Accelerated scaled memoryless BFGS preconditioned conjugate gradient algorithm for unconstrained optimization

NECULAI ANDREI
Research Institute for Informatics,
Center for Advanced Modeling and Optimization,
8-10, Averescu Avenue, Bucharest 1, Romania
and
Academy of Romanian Scientists,
54, Splaiul Independentei, Bucharest 5, Romania
E-mail: nandrei@ici.ro

Abstract. An accelerated scaled memoryless BFGS preconditioned conjugate gradient algorithm for solving unconstrained optimization problems is presented. The basic idea is to combine the scaled memoryless BFGS method and the preconditioning technique in the frame of the conjugate gradient method. The preconditioner, which is also a scaled memoryless BFGS matrix, is reset when the Beale-Powell restart criterion holds. The parameter scaling the gradient is selected as spectral gradient. For steplength computation the method uses the advantage that the step lengths in conjugate gradient algorithms may differ from 1 by two order of magnitude and tend to vary in an unpredictable manner. Thus, we suggest an acceleration scheme able to improve the efficiency of the algorithm. Under common assumptions, the method is proved to be globally convergent. It is shown that for uniformly convex functions the convergence of the accelerated algorithm is still linear, but the reduction in the function values is significantly improved. In very mild conditions it is shown that, for strongly convex functions, the algorithm is globally convergent. Computational results for a set consisting of 750 unconstrained optimization test problems show that this new accelerated scaled conjugate gradient algorithm substantially outperforms known conjugate gradient methods.

Keywords: Unconstrained optimization; conjugate gradient method; spectral gradient method; Wolfe line search; BFGS preconditioning, acceleration
49M07; 49M10; 90C06; 65K

1. Introduction

In this paper we consider the following unconstrained optimization problem:

$$
\min f(x)
$$

(1.1)

where \( f : \mathbb{R}^n \rightarrow \mathbb{R} \) is continuously differentiable and its gradient is available. We are interested in elaborating an algorithm for solving large-scale cases for which the Hessian of \( f \) is either not available or requires a large amount of storage and computational costs. The paper presents a conjugate gradient algorithm based on a combination of the scaled memoryless BFGS method and the preconditioning technique [3,4]. For general nonlinear functions a good preconditioner is any matrix that approximates \( \nabla^2 f(x^*)^{-1} \), where \( x^* \) is the solution of (1.1). In this algorithm the preconditioner is a scaled memoryless BFGS matrix which is reset when the Powell restart criterion holds. The scaling factor in the preconditioner is selected as spectral gradient [33].

The algorithm uses the conjugate gradient direction where the famous parameter \( \beta_k \) is obtained by equating the conjugate gradient direction with the direction corresponding to the Newton method [5,6]. Thus, we get a general formula for the direction computation, which could be particularized to include the Polak-Ribiére [29] and Polyak [30] and the
Fletcher and Reeves [18] conjugate gradient algorithms, the spectral conjugate gradient (SCG) by Birgin and Martínez [11] or the algorithm of Dai and Liao [13], for \( t = 1 \). This direction is then modified in a canonical manner as it was considered earlier by Oren and Luenberger [26], Oren and Spedicato [27], Perry [28] and Shanno [34, 35], by means of a scaled, memoryless BFGS preconditioner placed into the Beale-Powell restart technology. The scaling factor is computed in a spectral manner based on the inverse Rayleigh quotient, as suggested by Raydan [33]. The method could be considered as an extension of the spectral conjugate gradient (SCG) by Birgin and Martínez [11] or of a variant of the conjugate gradient algorithm by Dai and Liao [13] (for \( t = 1 \)) suggested to overcome the lack of positive definiteness of the matrix defining their search direction.

In [25] Jorge Nocedal articulated a number of open problems in conjugate gradient algorithms. One of them focuses on the step length. Intensive numerical experiments with conjugate gradient algorithms proved that the step length may differ from 1 up to two orders of magnitude, being larger or smaller than 1, depending on how the problem is scaled. Moreover, the sizes of the step length tend to vary in a totally unpredictable way. This is in sharp contrast with the Newton and quasi-Newton methods, which usually admit the unit step length for most of the iterations and require only very few function evaluations for step length determination. Therefore, in this paper we take the advantage of this behavior of the step lengths in conjugate gradient algorithms and present an acceleration scheme, which modify the step length in such a manner to improve the reduction in functions values.

The paper is organized as follows: In section 2 we present the scaled conjugate gradient algorithm BFGS preconditioned. The algorithm performs two types of steps: a standard one in which a double quasi-Newton updating scheme is used and a restart one where the current information is used to define the search direction. The convergence of the algorithm for strongly convex functions is proved in section 3. In section 4 we present an acceleration scheme of the algorithm. The idea of this computational scheme is to take advantage that the step lengths \( \alpha_k \) in conjugate gradient algorithms are very different from 1. Therefore, we suggest we modify \( \alpha_k \) in such a manner as to improve the reduction of the function values along the iterations. In section 5 we present the ASCALCG algorithm and we prove that for uniformly convex functions the convergence of the accelerated algorithm is still linear, but the reduction in function values is significantly improved. Finally, in section 6 we present computational results on a set of 750 unconstrained optimization problems from the CUTE [12] collection along with some other large-scale unconstrained optimization problems presented in [1]. The Dolan-Moré [17] performance profiles of ASCALCG versus some known conjugate gradient algorithms including SCALCG by Andrei [3,4,5,6], CG_DESCENT by Hager and Zhang [19,20,21], LBFGS by Liu and Nocedal [23], truncated Newton TN by Nash [24] and the accelerated conjugate gradient with modified secant condition ACGMSEC [10] prove that ASCALCG is top performer among these algorithms.

2. Scaled Conjugate Gradient Method

The algorithm generates a sequence \( x_k \) of approximations to the minimum \( x^* \) of \( f \), in which

\[
x_{k+1} = x_k + \alpha_k d_k, \tag{2.1}
\]

\[
d_{k+1} = -\theta_{k+1} g_{k+1} + \beta_k s_k, \tag{2.2}
\]

where \( g_k = \nabla f(x_k) \), \( \alpha_k \) is selected to minimize \( f(x) \) along the search direction \( d_k \), \( \beta_k \) is a scalar parameter, \( s_k = x_{k+1} - x_k \) and \( \theta_{k+1} \) is a parameter to be determined. The iterative process is initialized with an initial point \( x_0 \) and \( d_0 = -g_0 \).

