The Extended CG Method for Non-Quadratic Models
Abbas Y. Al-Bayati Basim A. Hassan Sawsan S. Ismael
profabbasalbayati@yahoo.com basimabas39@gmail.com sawsan.sami14@yahoo.com
College of Computer Sciences and Mathematics University of Mosul, Mosul, Iraq
Received on: 25 / 3 / 2007 Accepted on: 4 / 11 / 2007
ABSTRACT
This paper investigates an interleaved algorithm which combines between the extended conjugate gradient with the hybrid method of Touati-Storey. This combined algorithm is based on the exact line search to solve a number of non-linear test functions with different dimensions. Experimental results indicate that the modified algorithm is more efficient than the original Sloboda algorithm.
Keywords: A new conjugate gradient method, Numerical Results and Conclusions.

1. Introduction
The problem to be solved by the Sloboda algorithm is to calculate the least value of a general differentiable function of several variables. Let n be the number of variables, x be the vector of variables and q(x) be the objective function and g(x) be the gradient of q(x). i.e.

$$g(x) = \nabla q(x) \tag{1}$$

Conjugate gradient algorithm does not require any explicit second derivatives and it is an iterative method. The sequences of points $x_1, x_2, x_3, \ldots$ are calculated by the successive iteration procedure and it should converge to the point in the space of the variable at which $q(x)$ is least.

The conjugate gradient algorithm was first applied to the general unconstrained minimization problem by Fletcher & Reeves [7]. However, now there are several versions of the algorithm for this calculation. Let $x_1$ be a given starting point in the space of the variable and let i denote the number of the current iteration starting with i=1. the iteration requires the gradient:

$$g_i = g(x_i) \tag{2}$$

if $i = 1$, let $d_i$ be the steepest descent direction,

$$d_i = - g_i$$

otherwise, for $i > 1$, we apply the formula:
\[ d_i = -g_i + \beta_i d_{i-1} \]  
\[ \beta_i = \frac{y_i^T g_{i+1}}{y_i^T d_i} \]  
\[ y_i = g_{i+1} - g_i \]

where \( \beta_i \) and \( y_i \) have the values:

\[ \text{The vector norms being Euclidian and by searching for the least value of } f(x) \text{ from } x_i \text{ along the direction } d_i \text{ we can obtain the vector } x_{i+1}: \]
\[ x_{i+1} = x_i + \lambda_i d_i \]  

Where \( \lambda_i \) is the value of \( \lambda \) that minimizes the function of one variable.

\[ \phi(\lambda) = f(x_i + \lambda d_i) \]

This complete the value of the iteration, and another one is begun if \( f(x_{i+1}) \) or \( g_{i+1} \) is not sufficiently small, [10].

Let \( A \) denote a symmetric and positive definite \( i \times i \) matrix. For \( x \in \mathbb{R}^i \), we define:
\[ q(x) = \frac{1}{2} x^T A x + b^T x + c \]

Let \( F : \mathbb{R} \rightarrow \mathbb{R}^i \) denote a strictly monotonic increasing function and define:
\[ f(x) = F(q(x)) \]

Such a function is called an extended quadratic function, Spedicato [12].

When a minimization algorithm is applied to \( f \), the \( i^{\text{th}} \) iterate is denoted by \( x_i \), the corresponding function value by \( f_i \) and its gradient by \( G_i \), the function value and gradient value of \( q \) are denoted by \( q_i \) & \( g_i \), respectively and the derivative of \( F_i \) at \( q_i \) is denoted by \( F'_i \). we note that \( G_i = F'_i g_i \) and define \( \rho_i = c_i / c_{i+1} \) for \( i \), where \( c_i = F'_i \), Al-Assady and Al-Bayati [4]

2. Non-Quadratic Sloboda Method

Sloboda [11] proposed a generalized conjugate gradient algorithm for minimizing a strictly convex function of the general form:
\[ f(x) = F(q(x)) ; \quad \frac{df}{dq} > 0 \]

This algorithm is as follows:

**Algorithm 1:**

Step 1: Set \( i = 1 ; d_1 = -G_1 ; G_i = g_i^* \).

Step 2: Compute \( \lambda \) by ELS & set \( x_{i+1} = x_i + \lambda d_i \).

Step 3: Compute \( g_{i+1} = g(x - \frac{\lambda d}{2}) \)

Step 4: Test for convergence, if achieved stop. if not continue.

Step 5: If \( i=0 \) nod(n) go to Step 1, else continue.
Algorithm 1 terminates after \( i \) iterations in the case of a nonlinear scaled quadratic function using ELS.

A more general scaling has been considered by Spedicato [12], such a scaling transforms \( F \) into a new function where (9), is satisfied. He shows that the sequence of points generated is invariant with respect to nonlinear scaling if:

\[
\Phi = \frac{\partial F}{\partial q} v = \lambda_i^2 g_i^T H_i g
\]

However, this type of scaling also uses functions for which the analytic form is known apriori.

