method we use equation (9) as follows

Let \( \gamma_{k+1} = 1 + \mu_{k+1} \) 

(14)

Where \( \gamma_{k+1} \) and \( \mu_{k+1} \) are two parameters, then

\[ d_{k+1} = -(1 + \mu_{k+1}) g_{k+1} + \frac{g_{T}T_{k+1}g_{k+1}}{g_{k}Tg_{k}} d_{k} \]  

(15)

Or \[ d_{k+1} = -g_{k+1} - \mu_{k+1} g_{k+1} + \frac{g_{T}T_{k+1}g_{k+1}}{g_{k}Tg_{k}} d_{k} \]  

(16)

We incorporate the second order information to the search direction in (16) by assuming, the direction in (16) is parallel to the Newton direction i.e

\[ -G_{T}^{-1}g_{k+1} = -g_{k+1} - \mu_{k+1} g_{k+1} + \frac{g_{T}T_{k+1}g_{k+1}}{g_{k}Tg_{k}} d_{k} \]  

(17) Where \(-G_{T}^{-1}\) is the inverse Hessian matrix. Now suppose \(-G_{k+1}^{-1}\) is symmetric \((G^{-1} = (G^{-1})^T)\), positive definite and satisfies the Quasi – Newton condition i.e

\[ G_{k+1}^{-1}y_{k} = s_{k} \]  

(18)
Where \( s_k = x_{k+1} - x_k \) multiply both sides of (17) by \( y_k \) and considering \( G^{-1}_{k+1} \) be symmetric and positive definite, then

\[
-(G^{-1}_{k+1}y_k)^Tg_{k+1} = -y_k^Ty_{k+1} - \mu_{k+1}y_k^Tg_{k+1} + \frac{g_{k+1}^Tg_{k+1}}{g_k^Ty_k}y_k^Td_k
\]

Use the relation given in (18) to get

\[
-s_k^Tg_{k+1} = -y_k^Ty_{k+1} - \mu_{k+1}y_k^Tg_{k+1} + \frac{g_{k+1}^Tg_{k+1}}{g_k^Ty_k}y_k^Td_k
\]

Divide both sides in the above equation by \( y_k^Td_k \) then

\[
-\frac{s_k^Tg_{k+1}}{y_k^Td_k} = -\beta_{k+1}^{HS} - \mu_{k+1}\beta_{k+1}^{HS} + \beta_{k+1}^{FR}
\]

Or

\[
-\frac{s_k^Tg_{k+1}}{y_k^Td_k} = -(1 + \mu_{k+1})\beta_{k+1}^{HS} + \beta_{k+1}^{FR}
\]

Or

\[
(1 + \mu_{k+1})\beta_{k+1}^{HS} = \beta_{k+1}^{FR} + \frac{s_k^Tg_{k+1}}{y_k^Td_k}
\]

\[
\therefore \gamma_{k+1} = \frac{\beta_{k+1}^{FR}}{\beta_{k+1}^{HS}} + \frac{s_k^Tg_{k+1}}{\beta_{k+1}^{HS}y_k^Td_k}
\]

(19)

Use this value for the \( \gamma_{k+1} \) in (15) then

\[
d_{k+1} = -\gamma_{k+1}g_{k+1} + \beta_{k+1}^{FR}d_k
\]

(20)

We call the algorithm defined in (19) and (20) as spectral Fletcher – Reeves algorithm (SFR – CG) and we summarize it as the following algorithm SFR – CG.

Algorithm(SFR-CG):

\[
\text{Algorithm(SFR-CG)}:
\]
steep(1) : Initialization : select \( x_k \in \mathbb{R}^n, \gamma > 0 \), is small positive real value

and compute

\[ d_1 = -g_1, \quad \alpha_1 = \frac{1}{\|g_1\|} \quad \text{and} \quad k = 1 \]

step(2) : Test for convergence: If \( \|g_k\| \leq \varepsilon \), stop \( x_k \) is optimal solution

else go to step(3).

step(3) : Line search : compute \( \alpha_k \) satisfying the wolf conditions (11a),