Observe that if \( \theta_{k+1} = 1 \), then we get the classical conjugate gradient algorithms according to the value of the scalar parameter \( \beta_k \). On the other hand, if \( \beta_k = 0 \), then we
get another class of algorithms according to the selection of the parameter \( \theta_{k+1} \). Considering \( \theta_{k+1} = 0 \), there are two possibilities for \( \theta_{k+1} \): a positive scalar or a positive definite matrix. If \( \theta_{k+1} = 1 \), then we have the steepest descent algorithm. If \( \theta_{k+1} = \nabla^2 f(x_k) \), or an approximation of it, then we get the Newton or the quasi-Newton algorithms, respectively. Therefore, we see that in the general case, when \( \theta_{k+1} \neq 0 \) is selected in a quasi-Newton manner, and \( \beta_k \neq 0 \), (2.2) represents a combination between the quasi-Newton and the conjugate gradient methods. However, if \( \theta_{k+1} \) is a matrix containing some useful information about the inverse Hessian of function \( f \), we are better off using \( \theta_{k+1} \) since the addition of the term \( \beta_k s_k \) in (2.2) may prevent the direction from being a descent direction unless the line search is sufficiently accurate. Therefore, in this paper we shall consider \( \theta_{k+1} \) as a positive scalar which contains some useful information to the inverse Hessian of function \( f \).

Our motivation to get a good algorithm for solving (1.1) is to choose the parameter \( \beta_k \) in (2.2) in such a way so that for every \( k \geq 1 \) the direction \( d_k \) given by (2.2) be the Newton direction. Therefore, from the equality

\[
-\nabla^2 f(x_k) g_{k+1} = -\theta_{k+1} g_{k+1} + \beta_k s_k,
\]

we get:

\[
\beta_k = \frac{s_k^T \nabla^2 f(x_k) \theta_{k+1} g_{k+1} - s_k^T g_{k+1}}{s_k^T \nabla^2 f(x_k+1)}.
\]

The salient point with this formula for \( \beta_k \) computation is the presence of the Hessian. Observe that if the line search is exact we get the scaled Daniel method [16]. For large-scale problems, choices for the update parameter that do not require the evaluation of the Hessian matrix are often preferred in practice to the methods that require the Hessian. As we know, for quasi-Newton methods an approximation matrix \( B_k \) to the Hessian \( \nabla^2 f(x_k) \) is used and updated so that the new matrix \( B_{k+1} \) satisfies the secant condition \( B_{k+1} s_k = y_k \), where \( y_k = g_{k+1} - g_k \). Therefore, in order to have an algorithm for solving large-scale problems we can assume that the pair \( (s_k, y_k) \) satisfies the secant condition. In this case, Zhang, Deng and Chen [39] proved that if \( \|s_k\| \) is sufficiently small, then

\[
s_k^T \nabla^2 f(x_k) s_k - s_k^T y_k = O(\|s_k\|^3).
\]

Therefore, using this assumption we get:

\[
\beta_k = (\theta_{k+1} y_k - s_k)^T g_{k+1}.
\]

Birgin and Martínez [11] arrived at the same formula for \( \beta_k \), but using a geometric interpretation of quadratic function minimization. The direction corresponding to \( \beta_k \) given in (2.5) is as follows:

\[
d_{k+1} = -\theta_{k+1} g_{k+1} + \frac{(\theta_{k+1} y_k - s_k)^T g_{k+1}}{y_k^T s_k} s_k.
\]

The following particularizations are obvious. If \( \theta_{k+1} = 1 \), then (2.6) is the direction considered by Perry [28]. At the same time we see that (2.6) is the direction given by Dai and Liao [13] for \( t = 1 \), obtained this time by an interpretation of the conjugacy condition. Additionally, if \( s_j^T g_{j+1} = 0 \), \( j = 0,1, \ldots, k \), then from (2.6) we get:

\[
d_{k+1} = -\theta_{k+1} g_{k+1} + \frac{\theta_{k+1} y_k g_{k+1}}{a_k \theta_{k+1} g_k^T g_k} s_k,
\]

which is the direction corresponding to a generalization of the Polak and Ribièrè formula. Of course, if \( \theta_{k+1} = \theta = 1 \) in (2.7), we get the classical Polak and Ribièrè formula [29]. If
\[ s_j^T g_{j+1} = 0, \quad j = 0, 1, \ldots, k, \] and additionally the successive gradients are orthogonal, then from (2.6) we get:

\[ d_{k+1} = -\theta_{k+1} g_{k+1} + \frac{\theta_{k+1}}{\alpha_k} g_k^T g_k s_k, \tag{2.8} \]

which is the direction corresponding to a generalization of the Fletcher and Reeves formula [18]. Therefore, (2.6) is a general formula for direction computation in a conjugate gradient manner including the classical Fletcher and Reeves [18], and Polak-Ribière and Polyak [29, 30] formulas.

There is a result by Shanno [34, 35] that says that the conjugate gradient method is precisely the BFGS quasi-Newton method for which the initial approximation to the inverse of the Hessian, at every step, is taken as the identity matrix. The extension to the scaled conjugate gradient is very simple. Using the same methodology as considered by Shanno [34] we get the following direction:

\[ d_{k+1} = -\theta_{k+1} g_{k+1} + \theta_{k+1} \left( g_{k+1}^T y_k \right) y_k - \left[ 1 + \theta_{k+1} \left( g_{k+1}^T y_k \right) y_k^T y_k \right] g_k^T y_k - \theta_{k+1} g_{k+1}^T g_{k+1} y_k \right] s_k, \tag{2.8} \]

involving only 4 scalar products. Again observe that if \( g_{k+1}^T s_k = 0 \), then (2.9) reduces to:

\[ d_{k+1} = -\theta_{k+1} g_{k+1} + \theta_{k+1} g_{k+1}^T y_k \]

Thus, in this case, the effect is simply one of multiplying the Hestenes and Stiefel [22] search direction by a positive scalar.

In order to ensure the convergence of the algorithm (2.1), with \( d_{k+1} \) given by (2.9), we need to constrain the choice of \( \alpha_k \). We consider line searches that satisfy the Wolfe conditions [37, 38]:

\[ f(x_k + \alpha_k d_k) - f(x_k) \leq \rho \alpha_k g_k^T d_k, \tag{2.11} \]

\[ \nabla f(x_k + \alpha_k d_k)^T d_k \geq \sigma g_k^T d_k, \tag{2.12} \]

where \( 0 < \rho \leq \sigma < 1 \).

**Theorem 2.1.** Suppose that \( \alpha_k \) in (2.1) satisfies the Wolfe conditions (2.11) and (2.12), then the direction given by (2.9) is a descent direction.