Al-Bayati [1] introduced another family of self-scaling VM-methods given by:

\[
H_{i+1} = H_i + (\mu + \frac{a}{b}) v_i y_i^T - \frac{\theta - 1}{a} H_i y_i y_i^T H_i - \frac{\theta}{b} (H_i y_i v_i y_i^T + v_i y_i^T H_i)
\]

where \( \theta \) is again a free parameter; \( \mu = \frac{1}{M} \), and \( a = y_i^T H_i y_i \), \( b = v_i^T y_i \)

\[
M = \begin{pmatrix} c \\ b \end{pmatrix} (1 - \beta) ; 0 \leq \beta \leq 1
\]

where \( \varepsilon = \Phi \)

\[
H_{i+1} = H_i + (\mu + \frac{a}{b}) v_i y_i^T - \frac{1}{b} (H_i y_i v_i y_i^T + v_i y_i^T H_i)
\]

If an estimate of the inverse Hessian is maintained (rather than an estimate of the Hessian itself which is sometime preferred) then there is a strong motivation for choosing \( \beta = 0 \) in (11 a), namely, that \( H^1 \) is not required, this gives \( \mu = \frac{a}{b} \).

However, it is possible to generalize Al-Bayati’s family of self-scaling VM-updates (12) to be invariant to a nonlinear scaling by the following algorithm, Al-Bayati [2].

- **Algorithm 2:**

Step 1: Set \( i = 1; H_1 = I; d_1 = -H_1 G_1 ; G_1 = g^*_i \).

Step 2: Compute \( x_{i+1} = x_i + \lambda_i d_i \); \( \lambda_i \) determined by ELS.

Step 3: Set \( \frac{g_i^*}{w} \rightarrow \frac{g_i}{w}^* \); \( w = \frac{d_i^T g_i^*}{d_i^T g_i} \) and \( g_i + \frac{\lambda}{2} = g(x_i - \frac{1}{2} \lambda d) \).

Step 4: Test for convergence; if not continue.
Step 5: Update the matrix \( H \) using (12).
Step 6: Compute \( d_{i+1} = -H_i g_{i+1}^* \).
Step 7: Set \( i = i + 1 \) and go to Step 2.

In this section we shall describe also another algorithm which effectively interleaves CG & VM-steps. It is also related to one given originally by Buckley [6], but our implementation differs in that we use the scaled quadratic model instead of the quadratic itself.

CG-algorithm 1. and the generalized VM-algorithm 2. The objective here is to show that, using Al-Bayati’s self-scaling VM-update (12), the sequence of the generating points is the same in the generalized CG-algorithm 1.

Before making a few more observations we shall outline briefly the proposed strategy for the interleaved generalized CG-VM method Al-Bayati [2].

- **Algorithm 3:**

Let \( f \) be a non linear scaling of the quadratic function \( f \); given \( x_1 \) and a matrix \( H_1 = I \); set \( G_i = g_i^* ; i = 1 \) and \( t = 1 \) initially, \( k \) is the iteration index.

Step 1: Set \( d_t = -H_t g_t^* \) \hspace{1cm} (13)

Step 2: For \( k = t, t+1, t+2, \ldots \) iterate with
\[
x_{k+1} = x_k + \lambda_k d_k ,
\]
\[
\beta_k = \frac{(g_{k+1}^T H_k \bar{y}_k)}{d_k^T \bar{y}_k} ,
\]
\[
d_{k+1} = -H_k g_{k+1}^* + \beta_k d_k ,
\]
\[
\beta_k = \frac{\bar{y}_k^T g_{k+1}^*}{d_k^T \bar{y}_k} , \quad w = \frac{d_t^T g_i^*}{d_t^T g_{i+1}^*}
\]
\[
g_{k+1}^* = w_k g_{k+1}^* - g_k
\]
\[
g_{k+1/2} = g(x_k - \frac{1}{2} \lambda_k d_k )
\]
\[
\bar{y} = g_{k+1}^* - g_k
\]

Here \( i \) is the index of the matrix updated only at restart steps and \( k \) is the index of iteration and the algorithm is not converged, until a restart is indicated.