(11b) and update the variable \( x_{k+1} = x_k + \alpha_k \ d_k \),

compute

\[ f_{k+1}, \ g_{k+1}, \ y_k \ and \ s_k. \]

step(4) : Direction computation : compute \( y_{k+1} \) from (19), if

\[ y_{k+1} \geq 1 \ or \ y_{k+1} \leq 0 \quad \text{set} \quad y_{k+1} = 1 \quad \text{and} \]

\[ d = -y_{k+1} g_{k+1} + \beta_{k+1}^{FR} \ d_k \]

If Powell restart (12) is satisfied then \( d_{k+1} = -y_{k+1} g_{k+1} \) else

\[ d_{k+1} = d \]

and \( \alpha_{k+1} = \alpha_k \cdot \frac{\|d_k\|}{\|d_{k+1}\|}, \quad k = k + 1 \) go to step(2).

3. Descent property

In this section we prove that the our algorithm defined in the equations (19) and (20) generates descent directions for all iteration according to the following theorem
Theorem: Consider the algorithm defined in equation (4) where \( d_k \) computed from (19) and (20). Assume that the step size \( \alpha_k \) satisfies the Wolfe conditions (11a) and (11b). Then the search directions \( d_k \) generated by the SFR – CG algorithm are descent for all \( k \) provided \( y_k ^T g_{k+1} > 0 \).

Proof

The prove is by indication, for \( k = 1 \),

\[
d_1 = -g_1 \rightarrow d_1 ^T g_1 = -\|g_1\| < 0
\]

now suppose \( d_k ^T g_k < 0 \) or \( s_k ^T g_k < 0 \) since \( s_k = \alpha_k \cdot d_k \), then for \( k + 1 \) we have

\[
d_{k+1} = -\left( \frac{\beta_{FR}}{\beta_{IS}} + \frac{s_k ^T g_{k+1}}{y_k ^T g_{k+1}} \right) g_{k+1} + \frac{g_{k+1} ^T g_{k+1}}{g_k ^T g_k} d_k
\]

\[
d_{k+1} ^T g_{k+1} = -\left( \frac{g_{k+1} ^T g_{k+1}}{\alpha_k \cdot g_k ^T g_k} \cdot \frac{s_k ^T y_k}{y_k ^T g_k} + \frac{s_k ^T g_{k+1}}{y_k ^T g_{k+1}} \right) g_{k+1} ^T g_{k+1} + \frac{g_{k+1} ^T g_{k+1}}{\alpha_k \cdot g_k ^T g_k} s_k ^T g_{k+1}
\]

Divide both sides by \( \frac{g_{k+1} ^T g_{k+1}}{\alpha_k \cdot g_k ^T g_k} \) then

\[
\alpha_k \cdot \frac{g_k ^T g_k}{g_{k+1} ^T g_{k+1}} \cdot d_{k+1} ^T g_{k+1} = -\frac{1}{y_k ^T g_{k+1}} (s_k ^T y_k ^T g_{k+1} g_k + \alpha_k \cdot g_k ^T g_k s_k ^T g_{k+1}) + s_k ^T g_{k+1}
\]

\[
\therefore s_k ^T g_{k+1} = s_k ^T g_{k+1} - s_k ^T g_k + s_k ^T g_k = s_k ^T g_k + s_k ^T g_k < s_k ^T y_k
\]
\[
\alpha_k g_k^T g_k d_{k+1}^T g_{k+1} \leq -\frac{1}{y_k^T g_{k+1}} (s_k^T y_k g_{k+1}^T g_{k+1} + \alpha_k g_k^T g_k s_k^T y_k) + s_k^T y_k
\]

\[
\therefore d_{k+1}^T g_{k+1} \leq -\frac{s_k^T y_k}{y_k^T g_{k+1}} (g_{k+1}^T g_{k+1} + \alpha_k g_k^T g_k - y_k^T g_{k+1}) \frac{g_{k+1}^T g_{k+1}}{\alpha_k g_k^T g_k}
\]

\[
= -\frac{s_k^T y_k}{y_k^T g_{k+1}} (g_{k+1}^T g_{k+1} + \frac{g_{k+1}^T g_{k+1} g_{k+1}^T g_{k+1}}{\alpha_k g_k^T g_k})
\]

Use the Cuchy – Schurz inequality then

\[
\leq -\frac{s_k^T y_k}{y_k^T g_{k+1}} \|g_{k+1}\|^2 (1 + \frac{\|g_{k+1}\|\|g_k\|}{\alpha_k}) = -\frac{s_k^T y_k}{y_k^T g_{k+1}} \|g_{k+1}\|^2 (1 + \frac{1}{\alpha_k} \sqrt{\beta^2})
\]

\[
s_k^T y_k > 0 \text{ by Wolfe condition and } y_k^T g_{k+1} > 0 \text{ by assumption}
\]

\[
\therefore d_{k+1}^T g_{k+1} < 0
\]

The proof is complete.

