Scaled memoryless BFGS preconditioned conjugate gradient algorithm for unconstrained optimization

Neculai Andrei[1]
Research Institute for Informatics,
Center for Advanced Modeling and Optimization,
8-10, Averescu Avenue, Bucharest 1, Romania
E-mail: nandrei@ici.ro

Abstract. This work presents and analyzes a scaled memoryless BFGS preconditioned conjugate gradient algorithm, and its implementation, for solving unconstrained optimization problems. 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 a restart criterion holds. The computational scheme is embedded into the restart philosophy of Beale-Powell. The parameter scaling the gradient is selected as spectral gradient or in an anticipative manner by means of a formula using the function values in two successive points. In very mild conditions it is shown that, for strongly convex functions, the algorithm is globally convergent. Computational results and performance profiles, for a test set consisting of over 500 unconstrained optimization problems, show that this new scaled conjugate gradient algorithm substantially outperforms known conjugate gradient methods including: the spectral conjugate gradient by Birgin and Martínez [4], the conjugate gradient by Fletcher and Reeves [10], Polak and Ribière [17], and by Hestenes and Stiefel [13], as well as the most recent CG_DESCENT conjugate gradient method with guaranteed descent by Hager and Zhang [11,12].

Keywords: Unconstrained optimization, conjugate gradient method, spectral gradient method
AMS Subject Classification: 49M07, 49M10, 90C06, 65K

1. INTRODUCTION

In this paper we consider the following unconstrained optimization problem:

$$\min f(x) \tag{1}$$

where $f : R^n \to R$ 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 and analyze a conjugate gradient algorithm based on a combination of the scaled memoryless BFGS method and the preconditioning technique. For general nonlinear functions a good preconditioner is any matrix that approximates $\nabla^2 f(x^*)^{-1}$. In this algorithm the preconditioner is a scaled memoryless BFGS matrix which is reset when a restart criterion holds. The scaling factor in the preconditioner is selected as spectral gradient or in an anticipative manner by means of a formula using the function values in two successive points.

[1] The author was awarded the Romanian Academy Grant 168/2003.

The algorithm uses the conjugate gradient direction where the famous parameter β_k is obtained by equalizing the conjugate gradient direction with the direction corresponding to the Newton method. Thus, we get a general formula for the direction computation, which could be particularized to include the Polak and Ribiére [17] and the Fletcher and Reeves [10] conjugate gradient algorithms, the spectral conjugate gradient (SCG) by Birgin and Martínez [4] or the algorithm of Dai and Liao [6] for $(t = 1)$. This direction is then modified into a canonical manner as it was considered earlier by Oren and Luenberger [14], Oren and Spedicato [15], Perry [16] and Shanno [21,22], by means of a scaled, memoryless BFGS preconditioner placed into the Beale-Powell restart thechnology. The scaling factor is computed in a spectral manner based on the inverse Rayleigh quotient, as suggested by Raydan [20], or into an anticipative one using only the function values Andrei [1].

The method is an extension of the spectral conjugate gradient (SCG) by Birgin and Martínez [4] or of a variant of the conjugate gradient algorithm by Dai and Liao [6] (for $t = 1$) to overcome the lack of positive definiteness of the matrix defining their search direction.

The paper is organized as follows: In section 2 we present the method. Section 3 is dedicated to the SCALCG algorithm. The algorithm performs two types of steps: a normal 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 4. Finally, in section 5 we present detailed computational results on a set of 500 unconstrained optimization problems and compare the Dolan and Moré [9] performance profiles of the new algorithm to profiles for SCG conjugate gradient method by Birgin and Martínez [4], CG_DESCENT by Hager and Zhang [11,12], as well as the scaled Polak and Ribiére conjugate gradient algorithm. At the same time we present the performances of these algorithms on some applications from MINPACK-2 collection [2].

2. THE METHOD

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

$$x_{k+1} = x_k + \alpha_k d_k, \tag{2}$$
$$d_{k+1} = -\theta_{k+1} g_{k+1} + \beta_k s_k, \tag{3}$$

where $g_k = \nabla f(x_k)$, α_k is selected to minimize $f(x)$ along the search direction d_k, β_k is a scalar parameter and θ_{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 β_k. On the other hand, if $\beta_k = 0$, then we get another class of algorithms according to the selection of the parameter θ_{k+1}. There are two possibilities for θ_{k+1}: a positive scalar or a positive definite matrix. If $\theta_{k+1} = 1$ we have the steepest descent algorithm. If $\theta_{k+1} = \nabla^2 f(x_{k+1})^{-1}$, 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 $\beta_k \neq 0$, (3) represents a combination between the quasi-Newton and the conjugate gradient methods.

To determine β_k consider the following procedure. As we know, the Newton direction for solving (1) is given by $d_{k+1} = -\nabla^2 f(x_{k+1})^{-1} g_{k+1}$. Therefore, from the equality

$$-\nabla^2 f(x_{k+1})^{-1} g_{k+1} = -\theta_{k+1} g_{k+1} + \beta_k s_k,$$

we get:

$$\beta_k = \frac{\theta_{k+1} s_k^T \nabla^2 f(x_{k+1}) g_{k+1} - s_k^T g_{k+1}}{s_k^T \nabla^2 f(x_{k+1}) s_k}. \tag{4}$$

Using the Taylor development, after some algebra we obtain:

$$\beta_k = \frac{(\theta_{k+1} y_k - s_k)^T g_{k+1}}{y_k^T s_k}, \tag{5}$$

where $s_k = x_{k+1} - x_k$ and $y_k = g_{k+1} - g_k$. Birgin and Martínez [4] arived at the same formula for β_k, but using a geometric interpretatioin for quadratic function minimization. The direction corresponding to β_k given in (5) is as follows:

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \frac{(\theta_{k+1} y_k - s_k)^T g_{k+1}}{y_k^T s_k} s_k. \tag{6}$$

This direction is used by Birgin and Martínez [4] in their SCG (spectral conjugate gradient) package for unconstrained optimization, where θ_{k+1} is selected in a spectral manner, as suggested by Raydan [20]. The following particularizations are obvious. If $\theta_{k+1} = 1,$ then (6) is the direction considered by Perry [16]. At the same time we see that (6) is the direction given by Dai and Liao [6] for $t = 1$, obtained this time by an interpretation of the conjugacy condition. Additionally, if $s_j^T g_{j+1} = 0,\ j = 0,1,\dots,k,$ then from (6) we get:

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \frac{\theta_{k+1} y_k^T g_{k+1}}{\alpha_k \theta_k g_k^T g_k} s_k, \tag{7}$$

which is the direction corresponding to a *generalization of the Polak and Ribière* formula. Of course, if $\theta_{k+1} = \theta_k = 1$ in (7), we get the *classical Polak and Ribière* formula [17]. If $s_j^T g_{j+1} = 0,\ j = 0,1,\dots,k,$ and additionally the successive gradients are orthogonal, then from (6)

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \frac{\theta_{k+1} g_{k+1}^T g_{k+1}}{\alpha_k \theta_k g_k^T g_k} s_k, \tag{8}$$

which is the direction corresponding to a *generalization of the Fletcher and Reeves* formula [10]. Therefore, (6) is a general formula for direction computation in a conjugate gradient manner including the classical Fletcher and Reeves [10], and Polak and Ribière [17] formulas.