Step 3: If a restart is indicated, namely that the Powell [9] restarting criterion is satisfied, i.e.
\[
|g_{k+1}^* T g_k| \geq 0.2 |g_{k+1}^* T g_{k+1}| , \hspace{1cm} (14)
\]

Then reset \( t \) to the current \( k \), update \( H_t \) by:
\[
H_{i+1} = H_t - (H_t y_i v_i^T)/(b_i^T + v_i [(2a_i/b_i^2)v_i - (H_t y_i/b_i)]^T) .
\]

where \( a_i = y_i^T H_i y_i \) and \( b_i = v_i^T y_i \)

Step 4: Replace \( i \) by \( i + 1 \) and repeat from (13).
3. Hybrid Conjugate Gradient Method

Despite the numerical superiority of PR-method over FR-method the later has better theoretical properties than the formal see Al-Baali [3]. Under certain conditions FR-method can be shown to have global convergence with exact line search Powell [10] and also with inexact line search satisfying the strong Wolf-Powell condition. This anomaly leads to speculation on the best way to choose $\beta_i$.

Touati – Ahmed and Storey in (12) proposed the following hybrid method:

Step 1: if $\lambda \|g_{i-1}\|^2 \leq (2m)^{i-1}$, with $\frac{1}{2} > m > \eta$ and $\lambda > 0$ go to Step 2. Otherwise, Set $\beta_i = 0$.

Step 2: If $\beta_i^{PR} < 0$, Set $\beta_i = \beta_i^{PR}$, otherwise go to Step 3.

Step 3: If $\beta_i^{PR} \leq (1/2m)\left(\frac{\|g_{i-1}\|^2}{\|g_i\|^2}\right)$, with $m = \eta$, set $\beta_i = \beta_i^{PR}$. Otherwise Set $\beta_i = \beta_i^{PR}$.

Here $m$, $\eta$ and $\lambda$ user supplied parameters. This hybrid was shown to be globally convergence under both exact and inexact line searches and to be quite competitive with PR- and FR-methods.

4. New hybrid algorithm (Algorithm 4):

Step 1: Set $d_1 = -H_1 g_1^*$, $i = 1$, $t = 1$, $k$ is the index of iterations.

Step 2: For $k = t$, $t+1$, ..., iterate with

$x_{k+1} = x_k + \lambda_k d_k$,

and

$\beta_k = \frac{(g_{k+1}^T H_k \bar{y}_k)}{d_k^T \bar{y}_k}$,

Where $i$ is the index of the matrix updated only at restart steps.

Step 3: If $\lambda \|g_i\|^2 \leq (2m)^{k+1}$ with $\frac{1}{2} > m > \eta$ go to Step 1. Otherwise set $\beta_i = 0$.

Step 4: If $\beta_i^{PR} < 0$ set $\beta_i = \beta_i^{PR}$, otherwise go to Step 5.

Step 5: If $B_i^{PR} \leq \left(\frac{1}{2}m\right)\|g_{i-1}\|^2/\|g_i\|^2$ with $m > \eta$, Set $B_i = B_i^{PR}$, otherwise set $B_i = B_i^{FR}$.

Step 6: Compute $d_{k+1} = -H_k g_{k+1}^*$, where $g_{k+1}^* = w_k g_{k+1/2}^* - g_k^*$.

Step 7: If a restart is indicated, namely that the Powell, restarting criterion is satisfied, i.e.: $|g_{k+1}^T g_k | \geq 0.2 |g_{k}^T g_k |$, then reset $t$ to the current $k$, update $H_k$ by:

$H_{k+1} = H_k - \left(\frac{H_k y_i v_i^T}{b_i} + v_i [\frac{2 a_i}{b_i^2}] v_i - \left(\frac{H_k y_i}{b_i}\right)^T\right)$

Step 8: Replace $i$ by $i+1$ and repeat from (13).

5. Conclusions and Numerical results:

Several standard test functions were minimized (2 \leq n \leq 400) to compare the proposed algorithm with standard Sloboda algorithm which are coded in double
precision Fortran 90. The proposed hybrid algorithm needs matrix calculation for 400×400, this is the approximately the latest range for this computation of the matrix. The numerical results are obtained on personal Pentium IV Computer. The compete set of results are given in tables 5.1 and 5.2.

The linear search routine used was a cubic interpolation which use function and gradient values and it is adaptation of the routine published by Bunday [5].

We tabulate for all the algorithms; the number of functions evaluations (NOF) and the number of iterations (NOI). Overall totals are also given for NOF and NOI with each algorithm.

Table 5.1 gives the comparison between the standard Sloboda algorithm and the proposed algorithm. Table 5.2 indicates that the suggested algorithm is more efficient than the standard Sloboda algorithm. Namely, there are an improvement of about (53 %) in both NOI and NOF according to our selected group of test functions.