4. Computational Results and Comparisons.

This section presents the performance of FORTRAN implementation of our new spectral conjugate gradient algorithm (SFR -CG) on a set of unconstrained optimization test problems taken from [17]. We select (15) Lange – Scale test problems in extended or generalized from (see Appendix), for each function we have considered numerical experiments with the number of variables n=100, 1000 and 10000. We have compared the performance of this algorithms versus To FR –
CG algorithm. These algorithms are implemented with standard Wolfe conditions with \( \rho = 0.001 \) and \( \sigma = 0.9 \). where the initial step-size \( \alpha_1 = \frac{1}{\|g_1\|} \) and initial guess for other iterations i.e. 

\[(k > 1) \quad \alpha_k = \alpha_{k-1} \cdot \frac{\|d_{k-1}\|}{d_k} \]. In the all cases the stopping criterion is \( \|g_{k+1}\| \leq 10^{-5} \) and maximum number of iteration is 2000. Our comparison includes the following:

1. NoI : Number of iterations.
2. FGE : Number of function and gradient evaluations.
3. LIN : The number of calling line search subroutine.
4. The total time required to solve (15) problem in particular dimension.

Tables (1), (2) and (3) show the details of the results for (SFR–CG) algorithms versus FR-CG algorithm.

**Table (1) Comparison of algorithms for N=100**

<table>
<thead>
<tr>
<th>Test</th>
<th>Dimn</th>
<th>FR-CG algorithm</th>
<th>SFR-CG algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>NOI / FGE / LIN</td>
<td>NOI / FGE / LIN</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>18 / 34 / 13</td>
<td>18 / 33 / 12</td>
</tr>
<tr>
<td>2</td>
<td>100</td>
<td>42 / 86 / 35</td>
<td>43 / 91 / 35</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>36 / 76 / 31</td>
<td>31 / 67 / 25</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>11 / 28 / 10</td>
<td>9 / 25 / 9</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>10 / 19 / 8</td>
<td>10 / 20 / 9</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>68 / 130 / 61</td>
<td>73 / 137 / 63</td>
</tr>
<tr>
<td>7</td>
<td>100</td>
<td>63 / 98 / 33</td>
<td>63 / 98 / 33</td>
</tr>
<tr>
<td>8</td>
<td>100</td>
<td>72 / 164 / 69</td>
<td>65 / 154 / 63</td>
</tr>
<tr>
<td>9</td>
<td>100</td>
<td>50 / 89 / 38</td>
<td>48 / 85 / 36</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>88 / 190 / 86</td>
<td>88 / 192 / 86</td>
</tr>
<tr>
<td>11</td>
<td>100</td>
<td>77 / 120 / 42</td>
<td>79 / 120 / 40</td>
</tr>
<tr>
<td>12</td>
<td>100</td>
<td>30 / 48 / 17</td>
<td>31 / 52 / 20</td>
</tr>
<tr>
<td>13</td>
<td>100</td>
<td>24 / 46 / 19</td>
<td>24 / 43 / 16</td>
</tr>
<tr>
<td>14</td>
<td>100</td>
<td>17 / 33 / 15</td>
<td>17 / 33 / 15</td>
</tr>
</tbody>
</table>
## Table (2) Comparison of algorithms for N=1000