**Proof:** Since \( d_0 = -g_0 \), we have \( g_0^T d_0 = -\|g_0\|^2 \leq 0. \) Multiplying (2.9) by \( g_{k+1}^T \), we have:

\[ g_{k+1}^T d_{k+1} = \frac{1}{(y_k^T s_k)^2} \left[ -\theta_{k+1} \|g_k\|^2 (y_k^T s_k)^2 + 2\theta_{k+1} (g_{k+1}^T y_k)(g_{k+1}^T s_k)(y_k^T s_k) \\
- (y_k^T s_k) - \theta_{k+1} (y_k^T y_k)(g_{k+1}^T s_k)^2 \right]. \]

Applying the inequality \( u^T v \leq \frac{1}{2} (\|u\|^2 + \|v\|^2) \) to the second term of the right hand side of the above equality, with \( u = (s_k^T y_k) g_{k+1} \) and \( v = (g_{k+1}^T s_k) y_k \), we get:

\[ g_{k+1}^T d_{k+1} \leq -\frac{(g_{k+1}^T s_k)^2}{y_k^T s_k}. \tag{2.13} \]

But, by Wolfe condition (2.12), \( y_k^T s_k > 0 \). Therefore, \( g_{k+1}^T d_{k+1} < 0 \) for every \( k = 0, 1, \ldots \).

Observe that the second Wolfe condition (2.12) is crucial for the descent character of direction (2.9). Besides, we see that the estimation (2.13) is independent of the parameter \( \theta_{k+1} \).

Usually, all conjugate gradient algorithms are periodically restarted. The Powell restarting procedure [31, 32] is to test if there is very little orthogonality left between the current gradient and the previous one. At step \( r \) when:
we restart the algorithm using the direction given by (2.9).

At step \( r \) we know \( s_r, y_r \) and \( \theta_{r+1} \). If (2.14) is satisfied, then a restart step is considered, i.e. the direction is computed as in (2.9). For \( k \geq r + 1 \), we consider the same philosophy used by Shanno [34, 35], where the gradient \( g_{k+1} \) is modified by a positive definite matrix which best estimates the inverse Hessian without any additional storage requirements, i.e. we compute:

\[
\begin{align*}
\nu &= \theta_{r+1} g_{k+1} - \theta_{r+1} \left( \frac{g_{k+1}^T s_r}{y_r^T s_r} \right) y_r \\
&\quad + \left[ 1 + \theta_{r+1} \frac{y_r^T y_r}{y_r^T s_r} \right] \frac{g_{k+1}^T s_r}{y_r^T s_r} - \theta_{r+1} \frac{g_{k+1}^T y_r}{y_r^T s_r} \right] s_r, \\
\end{align*}
\]

and

\[
\begin{align*}
w &= \theta_{r+1} y_k - \theta_{r+1} \left( \frac{y_k^T s_r}{y_r^T s_r} \right) y_r \\
&\quad + \left[ 1 + \theta_{r+1} \frac{y_r^T y_r}{y_r^T s_r} \right] \frac{y_k^T s_r}{y_r^T s_r} - \theta_{r+1} \frac{y_k^T y_r}{y_r^T s_r} \right] s_r,
\end{align*}
\]

involving 6 scalar products. With these, at any nonrestart step, the direction \( d_{k+1} \) for \( k \geq r + 1 \), is computed using a double update scheme as in Shanno [34]:

\[
d_{k+1} = -\nu + \left( \frac{g_{k+1}^T s_k}{y_k^T s_k} \right) w + \left( \frac{g_{k+1}^T w}{y_k^T s_k} \right) s_k - \left[ 1 + \frac{y_k^T w}{y_k^T s_k} \right] \left( \frac{g_{k+1}^T s_k}{y_k^T s_k} \right) s_k, 
\]

involving only 4 scalar products. Observe that \( y_k^T s_k > 0 \) is sufficient to ensure that the direction \( d_{k+1} \) given by (2.17) is well defined and it is always a descent direction.

Motivated by the efficiency of the spectral gradient method introduced by Raydan [33] and used by Birgin and Martínez [11] in their spectral conjugate gradient method for unconstrained optimization, in our algorithm \( \theta_{k+1} \) is defined as a scalar approximation to the inverse Hessian. This is given as the inverse of the Rayleigh quotient:

\[
s_k^T \left[ \int_0^1 \nabla^2 f(x_k + ts_k) dt \right] s_k / s_k^T s_k,
\]

i.e.

\[
\theta_{k+1} = \frac{s_k^T s_k}{y_k^T s_k}.
\]

The inverse of Rayleigh quotient lies between the smallest and the largest eigenvalue of the Hessian average \( \int_0^1 \nabla^2 f(x_k + ts_k) dt \). Again observe \( y_k^T s_k > 0 \) is sufficient to ensure that \( \theta_{k+1} \) in (2.18) is well defined.

### 3. Convergence Analysis for Strongly Convex Functions

Throughout this section we assume that \( f \) is strongly convex and Lipschitz continuous on the level set

\[
S = \{ x \in \mathbb{R}^n : f(x) \leq f(x_0) \}.
\]
That is, there exists constants $\mu > 0$ and $L$ such that
\[
(\nabla f(x) - \nabla f(y))^T (x - y) \geq \mu \|x - y\|^2 \tag{3.2}
\]
and
\[
\|\nabla f(x) - \nabla f(y)\| \leq L \|x - y\|. \tag{3.3}
\]
for all $x$ and $y$ from $S$. For the convenience of the reader we include here the following lemma (see [20]).

**Lemma 3.1.** Assume that $d_k$ is a descent direction and $\nabla f$ satisfies the Lipschitz condition
\[
\|\nabla f(x) - \nabla f(x_k)\| \leq L \|x - x_k\| \tag{3.4}
\]
for every $x$ on the line segment connecting $x_k$ and $x_{k+1}$, where $L$ is a constant. If the line search satisfies the second Wolfe condition (2.12), then
\[
\alpha_k \geq \frac{1 - \sigma}{L} \frac{g_k^T d_k}{\|d_k\|^2}. \tag{3.5}
\]

**Proof:** Subtracting $g_k^T d_k$ from both sides of (2.12) and using the Lipschitz condition we have
\[
(\sigma - 1)g_k^T d_k \leq (g_{k+1} - g_k)^T d_k \leq L\alpha_k \|d_k\|^2. \tag{3.6}
\]
Since $d_k$ is a descent direction and $\sigma < 1$, (3.5) follows immediately from (3.6). $\blacksquare$

Therefore, satisfying the Wolfe line search conditions $\alpha$ is bounded away from zero, i.e. there exists a positive constant $\omega$, such that $\alpha \geq \omega$.