There is a result of Shanno [21,22] that says that the conjugate gradient method is the BFGS quasi-Newton method for which the approximation to the inverse of the Hessian is taken as the identity matrix at every iteration. The extension to the scaled conjugate gradient is very simple. Using the same methodology as considered by Shanno [21] we get the following direction d_{k+1}:

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \theta_{k+1}\left(\frac{g_{k+1}^T s_k}{y_k^T s_k}\right) y_k - \left[\left(1 + \theta_{k+1}\frac{y_k^T y_k}{y_k^T s_k}\right)\frac{g_{k+1}^T s_k}{y_k^T s_k} - \theta_{k+1}\frac{g_{k+1}^T y_k}{y_k^T s_k}\right] s_k, \tag{9}$$

involving only 4 scalar products. Again observe that if $g_{k+1}^T s_k = 0,$ then (9) reduces to:

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \theta_{k+1}\frac{g_{k+1}^T y_k}{y_k^T s_k} s_k. \tag{10}$$

Thus, in this case, the effect is simply one of multiplying the Hestenes and Stiefel [13] search direction by a positive scalar.

In order to ensure the convergence of the algorithm (2), with d_{k+1} given by (9), we need to constrain the choice of α_k. We consider line searches that satisfy the Wolfe conditions [24,25]:

$$f(x_k+\alpha_k d_k)-f(x_k)\le \sigma_1\alpha_k g_k^T d_k, \tag{11}$$

$$\nabla f(x_k+\alpha_k d_k)^T d_k \ge \sigma_2 g_k^T d_k, \tag{12}$$

where $0<\sigma_1\le\sigma_2<1$.

Theorem 1. *Suppose that α_k in (2) satisfies the Wolfe conditions (11) and (12), then the direction d_{k+1} given by (9) is a descent direction.*

Proof: Since $d_0=-g_0$, we have $g_0^T d_0=-\|g_0\|^2\le 0$. Multiplying (9) by g_{k+1}^T, we have

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

Applying the inequality $u^T v\le\frac{1}{2}(\|u\|^2+\|v\|^2)$ 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:

$$g_{k+1}^T d_{k+1}\le-\frac{(g_{k+1}^T s_k)^2}{y_k^T s_k}. \tag{13}$$

But, by Wolfe condition (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 (12) is crucial for the descent character of direction (9). Besides, we see that the estimation (13) is independent of the parameter θ_{k+1}.

Usually, all conjugate gradient algorithms are periodically restarted. The standard restarting point occurs when the number of iterations is equal to the number of variables, but some other restarting methods can be considered. The Powell restarting procedure [18,19] is to test if there is very little orthogonality left between the curent gradient and the previous one. At step r when:

$$\left|g_{r+1}^T g_r\right|\ge 0.2\|g_{r+1}\|^2, \tag{14}$$

we restart the algorithm using the direction given by (9). Another restarting procedure, considered by Birgin and Martínez [4], consists of testing if the angle between the current direction and $-g_{k+1}$ is not acute enough. Therefore, at step r when:

$$d_r^T g_{r+1}>-10^{-3}\|d_r\|_2\|g_{r+1}\|_2, \tag{15}$$

the algorithm is restarted using the direction given by (9).

At step r when one of the two criteria (14) or (15) is satisfied the direction is computed as in (9). For $k\ge r+1$, we consider the same philosophy used by Shanno [21,22], i.e. that of modifying the gradient g_{k+1} with a positive definite matrix which best estimates the inverse Hessian without any additional storage requirements. Therefore, the direction d_{k+1}, for $k\ge r+1$, is computed using a double update scheme as:

$$v\equiv H_{r+1}g_{k+1}=\theta_{r+1}g_{k+1}-\theta_{r+1}\left(\frac{g_{k+1}^T s_r}{y_r^T s_r}\right)y_r$$
$$+\left[\left(1+\theta_{r+1}\frac{y_r^T y_r}{y_r^T s_r}\right)\frac{g_{k+1}^T s_r}{y_r^T s_r}-\theta_{r+1}\frac{g_{k+1}^T y_r}{y_r^T s_r}\right]s_r, \tag{16}$$

and

$$w \equiv H_{r+1} y_k = \theta_{r+1} y_k - \theta_{r+1}\left(\frac{y_k^T s_r}{y_r^T s_r}\right) y_r$$

$$+\left[\left(1+\theta_{r+1}\frac{y_r^T y_r}{y_r^T s_r}\right)\frac{y_k^T s_r}{y_r^T s_r} - \theta_{r+1}\frac{y_k^T y_r}{y_r^T s_r}\right] s_r, \tag{17}$$

involving 6 scalar products. With these the direction d_{k+1}, at any nonrestart step, can be computed as:

$$d_{k+1} = -v + \frac{(g_{k+1}^T s_k) w + (g_{k+1}^T w) s_k}{y_k^T s_k} - \left(1 + \frac{y_k^T w}{y_k^T s_k}\right)\frac{g_{k+1}^T s_k}{y_k^T s_k} s_k, \tag{18}$$

involving only 4 scalar products. It is useful to note that $y_k^T s_k > 0$ is sufficient to ensure that the direction d_{k+1} given by (18) is well defined and it is always a descent direction.

We shall now consider some formulas for computation of θ_{k+1}. As we have already seen, in our algorithm θ_{k+1} is defined as a scalar approximation to the inverse Hessian. According to the procedures for a scalar estimation to the inverse Hessian we get a family of scaled conjugate gradient algorithms. The following procedures can be considered.

θ_{k+1} **spectral.** This is given as the inverse of the Rayleigh quotient:

$$\theta_{k+1} = \frac{s_k^T s_k}{y_k^T s_k}. \tag{19}$$

Observe that in point $x_{k+1} = x_k + s_k$ we can write:

$$f(x_{k+1}) = f(x_k) + g_k^T s_k + \frac{1}{2} s_k^T \nabla^2 f(z) s_k,$$

where z is on the line segment connecting x_k and x_{k+1}. Now, considering the local character of searching we can take $z = x_{k+1}$. On the other hand, in point $x_k = x_{k+1} - s_k$ we have:

$$f(x_k) = f(x_{k+1}) - g_{k+1}^T s_k + \frac{1}{2} s_k^T \nabla^2 f(x_{k+1}) s_k.$$

With these, considering γ_{k+1} as a scalar approximation of $\nabla^2 f(x_{k+1})$, and addind these two equalities we get exactly the Rayleigh quotient. The inverse of Rayleigh quotient is the spectral value of the scaling parameter. Again we notice that $y_k^T s_k > 0$ is sufficient to ensure that θ_{k+1} in (19) is well defined.