<table>
<thead>
<tr>
<th>Test Fun</th>
<th>Dim N</th>
<th>FR-CG algorithm</th>
<th>SFR-CG algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>NOI / FGE / LIN</td>
<td>NOI / FGE / LIN</td>
</tr>
<tr>
<td>1</td>
<td>1000</td>
<td>40 / 69 / 24</td>
<td>39 / 69 / 25</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>42 / 96 / 40</td>
<td>38 / 84 / 34</td>
</tr>
<tr>
<td>3</td>
<td>1000</td>
<td>37 / 80 / 33</td>
<td>37 / 77 / 29</td>
</tr>
<tr>
<td>4</td>
<td>1000</td>
<td>94 / 96 / 86</td>
<td>22 / 47 / 13</td>
</tr>
<tr>
<td>5</td>
<td>1000</td>
<td>22 / 35 / 12</td>
<td>11 / 21 / 9</td>
</tr>
<tr>
<td>6</td>
<td>1000</td>
<td>85 / 62 / 76</td>
<td>78 / 149 / 70</td>
</tr>
<tr>
<td>7</td>
<td>1000</td>
<td>67 / 105 / 35</td>
<td>67 / 105 / 35</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>170 / 336 / 86</td>
<td>73 / 179 / 71</td>
</tr>
<tr>
<td>9</td>
<td>1000</td>
<td>167 / 290 / 122</td>
<td>165 / 296 / 130</td>
</tr>
<tr>
<td>10</td>
<td>1000</td>
<td>86 / 184 / 84</td>
<td>93 / 199 / 91</td>
</tr>
<tr>
<td>11</td>
<td>1000</td>
<td>247 / 409 / 161</td>
<td>230 / 391 / 160</td>
</tr>
<tr>
<td>12</td>
<td>1000</td>
<td>33 / 63 / 26</td>
<td>36 / 62 / 22</td>
</tr>
<tr>
<td>13</td>
<td>1000</td>
<td>70 / 1313 / 60</td>
<td>67 / 1063 / 55</td>
</tr>
<tr>
<td>14</td>
<td>1000</td>
<td>19 / 42 / 18</td>
<td>22 / 49 / 21</td>
</tr>
<tr>
<td>15</td>
<td>1000</td>
<td>132 / 565 / 129</td>
<td>100 / 333 / 94</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>1311 / 3842 / 992</td>
<td>1078 / 3124 / 859</td>
</tr>
</tbody>
</table>

Total Time : 359 Sc 327 Sc

## Table (3) Comparison of algorithms for N=10000

<table>
<thead>
<tr>
<th>Test Fun</th>
<th>Dim n</th>
<th>FR-CG algorithm</th>
<th>SFR-CG algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>NOI / FGE / LIN</td>
<td>NOI / FGE / LIN</td>
</tr>
<tr>
<td>1</td>
<td>10000</td>
<td>37 / 65 / 25</td>
<td>32 / 60 / 24</td>
</tr>
<tr>
<td>2</td>
<td>10000</td>
<td>40 / 91 / 34</td>
<td>37 / 83 / 31</td>
</tr>
<tr>
<td>3</td>
<td>10000</td>
<td>44 / 95 / 39</td>
<td>39 / 85 / 30</td>
</tr>
<tr>
<td>4</td>
<td>10000</td>
<td>25 / 65 / 16</td>
<td>23 / 59 / 15</td>
</tr>
</tbody>
</table>
Spectral Fletcher-Reeves Algorithm... [34]

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>10000</td>
<td>23 / 37 / 13</td>
<td>23 / 37 / 13</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10000</td>
<td>104 / 199 / 93</td>
<td>61 / 116 / 53</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>10000</td>
<td>72 / 113 / 38</td>
<td>72 / 113 / 38</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>10000</td>
<td>77 / 174 / 73</td>
<td>70 / 70 / 68</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>10000</td>
<td>657 / 1158 / 500</td>
<td>579 / 1055 / 475</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10000</td>
<td>86 / 182 / 84</td>
<td>85 / 181 / 83</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>10000</td>
<td>49 / 91 / 41</td>
<td>41 / 75 / 33</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>10000</td>
<td>96 / 145 / 46</td>
<td>88 / 153 / 62</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>10000</td>
<td>112 / 2589 / 102</td>
<td>68 / 1380 / 64</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>10000</td>
<td>16 / 40 / 15</td>
<td>17 / 40 / 16</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>10000</td>
<td>221 / 6472 / 215</td>
<td>297 / 7498 / 290</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>1665/ 11516 / 1334</td>
<td>1533 / 11105 / 1294</td>
<td></td>
</tr>
</tbody>
</table>

Total Time : 7423 Sc 6157 Sc

References


[6] Kincella J. 'Course Notes For MS4327 Optimization'