**Lemma 3.2.** Assume that $\nabla f$ is strongly convex and Lipschitz continuous on $S$. If $\theta_{k+1}$ is selected by spectral gradient, then the direction given by (2.9) satisfies:
\[
\|d_{k+1}\| \leq \left(\frac{2}{\mu} + \frac{2L}{\mu^2} + \frac{L^2}{\mu^3}\right)\|g_{k+1}\|. \tag{3.7}
\]

**Proof:** By Lipschitz continuity (3.3) we have
\[
\|y_k\| = \|g_{k+1} - g_k\| = \|\nabla f(x_k + \alpha_k d_k) - \nabla f(x_k)\| \leq L\alpha_k \|d_k\| = L\|s_k\|. \tag{3.8}
\]
On the other hand, by strong convexity (3.2)
\[
y_k^T s_k \geq \mu \|s_k\|^2. \tag{3.9}
\]
Selecting $\theta_{k+1}$ as in (2.18), it follows that
\[
\theta_{k+1} = \frac{s_k^T s_k}{y_k^T s_k} \leq \frac{\|s_k\|^2}{\mu \|s_k\|^2} = \frac{1}{\mu}. \tag{3.10}
\]
Now, using the triangle inequality and the above estimates (3.8)-(3.10), after some algebra on $\|d_{k+1}\|$, where $d_{k+1}$ is given by (2.9), we get (3.7). $\blacksquare$

The convergence of the scaled conjugate gradient algorithm when $f$ is strongly convex is given by

**Theorem 3.1.** Assume that $f$ is strongly convex and Lipschitz continuous on the level set $S$. If at every step of the conjugate gradient (2.1) with $d_{k+1}$ given by (2.9) and the step length $\alpha_k$ selected to satisfy the Wolfe conditions (2.11) and (2.12), then either $g_k = 0$ for some $k$, or $\lim_{k \to \infty} g_k = 0$.

**Proof:** Suppose $g_k \neq 0$ for all $k$. By strong convexity we have
\[ y_k^T d_k = (g_{k+1} - g_k)^T d_k \geq \mu \alpha_k \|d_k\|^2. \]  
By theorem 2.1, \( g_k^T d_k < 0 \). Therefore, the assumption \( g_k \neq 0 \) implies \( d_k \neq 0 \). Since \( \alpha_k > 0 \), from (3.11) it follows that \( y_k^T d_k > 0 \). But \( f \) is strongly convex over \( S \), therefore \( f \) is bounded from below. Now, summing over \( k \) the first Wolfe condition (2.11) we have
\[ \sum_{k=0}^{\infty} \alpha_k g_k^T d_k > -\infty. \]

Considering the lower bound for \( \alpha_k \) given by (3.5) in Lemma 3.1 and having in view that \( d_k \) is a descent direction it follows that
\[ \sum_{k=1}^{\infty} \frac{g_k^T d_k^2}{\|d_k\|^2} < \infty. \]  
(3.12)

Now, from (2.13), using the inequality of Cauchy and (3.9) we get
\[ g_{k+1}^T d_{k+1} \leq -\frac{(g_{k+1} s_k)^2}{y_k^T s_k} \leq -\frac{\|g_{k+1}\| \|s_k\|^2}{\mu \|s_k\|^2} = -\frac{\|g_{k+1}\|^2}{\mu}. \]

Therefore, from (3.12) it follows that
\[ \sum_{k=0}^{\infty} \|g_k\|^2 < \infty. \]  
(3.13)

Now, inserting the upperbound (3.7), for \( d_k \) in (3.13) yields
\[ \sum_{k=0}^{\infty} \|g_k\|^2 < \infty, \]
which completes the proof.

For general functions the convergence of the algorithm is coming from theorem 2.1 and the restart procedure. Therefore, for strongly convex functions and under inexact line search it is global convergent. To a great extent, however, the algorithm is very close to the Perry/Shanno computational scheme [34, 35] which is a scaled memoryless BFGS preconditioned algorithm where the scaling factor is the inverse of a scalar approximation of the Hessian. If the Powell restart criterion (2.14) is used, for general functions \( f \) bounded from below with bounded second partial derivatives and bounded level set, using the same arguments considered by Shanno in [34] it is possible to prove that the iterates either converge to a point \( x^* \) satisfying \( \|g(x^*)\| = 0 \), or the iterates cycle. It remains for further study to determine a complete global convergence result and whether cycling can occur for general functions with bounded second partial derivatives and bounded level set.

More sophisticated reasons for restarting the algorithms have been proposed in the literature, but we are interested in the performance of an algorithm that uses the Powell restart criterion, associated with the scaled memoryless BFGS preconditioned direction choice for restart. Additionally, some convergence analysis with Powell restart criterion was given by Dai and Yuan [14] and can be used in this context of the preconditioned and scaled memoryless BFGS algorithm.

4. Acceleration of the algorithm
It is common to see that in conjugate gradient algorithms the search directions tend to be poorly scaled and as a consequence the line search must perform more function evaluations in order to obtain a suitable steplength \( \alpha_k \). In order to improve the performances of the conjugate gradient algorithms the efforts were directed to design procedures for direction computation based on the second order information. For example, CONMIN [36], and SCALCG [3,4,5,6] take this idea of BFGS preconditioning. In this section we focus on the
step length modification. In the context of gradient descent algorithm with backtracking the step length modification has been considered for the first time in [2].

Jorge Nocedal [25] pointed out that in conjugate gradient methods the step lengths may differ from 1 in a very unpredictable manner. They can be larger or smaller than 1 depending on how the problem is scaled. Numerical comparisons between conjugate gradient methods and the limited memory quasi Newton method, by Liu and Nocedal [23], show that the latter is more successful [8]. One explanation of the efficiency of the limited memory quasi-Newton method is given by its ability to accept unity step lengths along the iterations. In this section we take advantage of this behavior of conjugate gradient algorithms and present an acceleration scheme. Basically, this modifies the step length in a multiplicative manner to improve the reduction of the function values along the iterations [9].

Given the initial point \( x_0 \) we can compute \( f_0 = f(x_0) \), \( g_0 = \nabla f(x_0) \) and by Wolfe line search conditions (2.11) and (2.12) the steplength \( \alpha_0 \) is determined. With these, the next iteration is computed as: \( x_1 = x_0 + \alpha_0 d_0 \), \( (d_0 = -g_0) \) where \( f_1 \) and \( g_1 \) are immediately determined and the direction \( d_1 \) can be computed as in (2.9). Therefore, at the iteration \( k = 1, 2, ... \) we know \( x_k \), \( f_k \), \( g_k \) and \( d_k \). Suppose that \( d_k \) is a descent direction. By the Wolfe line search (2.11) and (2.12) we can compute \( \alpha_k \) with which the following point \( z = x_k + \alpha_k d_k \) is determined. The first Wolfe condition (2.11) shows that the steplength \( \alpha_k > 0 \) satisfies:

\[
 f(z) = f(x_k + \alpha_k d_k) \leq f(x_k) + \rho \alpha_k g_k^T d_k.
\]

With these, let us introduce the accelerated conjugate gradient algorithm by means of the following iterative scheme:

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

where \( \gamma_k > 0 \) is a parameter which follows to be determined in such a manner as to improve the behavior of the algorithm. Now, we have:

\[
f(x_k + \alpha_k d_k) = f(x_k) + \alpha_k g_k^T d_k + \frac{1}{2} \alpha_k^2 d_k^T \nabla^2 f(x_k) d_k + o\left(\|\alpha_k d_k\|^2\right).
\]