### Table 5.1
Comparison between Sloboda method and the proposed hybrid method for $2 \leq n \leq 400$

<table>
<thead>
<tr>
<th>Test function</th>
<th>Dimension</th>
<th>Sloboda method</th>
<th>Hybrid method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>NOI</td>
<td>NOF</td>
</tr>
<tr>
<td>ROSEN</td>
<td>2</td>
<td>33</td>
<td>85</td>
</tr>
<tr>
<td>BEAL</td>
<td>2</td>
<td>10</td>
<td>26</td>
</tr>
<tr>
<td>DIXON</td>
<td>2</td>
<td>6</td>
<td>17</td>
</tr>
<tr>
<td>NON.DI</td>
<td>10</td>
<td>16</td>
<td>49</td>
</tr>
<tr>
<td>ROSEN</td>
<td>10</td>
<td>16</td>
<td>49</td>
</tr>
<tr>
<td>POWELL</td>
<td>4</td>
<td>51</td>
<td>130</td>
</tr>
<tr>
<td>WOOD</td>
<td>4</td>
<td>33</td>
<td>80</td>
</tr>
<tr>
<td>SHALLO</td>
<td>4</td>
<td>8</td>
<td>21</td>
</tr>
<tr>
<td>BEAL</td>
<td>40</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>POWELL-3</td>
<td>40</td>
<td>17</td>
<td>37</td>
</tr>
<tr>
<td>POWELL</td>
<td>40</td>
<td>60</td>
<td>165</td>
</tr>
<tr>
<td>ROSEN</td>
<td>60</td>
<td>17</td>
<td>48</td>
</tr>
<tr>
<td>WOOD</td>
<td>60</td>
<td>102</td>
<td>217</td>
</tr>
<tr>
<td>WOLFE</td>
<td>80</td>
<td>52</td>
<td>105</td>
</tr>
<tr>
<td>POWELL</td>
<td>80</td>
<td>90</td>
<td>260</td>
</tr>
<tr>
<td>WOOD</td>
<td>80</td>
<td>98</td>
<td>209</td>
</tr>
<tr>
<td>BEAL</td>
<td>100</td>
<td>8</td>
<td>20</td>
</tr>
<tr>
<td>POWELL</td>
<td>100</td>
<td>112</td>
<td>384</td>
</tr>
<tr>
<td>WOLFE</td>
<td>100</td>
<td>53</td>
<td>107</td>
</tr>
<tr>
<td>WOOD</td>
<td>400</td>
<td>103</td>
<td>213</td>
</tr>
<tr>
<td>WOLF</td>
<td>400</td>
<td>53</td>
<td>107</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td>946</td>
<td>2349</td>
</tr>
</tbody>
</table>

### Table 5.2
Performance of the new algorithm in relation to Sloboda’s algorithm

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Sloboda</th>
<th>New Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOI</td>
<td>100</td>
<td>47.35</td>
</tr>
</tbody>
</table>
The Extended CG Method for Non-Quadratic Models

<table>
<thead>
<tr>
<th>NOF</th>
<th>100</th>
<th>47.72</th>
</tr>
</thead>
</table>

All the algorithms terminated when $|f - f_{\text{min}}| < 5 \times 10^{-10}$.

Appendix

These test function are from general literature [8].

1. Generalized Powell Function

$$F(X) = \sum_{i=2}^{n/4} [(x_{4i-3} - 10x_{4i-2})^2 + 5(x_{4i-1} - x_{4i})^2 + (x_{4i-2} - 2x_{4i-1})^4 + 10(x_{4i-3} - x_{4i})^2]$$

$x_o = (3, -1, 0, 1, \ldots)^T$

2. Generalized Wood Function

$$F(x) = \sum_{i=2}^{n/4} [100(x_{4i-2} - x_{4i-3})^2 + (1 - x_{4i-3})^2 + 90(x_{4i} - x_{4i-1})^4 + (1 - x_{4i-1})^2 + 1.0]$$

$x_o = (-3, -1, -3, -1, \ldots)^T$

3. Non-diagonal Functions:

$$F(x) = \sum_{i=2}^{n/2} [100(x_i - x_i^2)^2 + (1 - x_i)^2]$$

$x_o = (-1, \ldots)^T$

4. Generalized Dixon Function

$$F(x) = (1 - x_1)^2 + (1 - x_2)^2 + \sum_{i=2}^{9} (x_i - x_{i+1})$$

$x_o = (-1, \ldots)^T$

5. Wolfe Functions:

$$F(X) = (-x(3 - x_i/2) + 2x_2 - 1)^2 + \sum_{i=1}^{n/4} (x_{4i} - x_i(3 - x_i/2) + 2x_{i+1} - 1)^2 + (x_{n/4} - x_{n}(3x_{n/2} - 1)^2$$

$x_o = (-1, \ldots)^T$

6. Shallo Function

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

$x_o = (-2, -2, \ldots)^T$

7. Generalized Rosenbrock Function

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

$x_o = (-1, 2, 1, \ldots)^T$

8. Beale Function

$$F(x) = (1.5 - x_i(1 - x_i^2)) + (2.25 - x_i(1 - x_i^2)) + (2.625 - x_i(1 - x_i^3))^2$$

$x_o = (0, 0)^T$
REFERENCES