θ_{k+1} **anticipative.** Recently, Andrei [1], using the information in two successive points of the iterative process, generated another scalar approximation to Hessian of function f obtaining an algorithm which compares favourable with Barzilai and Borwein's [3]. This is only half a step from the spectral procedure presented above. Indeed, in point $x_{k+1} = x_k + \alpha_k d_k$ we can write

$$f(x_{k+1}) = f(x_k) + \alpha_k g_k^T d_k + \frac{1}{2}\alpha_k^2 d_k^T \nabla^2 f(z) d_k, \tag{20}$$

where z is on the line segment connecting x_k and x_{k+1}. As above, having in view the local character of the searching procedure and that the distance between x_k and x_{k+1} is small enough we can choose $z = x_{k+1}$ and consider γ_{k+1} as a scalar approximation of the $\nabla^2 f(x_{k+1})$, where $\gamma_{k+1} \in R$. This is an anticipative view point, in which a scalar approximation of the Hessian at point x_{k+1} is computed using only the local information from two successive points: x_k and x_{k+1}. Therefore, we can write:

$$\gamma_{k+1} = \frac{2}{d_k^T d_k}\frac{1}{\alpha_k^2}\left[f(x_{k+1}) - f(x_k) - \alpha_k g_k^T d_k\right]. \tag{21}$$

Observe that for convex functions $\gamma_{k+1} > 0$. If $f(x_{k+1}) - f(x_k) - \alpha_k g_k^T d_k < 0$, then the reduction $f(x_{k+1}) - f(x_k)$ in the function value is smaller than $\alpha_k g_k^T d_k$. In these cases the idea is to change a little the stepsize α_k as $\alpha_k - \eta_k$, maintaining the other quantities at their values, in such a manner so that γ_{k+1} to be positive. To get a value for η_k let us select a real $\delta > 0$, "small enough", but comparable with the value of the function, and consider

$$\eta_k = \frac{1}{g_k^T d_k}\left[f(x_k) - f(x_{k+1}) + \alpha_k g_k^T d_k + \delta\right], \tag{22}$$

with which a new value for γ_{k+1} can be computed as:

$$\gamma_{k+1} = \frac{2}{d_k^T d_k}\frac{1}{(\alpha_k - \eta_k)^2}\left[f(x_{k+1}) - f(x_k) - (\alpha_k - \eta_k) g_k^T d_k\right]. \tag{23}$$

With these, the value for parameter θ_{k+1} is selected as:

$$\theta_{k+1} = \frac{1}{\gamma_{k+1}}, \tag{24}$$

where γ_{k+1} is given by (21) or (23).

Proposition 1. *Assume that $f(x)$ is continuously differentiable and $\nabla f(x)$ is Lipschitz continuous, with a positive constant L. Then at point x_{k+1},*

$$\gamma_{k+1} \le 2L. \tag{25}$$

Proof: From (21) we have:

$$\gamma_{k+1} = \frac{2\left[f(x_k) + \alpha_k \nabla f(\xi_k)^T d_k - f(x_k) - \alpha_k \nabla f(x_k)^T d_k\right]}{\|d_k\|^2 \alpha_k^2},$$

where ξ_k is on the line segment connecting x_k and x_{k+1}. Therefore

$$\gamma_{k+1} = \frac{2\left[\nabla f(\xi_k) - \nabla f(x_k)\right]^T d_k}{\|d_k\|^2 \alpha_k}.$$

Using the inequality of Cauchy and the Lipschitz continuity it follows that

$$\gamma_{k+1} \le \frac{2\|\nabla f(\xi_k) - \nabla f(x_k)\|}{\|d_k\|\alpha_k} \le \frac{2L\|\xi_k - x_k\|}{\|d_k\|\alpha_k} \le \frac{2L\|x_{k+1} - x_k\|}{\|d_k\|\alpha_k} = 2L. \ \blacksquare$$

Therefore, from (24) we get a lower bound for θ_{k+1} as:

$$\theta_{k+1} \ge \frac{1}{2L},$$

i.e. it is bounded away from zero. It is worth saying that in the two-point stepsize gradient method, Dai, Yuan and Yuan [8] interpret the choice for the stepsize from the angle interpolation and arrive at the same formula as given in (21).

3. SCALCG ALGORITHM

Having in view the above developments and the definitions of g_k, s_k and y_k, as well as the selection procedures for θ_{k+1} computation, the following family of scaled conjugate gradient algorithms can be presented.

Step 1. Initialization. Select $x_0 \in R^n$, and the parameters $0 < \sigma_1 \le \sigma_2 < 1$. 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 α_k satisfying the Wolfe conditions (11) and (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 θ_k using a spectral (19) or an anticipative (24) approach.
Step 5. Restart direction. Compute the (restart) direction d_k as in (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 α_k satisfying the Wolfe conditions. 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 7. Store: $\theta = \theta_k$, $s = s_k$ and $y = y_k$.
Step 8. Test for continuation of iterations. If this test is satisfied the iterations are stopped, else set $k = k+1$.
Step 9. Restart. If the Powell restart criterion (14) or the angle restart criterion (15) is satisfied, then go to step 4 (a restart step); otherwise continue with step 10 (a normal step).
Step 10. Normal direction. Compute the direction d_k as in (18), where v and w are computed as in (16) and (17) with saved values θ, s and y.
Step 11. Line search. Compute the initial guess: $\alpha_k = \alpha_{k-1} \|d_{k-1}\|_2 / \|d_k\|_2$. Using this initialization compute α_k satisfying the Wolfe conditions. 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 12. Test for continuation of iterations. If this test is satisfied the iterations are stopped, else set $k = k+1$ and go to step 9. ■

It is well known that if f is bounded below along the direction d_k, then there exists a step length α_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 α_k in line search is computed as $\alpha_{k-1} \|d_{k-1}\|_2 / \|d_k\|_2$. This selection was considered for the first time by Shanno and Phua in CONMIN [23]. The same selection is taken by Birgin and Martínez in SCG [4].

Concerning the stopping criterion to be used in steps 3, 8 and 12 we have:

$$\|g_k\|_\infty \leq \varepsilon_g \quad \text{or} \quad \alpha_k |g_k^T d_k| \leq \varepsilon_f |f(x_{k+1})|, \tag{26}$$

where ε_f and ε_g are tolerances specified by the user. ε_f specifies the desired accuracy of the computed solution, ε_g specifies the desired accuracy for the gradient norm. The second criterion in (26) says that the estimated change in the function value is insignificant compared to the function value itself.

4. Convergence analysis for strongly convex functions

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

$$L_0 = \{x \in R^n : f(x) \leq f(x_0)\}. \tag{27}$$

That is, there exists constants $\mu > 0$ and L such that

$$(\nabla f(x) - \nabla f(y))^T (x-y) \geq \mu \|x-y\|^2 \tag{28}$$

and

$$\|\nabla f(x) - \nabla f(y)\| \leq L \|x-y\|, \tag{29}$$

for all x and y from L_0. For the convenience of the reader we include here the following lemma (see [11]).