Appendix

1. Extended Trigonometric Function

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

\[ x_0 = [0.2, 0.2, \ldots, 0.2]. \]

2. Extended Rosenbrock Function

\[ f(x) = \sum_{i=4}^{\frac{n}{2}} c(x_{2i} - x_{2i-1})^2 + (1 - x_{2i-1})^2, \]

\[ x_0 = [-1.2, 1, \ldots, -1.2, 1], c = 100 \]

3. Extended White & Host Function

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

\[ x_0 = [-1.2, 1, \ldots, -1.2, 1], c = 100 \]

4. Extended Penalty Function
\[ f(x) = \sum_{i=1}^{n-1} (x_i - 1)^2 + \left( \sum_{j=1}^{n} x_j^2 - 0.25 \right)^2, \]
\[ x_0 = [1, 2, \ldots, n] \]

5. Extended Himmelblau Function
\[ f(x) = \sum_{i=1}^{\frac{n}{2}} (x_{2i-1}^2 + x_{2i} - 1)^2 + (x_{2i-1} + x_{2i} - 7)^2, \]
\[ x_0 = [1, 1, \ldots, 1]. \]

6. Generalized PSC1 Function
\[ f(x) = \sum_{i=1}^{\frac{n}{2}} (x_i^2 + x_{i+1}^2 + x_i x_{i+1})^2 + \sin^2(x_i) + \cos^2(x_i), \]
\[ x_0 = [3, 0.1, \ldots, 3, 0.1] \]

7. Extended PSC1 Function
\[ f(x) = \sum_{i=1}^{\frac{n}{2}} (x_{2i-1}^2 + x_{2i}^2 + x_{2i-1} x_{2i})^2 + \sin^2(x_{2i-1}) + \cos^2(x_{2i}), \]
\[ x_0 = [3, 0.1, \ldots, 3, 0.1]. \]

8. Extended Powell Function
\[ f(x) = \sum_{i=1}^{\frac{n}{4}} (x_{4i-3} + 10 x_{4i-1})^2 + 5 (x_{4i-1} - x_{4i})^2 + (x_{4i-2} - 2 x_{4i-1})^4 + 10 (x_{4i-3} - x_{4i})^4, \]
\[ x_0 = [3, -1, 0.1, \ldots, 3, -1, 0.1] \]

9. Full Hessian FH2 Function
\[ f(x) = (x_1 - 5)^2 + \sum_{i=2}^{n} (x_1 + x_2 + \ldots + x_i - 1)^2, \]
\[ x_0 = [0.01, 0.01, \ldots, 0.01]. \]

10. Extended Maratos function (c=100)
\[ f(x) = \sum_{i=1}^{\frac{n}{2}} x_{2i-1} + c (x_{2i-1}^2 + x_{2i}^2 - 1)^2, \]
\[ x_0 = [1.1, 0.1, \ldots, 1.1, 0.1]. c = 100 \]

11. NONDQUAR Function (CUTE)
Spectral Fletcher-Reeves Algorithm...

\[ f(x) = (x_1 - x_2)^2 + \sum_{i=1}^{n-2} (x_i + x_{i+1})^4 + (x_{n-1} + x_n)^2, \]

\[ x_0 = [1., -1., \ldots, 1., -1.]. \]

12. DQDRTIC function (CUTE)

\[ f(x) = \sum_{i=1}^{n-2} (x_i^2 + cx_i^2 + dx_i^2), \]

\[ c = 100., d = 100. \]

\[ x_0 = [3., 3., \ldots, 3.]. \]

13. DIXMAANA-DIXMAANL Function

\[ f(x) = 1 + \sum_{i=1}^{n} \alpha_i n^{-1} k_1 + \sum_{i=1}^{n-1} \beta (x_{i+1} + x_i)^2 n^{-1} k_2 + \sum_{i=1}^{2m} \gamma (x_{i+m} n^{-1} k_3 + \sum_{i=1}^{m} \delta x_{i+2m} n^{-1} k_4 \]

14. Almost Pertubed Quadratic Function

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

\[ x_0 = [0.5, 0.5, \ldots, 0.5] \]

15. Staircase 2 Function

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

\[ x_0 = [0, 0, \ldots, 0]. \]