(4.2)

On the other hand, for \( \gamma > 0 \) we have:

\[
f(x_k + \gamma \alpha_k d_k) = f(x_k) + \gamma \alpha_k g_k^T d_k + \frac{1}{2} \gamma^2 \alpha_k^2 d_k^T \nabla^2 f(x_k) d_k + o\left(\|\gamma \alpha_k d_k\|^2\right).
\]

With these we can write:

\[
f(x_k + \gamma \alpha_k d_k) = f(x_k + \alpha_k d_k) + \Psi_k(\gamma),
\]

\[
\Psi_k(\gamma) = \frac{1}{2} (\gamma^2 - 1) \alpha_k^2 d_k^T \nabla^2 f(x_k) d_k + (\gamma - 1) \alpha_k g_k^T d_k
\]

\[
+ \gamma^2 \alpha_k o\left(\alpha_k \|d_k\|^2\right) - \alpha_k o\left(\alpha_k \|d_k\|^2\right).
\]

Let us denote:

\[
a_k = \alpha_k g_k^T d_k \leq 0,
\]

\[
b_k = \alpha_k^2 d_k^T \nabla^2 f(x_k) d_k,
\]

\[
\varepsilon_k = o\left(\alpha_k \|d_k\|^2\right).
\]

Let us denote:

Observe that \( a_k \leq 0 \), since \( d_k \) is a descent direction, and for convex functions \( b_k \geq 0 \). Therefore,

\[
\Psi_k(\gamma) = \frac{1}{2} (\gamma^2 - 1) b_k + (\gamma - 1) a_k + \gamma^2 \alpha_k \varepsilon_k - \alpha_k \varepsilon_k.
\]
Now, we see that $\Psi'_d(\gamma) = (b_k + 2\alpha_k e_k)\gamma + a_k$ and $\Psi'_m(\gamma_m) = 0$, where

$$\gamma_m = -\frac{a_k}{b_k + 2\alpha_k e_k}. \quad (4.7)$$

Observe that $\Psi'_d(0) = a_k < 0$. Therefore, assuming that $b_k + 2\alpha_k e_k > 0$, then $\Psi'_d(\gamma)$ is a convex quadratic function with minimum value in point $\gamma_m$ and

$$\Psi'_d(\gamma_m) = -\frac{(a_k + (b_k + 2\alpha_k e_k))^2}{2(b_k + 2\alpha_k e_k)} \leq 0.$$

Considering $\gamma = \gamma_m$ in (4.4) and since $b_k \geq 0$, we see that for every $k$

$$f(x_k + \gamma_m a_k d_k) = f(x_k + \alpha_k d_k) - \frac{(a_k + (b_k + 2\alpha_k e_k))^2}{2(b_k + 2\alpha_k e_k)} \leq f(x_k + \alpha_k d_k),$$

which is a possible improvement of the values of function $f$ (when $a_k + (b_k + 2\alpha_k e_k) \neq 0$).

Therefore, using this simple multiplicative modification of the stepsize $\alpha_k$ as $\gamma_m \alpha_k$ where $\gamma_m = -a_k / (b_k + 2\alpha_k e_k)$ we get:

$$f(x_{k+1}) = f(x_k + \gamma_m \alpha_k d_k) \leq f(x_k) + \rho \alpha_k g_k^T d_k - \frac{(a_k + (b_k + 2\alpha_k e_k))^2}{2(b_k + 2\alpha_k e_k)} \leq f(x_k), \quad (4.8)$$

since $a_k \leq 0$, ($d_k$ is a descent direction).

Observe that if $d_k$ is a descent direction, then

$$\frac{(a_k + (b_k + 2\alpha_k e_k))^2}{2(b_k + 2\alpha_k e_k)} > \frac{(a_k + b_k)^2}{2b_k}$$

and from (4.8) we get:

$$f(x_{k+1}) \leq f(x_k) - \left[\frac{(a_k + (b_k + 2\alpha_k e_k))^2}{2(b_k + 2\alpha_k e_k)} - \rho a_k\right] \leq f(x_k).$$

Therefore, neglecting the contribution of $e_k$, we still get an improvement on the function values.

Now, in order to get the algorithm we have to determine a way for $b_k$ computation. For this, at point $z = x_k + \alpha_k d_k$ we have:

$$f(z) = f(x_k + \alpha_k d_k) = f(x_k) + \alpha_k g_k^T d_k + \frac{\alpha_k^2}{2} d_k^T \nabla^2 f(\tilde{x}_k) d_k,$$

where $\tilde{x}_k$ is a point on the line segment connecting $x_k$ and $z$. On the other hand, at point $x_k = z - \alpha_k d_k$ we have:

$$f(x_k) = f(z - \alpha_k d_k) = f(z) - \alpha_k g_k^T d_k + \frac{\alpha_k^2}{2} d_k^T \nabla^2 f(\tilde{x}_k) d_k,$$

where $g_z = \nabla f(z)$ and $\tilde{x}_k$ is a point on the line segment connecting $x_k$ and $z$. Having in view the local character of searching and that the distance between $x_k$ and $z$ is small enough, we can consider $\tilde{x}_k = \tilde{z}_k = x_k$. So, adding the above equalities we get:
where \( y_k = g_k - g_i \).

Observe that if \( |a_k| > b_k \), then \( \gamma_k > 1 \). In this case \( \gamma_k \alpha_k > \alpha_i \) and it is also possible that \( \gamma_k \alpha_k \leq 1 \) or \( \gamma_k \alpha_k > 1 \). Hence, the steplength \( \gamma_k \alpha_k \) can be greater than 1. On the other hand, if \( |a_k| \leq b_k \), then \( \gamma_k \leq 1 \). In this case \( \gamma_k \alpha_k \leq \alpha_i \), so the steplength \( \gamma_k \alpha_k \) is reduced. Therefore, if \( |a_k| \neq b_k \), then \( \gamma_k \neq 1 \) and the steplength \( \alpha_k \) computed by Wolfe conditions will be modified by its increasing or its reducing through factor \( \gamma_k \).

Neglecting \( \varepsilon_k \) in (4.6), we see that \( \Psi_k(1) = 0 \) and if \( |a_k| \leq b_k / 2 \), then \( \Psi_k(0) = -a_k - b_k / 2 \leq 0 \) and \( \gamma_k < 1 \). Therefore, for any \( \gamma \in [0, 1] \), \( \Psi_k(\gamma) \leq 0 \). As a consequence for any \( \gamma \in (0, 1) \), it follows that \( f(x_k + \gamma \alpha_k d_k) < f(x_k) \). In this case, for any \( \gamma \in [0, 1] \), \( \gamma_k \alpha \leq \alpha_i \). However, in our algorithm we selected \( \gamma_k = \gamma_m \) as the point achieving the minimum value of \( \Psi_k(\gamma) \).