Lemma 1. *Assume that d_k is a descent direction and ∇f satisfies the Lipschitz condition*

$$\|\nabla f(x) - \nabla f(x_k)\| \le L\|x - x_k\|, \tag{30}$$

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 (12), then

$$\alpha_k \ge \frac{1-\sigma_2}{L}\frac{|g_k^T d_k|}{\|d_k\|^2}. \tag{31}$$

Proof: Subtracting $g_k^T d_k$ from both sides of (12) and using the Lipschitz condition we have

$$(\sigma_2 - 1)g_k^T d_k \le (g_{k+1} - g_k)^T d_k \le L\alpha_k\|d_k\|^2. \tag{32}$$

Since d_k is a descent direction and $\sigma_2 < 1$, (31) follows immediately from (32). ■

Lemma 2. *Assume that ∇f is strongly convex and Lipschitz continuous on L_0. If θ_{k+1} is selected by spectral gradient, then the direction d_{k+1} given by (9) satisfies:*

$$\|d_{k+1}\| \le \left(\frac{2}{\mu} + \frac{2L}{\mu^2} + \frac{L^2}{\mu^3}\right)\|g_{k+1}\|. \tag{33}$$

Proof: By Lipschitz continuity (29) we have

$$\|y_k\| = \|g_{k+1} - g_k\| = \|\nabla f(x_k + \alpha_k d_k) - \nabla f(x_k)\| \le L\alpha_k\|d_k\| = L\|s_k\|. \tag{34}$$

On the other hand, by strong convexity (28)

$$y_k^T s_k \ge \mu\|s_k\|^2. \tag{35}$$

Selecting θ_{k+1} as in (19), it follows that

$$\theta_{k+1} = \frac{s_k^T s_k}{y_k^T s_k} \le \frac{\|s_k\|^2}{\mu\|s_k\|^2} = \frac{1}{\mu}. \tag{36}$$

Now, using the triangle inequality and the above estimates (34)-(36), after some algebra on $\|d_{k+1}\|$, where d_{k+1} is given by (9), we get (33). ■

Lemma 3. *Assume that ∇f is strongly convex and Lipschitz continuous on L_0. If θ_{k+1} is selected by the anticipative procedure, then the direction d_{k+1} given by (9) satisfies:*

$$\|d_{k+1}\| \le \left(\frac{1}{m} + \frac{2L}{m\mu} + \frac{1}{\mu} + \frac{L^2}{m\mu^2}\right)\|g_{k+1}\|. \tag{37}$$

Proof: By strong convexity on L_0, there exists the constant $m > 0$, so that $\nabla^2 f(x) \ge mI$, for all $x \in L_0$. Therefore, for every k, $\gamma_{k+1} \ge m$. Now, from (24) we see that, for all k,

$$\theta_{k+1} \le \frac{1}{m}. \tag{38}$$

With this, like in lemma 2, we get (37). ■

The convergence of the scaled conjugate gradient algorithm (SCALCG) when f is strongly convex is given by

Theorem 2. *Assume that f is strongly convex and Lipschitz continuous on the level set L_0. If at every step of the conjugate gradient (2) with d_{k+1} given by (9) and the step length α_k selected to satisfy the Wolfe conditions (11) and (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

$$y_k^T d_k = (g_{k+1} - g_k)^T d_k \geq \mu\alpha_k \|d_k\|^2. \tag{39}$$

By theorem 1, $g_k^T d_k < 0$. Therefore, the assumption $g_k \neq 0$ implies $d_k \neq 0$. Since $\alpha_k > 0$, from (39) it follows that $y_k^T d_k > 0$. But f is strongly convex over L_0, therefore f is bounded from below. Now, summing over k the first Wolfe condition (11) we have

$$\sum_{k=0}^{\infty} \alpha_k g_k^T d_k > -\infty.$$

Considering the lower bound for α_k given by (31) in lemma 1 and having in view that d_k is a descent direction it follows that

$$\sum_{k=1}^{\infty} \frac{\left|g_k^T d_k\right|^2}{\|d_k\|^2} < \infty. \tag{40}$$

Now, from (13), using the inequality of Cauchy and (35) we get

$$g_{k+1}^T d_{k+1} \leq -\frac{(g_{k+1}^T s_k)^2}{y_k^T s_k} \leq -\frac{\|g_{k+1}\|^2 \|s_k\|^2}{\mu \|s_k\|^2} = -\frac{\|g_{k+1}\|^2}{\mu}.$$

Therefore, from (40) it follows that

$$\sum_{k=0}^{\infty} \frac{\|g_k\|^4}{\|d_k\|^2} < \infty. \tag{41}$$

Now, inserting the upperbound (33), or (37), for d_k in (41) 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 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 SCALCG algorithm is very close to the Perry/Shanno computational scheme [21,22]. SCALCG 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 (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 [22] 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, asociated with the scaled memoryless BFGS preconditioned direction choice for restart. Additionally, some convergence analysis with Powell restart criterion was given by Dai and Yuan [7] and can be used in this context of the preconditioned and scaled memoryless BFGS algorithm.

5. Computational results and comparisons

In this section we present the performance of a Fortran implementation of the *SCALCG - scaled conjugate gradient algorithm* on a number of 500 test unconstrained optimization problems. At the same time, we compare the performance of SCALCG with *the best spectral conjugate gradient algorithm, SCG* (betatype=1,Perry-M1), by Birgin and Martínez [4], with the *CG_DESCENT - a conjugate gradient method with guaranteed descent* by Hager and Zhang [11,12] and with the *scaled Polak-Ribière algorithm (7)*.

The SCALCG code is authored by Andrei, while the SCG and Polak-Ribière are co-authored by Birgin and Martínez and CG_DESCENT is co-authored by Hager and Zhang. All codes are written in Fortran and compiled with f77 (default compiler settings) on an Intel Pentium 4, 1.5Ghz. The SCALCG code implements both the scaled conjugate gradient with the spectral choice of scaling parameter θ_{k+1}, as well as with the anticipative choice of this parameter. In order to compare SCALCG with SCG and Polak-Ribière we manufactured a new SCG code of Birgin and Martínez by introducing a sequence of code to compute the θ_{k+1} anticipative according to (24). The CG_DESCENT code contains the variant implementing the Wolfe line search (W) and the variant corresponding to the approximate Wolfe conditions (aW) [11,12].

The test problems are the unconstrained problems in the CUTE [5] library, along with other large-scale optimization test problems. We selected 50 large-scale unconstrained optimization test problems (11 from CUTE library) in extended or generalized form. For each test function we have considered 10 numerical experiments with number of variables $n = 1000, 2000, \ldots, 10000$.

The numerical results concerning the number of iterations, the number of restart iterations, the number of function and gradient evaluations, cpu time in seconds, for each of the methods are posted at the following web site:

http://www.ici.ro/camo/neculai/ansoft.htm/ (please see scalcg).

In the following we present the numerical performances of all these codes, including the performance profiles of Dolan and Moré [9] subject to the number of iterations, the number of function evaluations and cpu time metrics, respectively. For these algorithms we present their performances on 10 applications from MINPACK-2 library [2].

5.1. SCALCG with θ^s (theta spectral) versus SCALCG with θ^a (theta anticipative).

In the SCALCG code the computations are terminated as soon as the criteria (26) are satisfied, where $\varepsilon_g = 10^{-6}$ and $\varepsilon_f = 10^{-20}$. The Wolfe line search conditions are implemented with the following values of parameters σ_1 and σ_2 : $\sigma_1 = 0.0001$ and $\sigma_2 = 0.9$.

Concerning the restart criterion, we implemented both the Powell and the angle criteria. Both these criteria have a crucial role on the practical efficiency of the conjugate gradient algorithms. However, we observed that for SCALCG the Powell criterion is more performant than the angle criterion. As the numerical experiments prove, a large percentage of the SCALCG's iterations are restart iterations. Therefore we compare SCALCG algorithms with the Powell restart criterion where θ_{k+1} is selected in a spectral or in an anticipative manner.

Table 1 shows the global characteristics corresponding to these 500 test problems, refering to the total number of iterations, the total number of function evaluations and the total cpu time for these algorithms.

Table 1. Global characteristics of SCALCG with θ^s versus SCALCG with θ^a. Powell restart. 500 problems.

Global characteristics	θ^s	θ^a
Total number of iterations	123263	121352
Total number of function evaluations	229887	182643
Total cpu time (seconds)	4276.73	3669.51

Out of 500 problems solved in this set of experiments the criterion $\|g_k\|_\infty < \varepsilon_g$ stopped the iterations for 428 problems, i.e. 85.6%, in case of SCALCG with θ^s, and for 412 problems, i.e. 82.4%, in case of SCALCG with θ^a. Table 2 shows the number of problems, out of 500, for which SCALCG with θ^s and SCALCG with θ^a achieved the minimum number of iterations, the minimum number of function evaluations and the minimum cpu time, respectively.

Table 2. Performance of SCALCG algorithms. Powell restart. 500 problems.

Performance criterion	**# of problems**
SCALCG with θ^s achieved minimum # of iterations in	332
SCALCG with θ^a achieved minimum # of iterations in	307
SCALCG with θ^s and SCALCG with θ^a achieved the same # of iterations in	139
SCALCG with θ^s achieved minimum # of function evaluations in	307
SCALCG with θ^a achieved minimum # of function evaluations in	322
SCALCG with θ^s and SCALCG with θ^a achieved the same # of function evaluations in	129
SCALCG with θ^s achieved minimum cpu time in	273
SCALCG with θ^a achieved minimum cpu time in	303
SCALCG with θ^s and SCALCG with θ^a achieved the same cpu time in	76

Observe that the total number in Table 2 exceeds 500 due to ties for some problems. The performance of these algorithms have been evaluated using the profiles of Dolan and Moré [9]. That is, for each algorithm, we plot the fraction of problems for which the algorithm is within a factor of the best number of iterations and cpu time, respectively. The left side of these Figures gives the percentage of the test problems, out of 500, for which an algorithm is more performant; the right side gives the percentage of the test problems that were succesfully solved by each of the algorithms. Mainly, the right side represents a measure of an algorithm's robustness. In Figures 1 and 2 we compare the performance profiles of SCALCG with θ^s and SCALCG with θ^a refering to the number of function evaluations and cpu time, respectively.

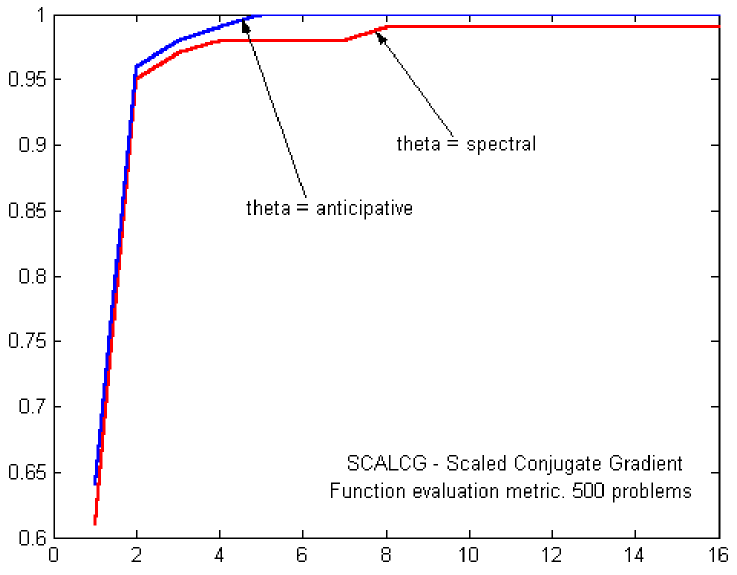

Fig.1. Function evaluations performance profile of SCALCG.
Powell restart. θ^s versus θ^a.

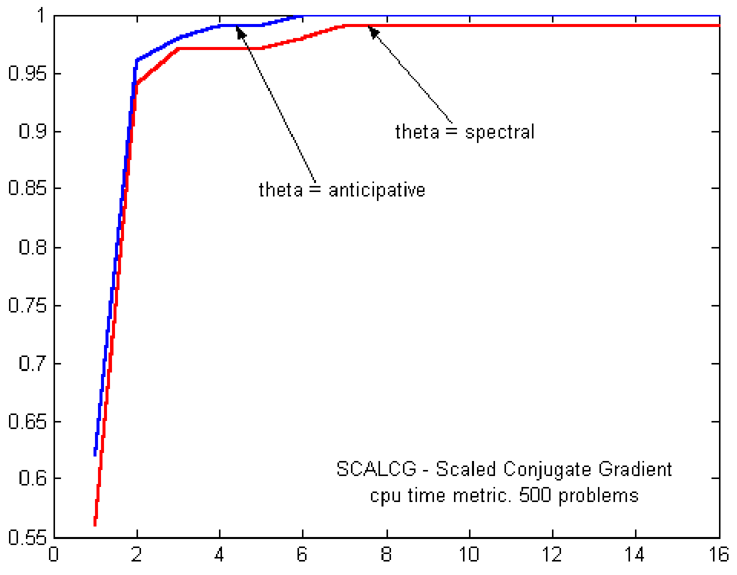

Fig. 2. CPU time performance profile of SCALCG.
Powell restart. θ^s versus θ^a.

The top curve corresponds to the algorithm that solved the most problems in a number of function evaluations (Figure 1) or in a cpu time (Figure 2) that was within a given factor τ of the best number of function evaluations or cpu time, respectively. Since the top curve in Figures 1 and 2 corresponds to SCALCG with θ^a, this algorithm is clearly better than SCALCG with θ^s. However, both codes have similar performances for the value of τ near 1. Therefore, we see that the anticiaptive selection of the scaling parameter which uses the function values compares favourable with the spectral approach.