### 5. ASCALCG Algorithm

Having in view the above developments and the definitions of \( g_k \), \( s_k \) and \( y_k \), as well as the selection procedure for \( \theta_{k+1} \) computation, the following accelerated scaled conjugate gradient algorithm can be presented.

**Step 1. Initialization.** Select \( x_0 \in \mathbb{R}^n \), and the parameters \( 0 < \rho < \sigma < 1 \). Compute \( f(x_0) \) and \( g_0 = \nabla f(x_0) \). Set \( d_0 = -g_0 \) and \( \alpha_0 = 1 / \|g_0\| \). Set \( k = 0 \).

**Step 2. Line search.** Compute \( \alpha_k \) satisfying the Wolfe conditions (2.11) and (2.12). Update the variables \( x_{k+1} = x_k + \alpha_k d_k \). Compute \( f(x_{k+1}), g_{k+1} \) and \( s_k = x_{k+1} - x_k \), \( y_k = g_{k+1} - g_k \).

**Step 3. Test for continuation of iterations.** If this test is satisfied the iterations are stopped, else set \( k = k + 1 \).

**Step 4. Scaling factor computation.** Compute \( \theta_k \) using (2.18).

**Step 5. Restart direction.** Compute the (restart) direction \( d_k \) as in (2.9).

**Step 6. Line search.** Compute the initial guess: \( \alpha_k = \alpha_k-1 \|d_k-1\|_2 / \|d_k\|_2 \). Using this initialization compute \( \alpha_k \) satisfying the Wolfe conditions. Update the variables \( x_{k+1} = x_k + \alpha_k d_k \). Compute \( f(x_{k+1}) \) and \( g_{k+1} \).

**Step 7. Acceleration scheme.** Compute \( a_k = g_k^T d_k \) and \( b_k = (g_k - g_{k+1})^T d_k \). If \( b_k \neq 0 \), then compute \( \gamma_k = a_k / b_k \) and update the variables as: \( x_{k+1} = x_k + \gamma_k \alpha_k d_k \). Compute \( f(x_{k+1}) \), \( g_{k+1} \) and \( s_k = x_{k+1} - x_k \), \( y_k = g_{k+1} - g_k \). Otherwise (if \( b_k = 0 \)), then compute \( s_k = x_{k+1} - x_k \), \( y_k = g_{k+1} - g_k \).

**Step 8. Store:** \( \theta = \theta_k \), \( s = s_k \) and \( y = y_k \).

**Step 9. Test for continuation of iterations.** If this test is satisfied the iterations are stopped, else set \( k = k + 1 \).

**Step 10. Restart.** If the Powell restart criterion (2.14) is satisfied, then go to step 4 (a restart step); otherwise continue with step 11 (a standard step).

**Step 11. Standard direction.** Compute the direction \( d_k \) as in (2.17), where \( v \) and \( w \) are computed as in (2.15) and (2.16) with saved values \( \theta \), \( s \) and \( y \).
Step 12. **Line search.** Compute the initial guess: \( \alpha_k = \alpha_{k-1} \left\| d_{k-1} \right\|_2 / \left\| d_k \right\|_2 \). Using this initialization compute \( \alpha_k \) satisfying the Wolfe conditions. Update the variables \( x_{k+1} = x_k + \alpha_k d_k \). Compute \( f(x_{k+1}) \) and \( g_{k+1} \).

Step 13. **Acceleration scheme.** Compute \( a_k = g_k^T d_k \) and \( b_k = (g_k - g_{k+1})^T d_k \). If \( b_k \neq 0 \), then compute \( \gamma_k = a_k / b_k \) and update the variables as: \( x_{k+1} = x_k + \gamma_k a_k d_k \). Compute \( f(x_{k+1}) \), \( g_{k+1} \) and \( s_k = x_{k+1} - x_k \), \( y_k = g_{k+1} - g_k \). Otherwise (if \( b_k = 0 \)), then compute \( s_k = x_{k+1} - x_k \), \( y_k = g_{k+1} - g_k \).

Step 14. **Test for continuation of iterations.** If this test is satisfied the iterations are stopped, else set \( k = k + 1 \) and go to step 10.

It is well known that if \( f \) is bounded below along the direction \( d_k \), then there exists a step length \( \alpha_k \) satisfying the Wolfe conditions. The initial selection of the step length crucially affects the practical behavior of the algorithm. At every iteration \( k \geq 1 \) the starting guess for the step \( \alpha_k \) in line search is computed as \( \alpha_k = a_k / b_k \). This procedure was considered for the first time by Shanno and Phua in CONMIN [36]. The same one is taken by Birgin and Martínez in SCG [11] and in the SCALCG algorithm [3,4,5,6]. In steps 3, 9 and 14 we can consider, for example, the following test:

\[ 6(\alpha_k) \leq 10^{-6}. \]

**Proposition 5.1.** Suppose that \( f \) is an uniformly convex function on the level set \( S = \{ x : f(x) \leq f(x_0) \} \), and \( d_k \) satisfies the sufficient descent condition \( g_k^T d_k < -c_1 \left\| g_k \right\|^2 \), where \( c_1 > 0 \), and \( \left\| d_k \right\|^2 \leq c_2 \left\| g_k \right\|^2 \), where \( c_2 > 0 \). Then the sequence generated by ASCALCG converges linearly to \( x^* \), solution to the problem (1.1).

**Proof.** From (4.8) we have that \( f(x_{k+1}) \leq f(x_k) \) for all \( k \). Since \( f \) is bounded below, it follows that
\[
\lim_{k \to \infty} (f(x_k) - f(x_{k+1})) = 0.
\]
Now, since \( f \) is uniformly convex there exist positive constants \( m \) and \( M \), such that \( mI \leq \nabla^2 f(x) \leq MI \) on \( S \). Suppose that \( x_k + \alpha d_k \in S \) and \( x_k + \gamma \alpha d_k \in S \) for all \( \alpha > 0 \). We have:
\[
f(x_k + \gamma \alpha d_k) \leq f(x_k + \alpha d_k) - \frac{(a_k + b_k)^2}{2b_k}.
\]
But, from uniform convexity we have the following quadratic upper bound on \( f(x_k + \alpha d_k) \):
\[
f(x_k + \alpha d_k) \leq f(x_k) + \alpha g_k^T d_k + \frac{1}{2} M \alpha^2 \left\| d_k \right\|^2.
\]
Therefore,
\[
f(x_k + \alpha d_k) \leq f(x_k) + a_k \left\| g_k \right\|^2 + \frac{1}{2} M \left\| g_k \right\|^2 = f(x_k) + \left[ -c_1 \alpha + \frac{1}{2} M c_2 \alpha^2 \right] \left\| g_k \right\|^2.
\]
Observe that for \( 0 \leq \alpha \leq c_1 /(M c_2) \), \( -c_1 \alpha + \frac{1}{2} M c_2 \alpha^2 \leq -\frac{c_1}{2} \alpha \) which follows from the convexity of \( -c_1 \alpha + (M c_2 / 2) \alpha^2 \). Using this result we get:
\[ f(x_k + ad_k) \leq f(x_k) - \frac{1}{2} c_i \alpha \left\| g_k \right\|^2 \leq f(x_k) - \rho c_i \alpha \left\| g_k \right\|^2, \]

since \( \rho < 1/2 \).