5.2. SCG with θ^s versus SCG with θ^a.

The direction in SCG code by Birgin and Martínez [4] is computed according to (6), where θ_{k+1} is determined in a spectral manner as in (19), or in an anticipative one like in (24). The Wolfe line search conditions are implemented like in SCALCG with $\sigma_1 = 0.0001$ and $\sigma_2 = 0.9$. In SCG the iterations are stopped whenever:

$$\|g_k\|_2 \le \varepsilon_g \quad \text{or} \quad \alpha_k |g_k^T d_k| \le \varepsilon_f |f(x_{k+1})|, \tag{42}$$

where $\varepsilon_g = 10^{-6}$ and $\varepsilon_f = 10^{-20}$. The original stopping criterion used in SCG was: $\|\nabla f(x_k)\|_2 \le \varepsilon_g max\{1, |f(x_{k+1}|\}$. However, except for the cases in which f vanishes at an optimum, this criterion often leads to a premature termination. In fact, for some problems, when f is large at the starting point, SCG stops the iterations almost immediately, far from the optimum. Therefore, we changed the stopping criteria as in (42). Our numerical experiments show that the Powell restart criterion in the SCG package is not profitable. Therefore we present the numerical results of the SCG algorithms with angle restart criterion. Like in SCALCG the initial guess of the step length at the first iteration is selected as: $1/\|g_0\|$. At the following iterations the starting guess for the step α_k is computed as $\alpha_{k-1}\|d_{k-1}\|_2 / \|d_k\|_2$. Table 3 shows the global characteristics corresponding to these 500 test problems.

Table 3. Global characteristics of SCG with θ^s versus SCG with θ^a.
Angle restart. 500 problems.

Global characteristics	θ^s	θ^a
Total number of iterations	227435	231854
Total number of function evaluations	362557	381055
Total cpu time (seconds)	9634.70	9878.53

Observe that both codes have similar performances. Since the codes only differ in the procedure for θ_{k+1} computation we see that θ_{k+1} computed in an anticipative manner, which is based only on the function values in two successive points, is competitive with the spectral formula which considers the Hessian average between two successive iterations. It is worth saying that out of 500 problems solved by SCG with θ^a in this numerical experiment only for 150 (i.e. 30%) γ_{k+1} in (21) was negative.

5.3. Polak-Ribière with θ^s versus Polak-Ribière with θ^a.

The search direction in the Polak-Ribière algorithm is computed as in (7), where the scaling parameter θ_{k+1} is selected in a spectral or in an anticipative manner. The iterations are stopped whenever (42) is satisfied. Like in SCALCG, the same initial guess of the step length is considered. Table 4 shows the global characteristics corresponding to these 500 test problems.

Table 4. Global characteristics of Polak-Ribière.
Angle restart. 500 problems.

Global characteristics	θ^s	θ^a
Total number of iterations	210208	209014
Total number of function evaluations	361709	359300
Total cpu time (seconds)	9299.45	9251.54

We see that the scaled Polak and Ribiére algorithm with θ^a is better than its variant using θ^s.

5.4. SCALCG versus SCG and CG_DESCENT

In [11,12] Hager and Zhang present a *new conjugate gradient algorithm* and the performance of the Fortran 77 package CG_DESCENT which implements it, for solving (1). In CG_DESCENT the iterations are stopped whenever $\|g_k\|_\infty \leq \varepsilon_g$, where $\varepsilon_g = 10^{-6}$. The Wolfe line search is implemented in two manners: the *classical Wolfe line search* (W) and the *approximate Wolfe line search* (aW), which is based on a very fine interpretation of the numerical issue concerning the first Wolfe condition. In both these aproaches $\sigma_1 = 0.1$ and $\sigma_2 = 0.9.$ These values represent a compromise between the desire for a rapid termination of the line search with a semnificative improvement in the function value. The line search implementing the aproximate Wolfe conditions is very sophisticated, with a lot of parameters which can be specified by the user. In our numerical experiments we considered the values of these parameters given by default. The algorithm is restarted at every multiple of n. In the following we compare the SCALCG (θ^a) with SCG (θ^s) and CG_DESCENT (aW). Table 5 shows the global characteristics, corresponding to these 500 test problems, refering to the total number of iterations, the total number of function evaluations and the total cpu time for these algorithms.

Table 5. Global characteristics of SCALCG (θ^a), SCG (θ^s) and CG_DESCENT (aW). 500 problems.

Global characteristics	SCALCG	SCG	CG_DESCENT
Total number of iterations	121352	227435	153116
Total number of function evaluations	182643	362557	298140
Total cpu time (seconds)	3669.51	9634.70	8411.52

Table 6 illustrates the number of problems, out of 500, for which the above algorithms achieved the minimum number of iterations, the minimum number of function evaluations and the minimum cpu time, respectively.

Table 6. Performance of SCALCG (θ^a), SCG (θ^s) and CG_DESCENT (aW) algorithms. 500 problems.

Performance criterion	**# of problems**
SCALCG (θ^a) achieved minimum # of iterations in	308
SCG (θ^s) achieved minimum # of iterations in	54
CG_DESCENT (aW) achieved minimum # of iterations in	175
SCALCG (θ^a) achieved minimum # of function evaluations in	314
SCG (θ^s)achieved minimum # of function evaluations in	40
CG_DESCENT (aW) achieved minimum # of function evaluations in	167
SCALCG (θ^a) achieved minimum cpu time in	375
SCG (θ^s) achieved minimum cpu time in	36
CG_DESCENT (aW) achieved minimum cpu time in	129

In Figures 3 and 4 we compare the performance profiles of all these three algorithms subject to function evaluations and cpu time metrics, respectively.

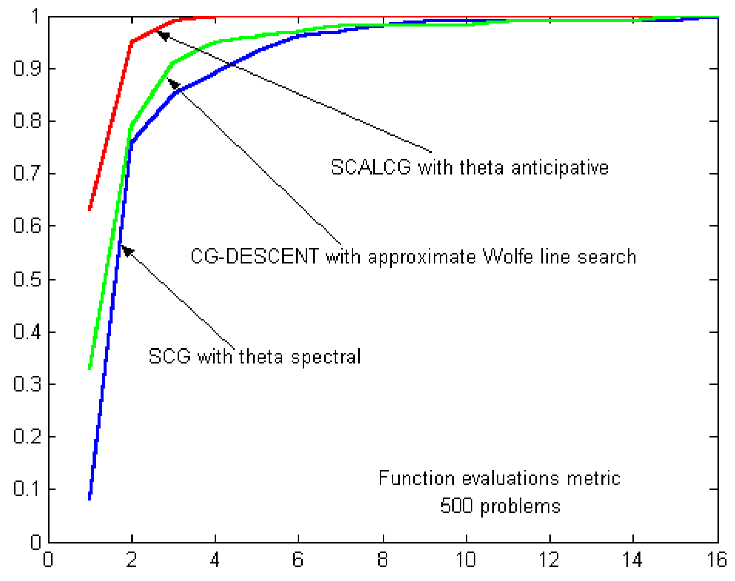

Fig. 3. Performance based on number of function evaluations.
SCALCG (θ^a) versus SCG (θ^s) and CG_DESCENT (aW).