From Lemma 3.1 the Wolfe line search terminates with a value \( \alpha \geq \omega > 0 \). Therefore, for \( 0 \leq \alpha \leq c_i/(Mc_2) \), this provides a lower bound on the decrease in the function \( f \), i.e.

\[ f(x_k + ad_k) \leq f(x_k) - \rho c_i \alpha \left\| g_k \right\|^2. \quad (5.1) \]

On the other hand,

\[ \frac{(a_k + b_k)^2}{2b_k} \geq \frac{(\alpha^2 Mc_2 - \alpha c_i)^2}{2 \alpha^2 Mc_2} \left\| g_k \right\|^2 \geq \frac{(\omega Mc_2 - c_i)^2}{2 Mc_2} \left\| g_k \right\|^2. \quad (5.2) \]

Considering (5.1) and (5.2) we get:

\[ f(x_k + \gamma_m ad_k) \leq f(x_k) - \rho c_i \omega \left\| g_k \right\|^2 - \frac{(\omega Mc_2 - c_i)^2}{2 Mc_2} \left\| g_k \right\|^2. \quad (5.3) \]

Therefore,

\[ f(x_k) - f(x_k + \gamma_m ad_k) \geq \rho c_i \omega + \frac{(\omega Mc_2 - c_i)^2}{2 Mc_2} \left\| g_k \right\|^2. \]

But, \( f(x_k) - f(x_{k+1}) \to 0 \) and as a consequence \( g_k \) goes to zero, i.e. \( x_k \) converges to \( x^* \).

Having in view that \( f(x_k) \) is a nonincreasing sequence, it follows that \( f(x_k) \) converges to \( f(x^*) \). From (5.3) we see that

\[ f(x_{k+1}) \leq f(x_k) - \rho c_i \omega + \frac{(\omega Mc_2 - c_i)^2}{2 Mc_2} \left\| g_k \right\|^2. \quad (5.4) \]

Combining this with \( \left\| g_k \right\|^2 \geq 2mf(x_k) - f^* \) and subtracting \( f^* \) from both sides of (5.4) we conclude:

\[ f(x_{k+1}) - f^* \leq c(f(x_k) - f^*), \]

where

\[ c = 1 - 2m \left[ \rho c_i \omega + \frac{(\omega Mc_2 - c_i)^2}{2 Mc_2} \right] < 1. \]

Therefore, \( f(x_k) \) converges to \( f^* \) at least as fast as a geometric series with a factor that depends on the parameter \( \rho \) in the first Wolfe condition and the bounds \( m \) and \( M \). So, the convergence of the acceleration scheme is at least linear.

\section*{6. Computational results and comparisons}

In this section we present the performance of a Fortran implementation of the \textit{ASCA CG} – \textit{accelerated scaled conjugate gradient algorithm} on a set of 750 unconstrained optimization test problems. At the same time, we compare the performance of ASCA CG with some conjugate gradient algorithms including SCALCG [3,4,5,6], Hestenes-Stiefel (HS) [22], Polak-Ribière-Polyak (PRP) [29,30], Dai-Yuan (DY) [15], Dai-Liao (DL) [13], conjugate gradient with sufficient descent CGSD [7], hybrid Dai-Yuan (hDY) [15], hybrid Dai-Yuan zero (hDYz) [15], CG DESCENT [20,21], LBFGS (m=3, m=5) [23], truncated Newton TN by Nash [24] and accelerated conjugate gradient with modified secant condition ACGMSEC [10].

All codes are written in Fortran and compiled with f77 (default compiler settings) on an Intel Pentium 4, 1.5Ghz. The CG DESCENT code by Hager and Zhang [20,21] contains the variant implementing the Wolfe line search (W) and the variant corresponding to the
approximate Wolfe conditions (aW). All algorithms implement the same stopping criteria

\[ \| g_i \|_\infty \leq \varepsilon_g, \]  

where \( \| \cdot \|_\infty \) denotes the maximum absolute component of a vector and \( \varepsilon_g = 10^{-6} \).

The test problems are the unconstrained problems in the CUTE [12] collection, along with other large-scale optimization problems [1]. We selected 75 large-scale unconstrained optimization problems in extended or generalized form. For each function we have considered 10 numerical experiments with number of variables \( n = 1000, 2000, \ldots, 10000 \).

The comparisons of algorithms are given in the following context. Let \( f_i^{\text{ALG1}} \) and \( f_i^{\text{ALG2}} \) be the optimal value found by ALG1 and ALG2, for problem \( i = 1, \ldots, 750 \), respectively. We say that, in the particular problem \( i \), the performance of ALG1 was better than the performance of ALG2 if:

\[ |f_i^{\text{ALG1}} - f_i^{\text{ALG2}}| < 10^{-3}, \]

and the number of iterations, or the number of function-gradient evaluations, or the CPU time of ALG1 was less than the number of iterations, or the number of function-gradient evaluations, or the CPU time corresponding to ALG2, respectively.

In the first set of numerical experiments we compare ASCALCG with SCALCG. Basically, SCALCG [3,4,5,6] is the unaccelerated variant of ASCALCG. Figure 1 presents the Dolan-Moré [17] CPU performance profiles of these algorithms, i.e. we plot the fraction of problems for which the given method is within a factor \( \tau \) of the best time.

We see that the best performance, relative to the CPU time metric, was obtained by ASCALCG, the top curve in Figure 1. Hence, ASCALCG appears to generate the best steplength, on average.

In the second set of numerical experiments we compare ASCALCG to HS, PRP, DY, DL (t=1), CGSD, hDy and hDYz. Figures 2-8 present the Dolan-Moré CPU performance profiles of these classical conjugate gradient algorithms.
Fig. 2. ASCALCG versus Hestenes – Stiefel (HS).

<table>
<thead>
<tr>
<th></th>
<th>ASCALCG</th>
<th>HS</th>
</tr>
</thead>
<tbody>
<tr>
<td>#iter</td>
<td>579</td>
<td>38</td>
</tr>
<tr>
<td>#fg</td>
<td>522</td>
<td>84</td>
</tr>
<tr>
<td>cpu</td>
<td>206</td>
<td>127</td>
</tr>
</tbody>
</table>

CPU time metric, 658 problems

Fig. 3. ASCALCG versus Polak-Ribiére-Polyak (PRP).

<table>
<thead>
<tr>
<th></th>
<th>ASCALCG</th>
<th>PRP</th>
</tr>
</thead>
<tbody>
<tr>
<td>#iter</td>
<td>579</td>
<td>34</td>
</tr>
<tr>
<td>#fg</td>
<td>527</td>
<td>76</td>
</tr>
<tr>
<td>cpu</td>
<td>216</td>
<td>111</td>
</tr>
</tbody>
</table>

CPU time metric, 653 problems
Fig. 4. ASCALCG versus Dai-Yuan (DY).

<table>
<thead>
<tr>
<th></th>
<th>ASCALCG</th>
<th>DY</th>
<th>Dy</th>
</tr>
</thead>
<tbody>
<tr>
<td>#iter</td>
<td>585</td>
<td>38</td>
<td>36</td>
</tr>
<tr>
<td>#f</td>
<td>539</td>
<td>71</td>
<td>49</td>
</tr>
<tr>
<td>cpu</td>
<td>205</td>
<td>130</td>
<td>324</td>
</tr>
</tbody>
</table>

CPU time metric, 659 problems

---

Fig. 5. ASCALCG versus Dai-Liao (t=1) (DL).

<table>
<thead>
<tr>
<th></th>
<th>ASCALCG</th>
<th>DL</th>
<th>Dy</th>
</tr>
</thead>
<tbody>
<tr>
<td>#iter</td>
<td>585</td>
<td>33</td>
<td>41</td>
</tr>
<tr>
<td>#f</td>
<td>533</td>
<td>73</td>
<td>53</td>
</tr>
<tr>
<td>cpu</td>
<td>213</td>
<td>125</td>
<td>321</td>
</tr>
</tbody>
</table>

CPU time metric, 659 problems
Fig. 6. ASCALCG versus CG with Sufficient Descent Condition (CGSD).

Fig. 7. ASCALCG versus Hybrid Dai-Yuan (hDY).
In the third set of numerical experiments we compare ASCALCG to CG_DESCENT. In Figures 9 and 10 the Dolan-Moré CPU performance profiles are presented.

**Fig. 8.** ASCALCG versus Hybrid Dai-Yuan zero (hDYz).

**Fig. 9.** ASCALCG versus CG_DESCENT (Wolfe conditions).
Fig. 10. ASCALCG versus CG_DESCENT (approximate Wolfe conditions).

In the fourth set of numerical experiments in Figures 11 and 12 the Dolan-Moré performance profiles of ASCALCG versus LBFGS (m=3) and LBFGS (m=5) are presented.

Fig. 11. ASCALCG versus LBFGS (m=3).
Finally, Figures 13 and 14 illustrate the performance profiles of ASCALCG versus the truncated Newton TN algorithm by Nash and accelerated conjugate gradient with modified secant condition ACGMSEC [10], respectively.

Fig. 12. ASCALCG versus LBFGS (m=5).

Fig. 13. ASCALCG versus truncated Newton TN (Nash).
The percentage of the test problems for which a method is the fastest is given on the left axis of the plot. The right side of the plot gives the percentage of the test problems that were successfully solved by these algorithms, respectively.

When comparing ASCALCG with SCALCG (Figure 1), subject to the number of iterations, we see that ASCALCG was better in 476 problems (i.e. it achieved the minimum number of iterations in 476 problems). SCALCG was better in 58 problems and they achieved the same number of iterations in 125 problems, etc. Out of 750 problems, only for 650 problems does the criterion (6.1) hold. Clearly, introducing the acceleration scheme represents an important ingredient in getting an efficient conjugate gradient algorithm. Numerical experiments proved that for the majority of iterations $\gamma_k = a_k / b_k < 1$, i.e. the acceleration scheme has the propensity to reduce the values of the step lengths. From Figures 2-8 we see that the accelerated conjugate gradient algorithm ASCALCG is more successful and more robust than the considered classical and hybrid conjugate gradient algorithms considered in this numerical study. Hence, ASCALCG appears to generate the best search direction and the best steplength, on average. Not only ASCALCG is the fastest among these algorithms, but it is also more robust in solving a large variety of unconstrained optimization problems.

In Figures 9 and 10 we present the performance profiles of ASCALCG and CG_DESCENT by Hager and Zhang [20, 21]. The top solid curve in these Figures corresponds to ASCALCG. Observe that subject to the CPU time metric, ASCALCG is more robust. Also, it is interesting to observe in Figure 9 that for $\tau = 1$, relative to the CPU time metric, CG_DESCENT is slighter better. However, for $\tau > 1$, ASCALCG turn out to be faster than CG_DESCENT. Even that CG_DESCENT with approximate Wolfe conditions computes a more accurate steplength, the acceleration scheme introduced in scaled BFGS preconditioned conjugate gradient algorithm, as implemented in ASCALCG, is way more robust (Figure 10).

From Figures 11 and 12 again we see that the best performance is obtained by ASCALCG. LBFGS is a professional implementation of the limited memory quasi-Newton method [23], where $m$ is the number of the stored pairs $(s_k, y_k)$. Even that LBFGS admits unit step
lengths for the most of the iterations, thus requiring only few function and gradient evaluations for steplength determination, ASCALCG combines in a more adequate way the direction determination and step length to be the top performer.

It is interesting to observe in Figure 13 that TN by Nash [24] is completely outperformed by ASCALCG. Finally, in Figure 14 again we see that the best performance is obtained by ASCALCG. In ACGMSEC [10], which is the accelerated conjugate gradient algorithm with modified secant condition, \( s_k^T \nabla^2 f(x_{k+1}) s_k \) in (2.4) is approximated by the modified secant condition \( B_{k+1} s_k = y_k, \quad y_k = v_k + \eta_k u_k / (s_k^T u_k) \), where \( \eta_k = 6(f_k - f_{k+1}) + 3(g_k + g_{k+1})^T s_k \) and \( u_k \in \mathbb{R}^n \) is any vector such that \( s_k^T u_k \neq 0 \).

7. Conclusion

We have presented a new conjugate gradient algorithm which mainly is an acceleration of SCALCG – scaled BFGS preconditioned conjugate gradient algorithm [3,4,5,6]. The acceleration scheme is simple and proved to be robust in numerical experiments. In very mild conditions we proved that the algorithm is globally convergent. For uniformly convex functions the convergence of the accelerated algorithm is still linear, but the reduction in the function values is significantly improved. For a set of 750 test unconstrained optimization problems (some from CUTE library) with dimensions ranging between 1000 and 10000 variables, the CPU time performance profile for ASCALCG was higher than those of SCALCG, HS, PRP, DY, DL (t=1), CGSD, hDY, hDYz, CG_DESCENT, LBFGS, TN and ACGMSEC.

References


March 24, 2008