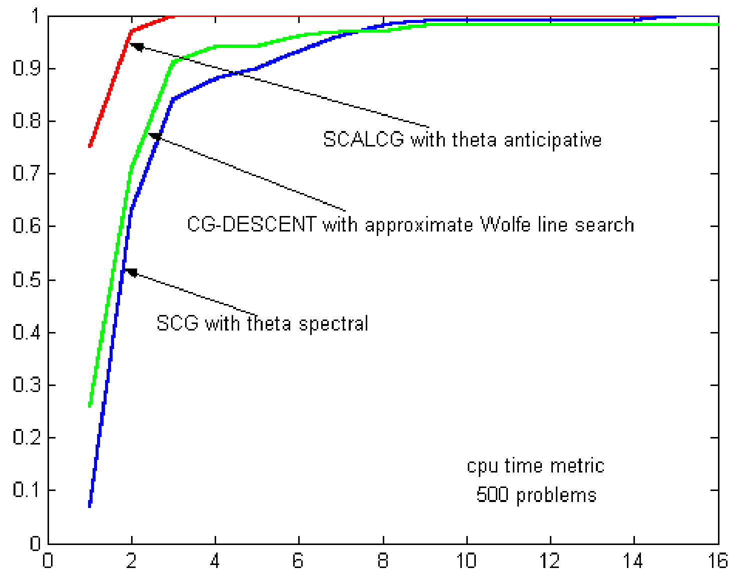

Fig. 4. Performance based on cpu time.
SCALCG (θ^a) versus SCG (θ^s) and CG_DESCENT (aW).

Notice that relative to the function evaluations and cpu time metrics from Figures 3 and 4 it follows that SCALCG (θ^a) is the top performer for all values of τ. The Figures indicate that relative to these metrics SCALCG (θ^a) appears to be the best, followed by CG_DESCENT (aW), and followed by SCG(θ^s).

Now, since the SCALCG and SCG codes use the same line search (with the same values for σ_1 and σ_2 parameters), these codes only differ in their choice of the search direction. Therefore, on average, SCALCG appears to generate a better search direction than SCG. From Table 6 we see that refering to the number of iterations SCALCG is about 5.7

times more performant than SCG. Considering the number of function evaluations it is about 7.85 times more performant and about 10.4 times faster than SCG.

In the function evaluation metric, from Figure 3, we see that for τ near 1, SCALCG is almost twice more performant than CG_DESCENT. Figure 4 gives computational evidence that at least for this set of 500 problems, with dimensions ranging from 10^3 to 10^4, in the cpu time metric SCALCG is about three times faster than CG_DESCENT, i.e. the fraction of problems for which SCALCG achieved the best time is about three times greater than that corresponding to CG_DESCENT. The salient point computationally is that, even the CG_DESCENT uses the loop unrolling to a depth of 5, however SCALCG is faster.

In the line search, more function evaluations are needed by CG_DESCENT with approximate Wolfe to achieve the stopping criterion, while in SCALCG the number of calls of the Wolfe line search subroutine is substantially smaller than the number of iterations, i.e. the initial guess of the step length is accepted as satisfying the Wolfe conditions at the very first iteration of the subroutine. This has a great influence on the cpu time. Concerning the total cpu time for solving this set of 500 problems, from Table 5 we see that SCALCG is almost 2.3 times faster than CG_DESCENT.

The SCALCG and CG_DESCENT algorithms (and codes) differ in many respects. Since both of them use the Wolfe line search (however, implemented in different manners), mainly these codes differ in their choice of the search direction. SCALCG appears to generate a better search direction, on average. The direction d_{k+1} used in SCALCG is more elaborate, it satisfies the quasi-Newton equation in a restart environment. Although the update formulas (9) and (16)-(18) are more complicated, this scheme proved to be more efficient and more robust in numerical experiments. However, since each of these codes are different in the amount of linear algebra required in each iteration and in the relative number of function and its gradient evaluations, it is quite clear that different codes will be superior in different problem sets.

5.5. SCALCG versus SCG, CG_DESCENT and Polak-Ribière

In the following we compare SCALCG (θ^a) with SCG (θ^s), CG_DESCENT (aW) and Polak-Ribière with the spectral selection of the scaling parameter θ_k. Table 7 shows the global characteristics corresponding to these 500 test problems, refering to the total number of iterations, the total number of function evaluations and the total cpu time for these algorithms.

Table 7. Global characteristics of SCALCG (θ^a), SCG (θ^s), CG_DESCENT (aW) and Polak-Ribière (θ^s). 500 problems.

Global characteristics	SCALCG	SCG	CG_DESCENT	Polak-Ribière
Total number of iterations	121352	227435	153116	210208
Total number of function evaluations	182643	362557	298140	361709
Total cpu time (seconds)	3669.51	9634.70	8411.52	9299.45

Table 8 presents the number of problems, out of 500, for which these algorithms achieved the minimum number of iterations, function evaluations and cpu time, respectively.

Table 8. Performance of algorithms. 500 problems.

	Number of problems			
Minimum number of:	**SCALCG**	**CG_DESCENT**	**SCG**	**Polak-Ribiere**
iterations	303	173	52	8
function evaluations	309	165	39	10
cpu time	371	125	34	18

In Figures 5-7 we compare the performance profiles of these algorithms subject to the number of iterations, the number of function evaluations and cpu time, respectively.

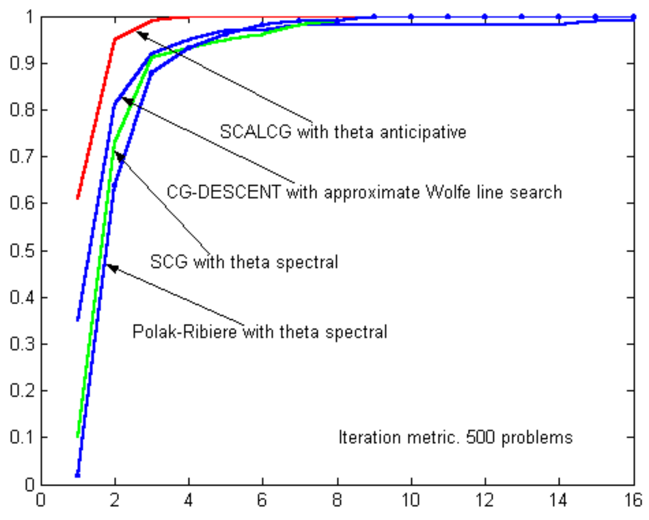

Fig. 5. Performance based on thenumber of iterations.

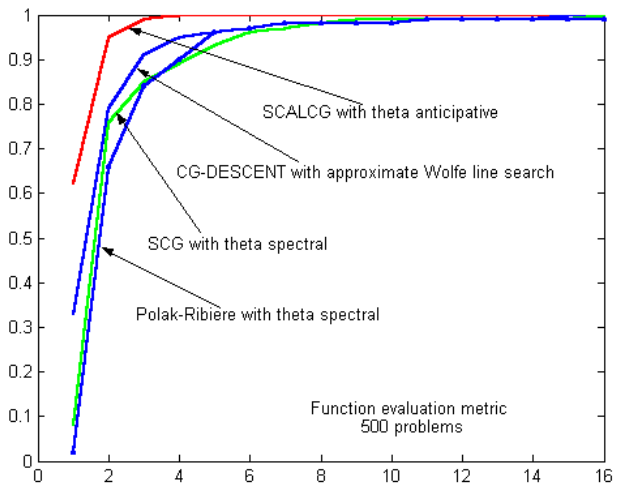

Fig. 6. Performance based on the number of function evaluations.

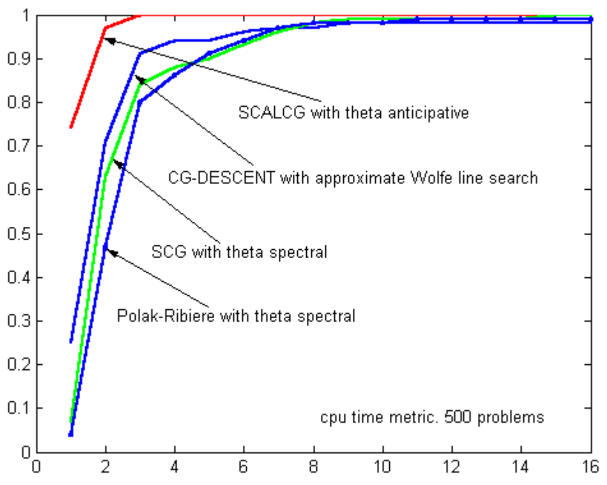

Fig. 7. Performance based on the cpu time.

The top curve corresponds to the algorithm that solved the most problems in a number of iterations (Figure 5) or in a number of function evaluations (Figure 6) or in a cpu time (Figure 7) that was within a given factor of the best number of iterations, function evaluations and cpu time, respectively. Since the top curve in Figures 5, 6 and 7 corrersponds to SCALCG, this algorithm is clearly the best among the algorithms considered in this study.

5.6. Accuracy comparisons

In the next series of experiments we explore the ability of the algorithms to *accurately* solve the problems. We compare SCALCG and CG_DESCENT and consider the following two problems:

$$f_1(x) = \sum_{i=1}^{n-4} (-4x_i - 3)^2 + (x_i^2 + 2x_{i+1}^2 + 3x_{i+2}^2 + 4x_{i+3}^2 + 5x_n^2)^2, \text{ (BDQRTIC - CUTE)}$$

$$f_2 = \sum_{i=1}^{n-1} \sin(x_1 + x_i^2 - 1) + \frac{1}{2}\sin(x_n^2), \text{ (EG2 - CUTE)}$$

with the initial point $x_0 = [1,\dots,1]$. Tables 9a,b and 10a,b contain the number of iterations, cpu time (seconds) and the value of function in optimal point for all these problems, subject to five values of tolerance ε_g on $\|g_k\|_\infty$.

Table 9a. Iteration and solution time versus tolerance ε_g, SCALCG, $f_1(x)$, $n = 10000$.

	θ^s			θ^a		
ε_g	**#iter**	**time**	$f(x^*)$	**#iter**	**time**	$f(x^*)$
10^{-3}	335	32.07	40034.30553831	545	86.73	40034.30553838
10^{-5}	361	50.26	40034.30553829	545	86.67	40034.30553838
10^{-7}	361	50.26	40034.30553829	545	86.67	40034.30553838
10^{-9}	361	50.26	40034.30553829	545	86.67	40034.30553838
10^{-11}	361	50.25	40034.30553829	545	86.62	40034.30553838

Table 9b. Iteration and solution time versus tolerance ε_g, CG_DESCENT, $f_1(x)$, $n = 10000$.

	W			aW		
ε_g	**#iter**	**time**	$f(x^*)$	**#iter**	**time**	$f(x^*)$
10^{-3}	815*	159.01	40034.30554037	2005	412.22	40034.30553834
10^{-5}	815*	159.01	40034.30554037	5259	1097.08	40034.30553825
10^{-7}	815*	159.01	40034.30554037	8932	1897.67	40034.30553825
10^{-9}	815*	159.01	40034.30554037	10020	2140.18	40034.30553825
10^{-11}	815*	158.46	40034.30554037	10048	2136.65	40034.30553825

* Line search fails, too many secant steps. - your tolerance is too strict.

Table 10a. Iteration and solution time versus tolerance ε_g ,
SCALCG, $f_2(x)$, $n = 10000$.

ε_g	θ^s			θ^a		
	#iter	**time**	$f(x^*)$	**#iter**	**time**	$f(x^*)$
10^{-3}	111	4.95	-9998.947391877	107	4.88	-9998.947382045
10^{-5}	123	8.57	-9998.947392267	111	7.36	-9998.947382099
10^{-7}	204	81.01	-9998.947392269	111	7.42	-9998.947382099
10^{-9}	204	81.07	-9998.947392269	111	7.42	-9998.947382099
10^{-11}	204	81.18	-9998.947392269	111	7.42	-9998.947382099

Table 10b. Iteration and solution time versus tolerance ε_g ,
CG_DESCENT, $f_2(x)$, $n = 10000$.

ε_g	W			aW		
	#iter	**time**	$f(x^*)$	**#iter**	**time**	$f(x^*)$
10^{-3}	114	6.92	-9998.999975574	114	6.92	-9998.999975574
10^{-5}	125	8.90	-9998.999985862	161	10.43	-9999.
10^{-7}	125	8.84	-9998.999985862	179	11.59	-9999.
10^{-9}	125	8.90	-9998.999985862	199	12.96	-9999.
10^{-11}	125*	8.73	-9998.999985862	267	16.86	-9999.

* Line search fails, too many secant steps. - your tolerance is too strict.

Some comments are in order.

- These two problems were selected since they illustrate the typical behaviour noticed in the test problems. Observe that both of them have a nonzero optimal function value. When the optimal function value is zero, while the minimizer is not zero, then the estimate $\varepsilon_f |f(x_{k+1})|$ for the error in function value used in the second stopping criterion in (26) can be very poor as the iterates approach the minimizer (where f vanishes).
- Both SCALCG and CG_DESCENT are able to solve these problems subject to different values of tolerance ε_g. However, the results obtained by CG_DESCENT are more accurate. This is due to the line search procedure implementing the Wolfe conditions. The approximate Wolfe conditions are more accurate. In fact, a line search implementing the Wolfe conditions can compute a solution with accuracy on the order of the square root of the machine epsilon. On the other hand, a line search based on the approximate Wolfe conditions can compute a solution with accuracy on the order of the machine epsilon [11].
- Observe that in terms of number of iterations or cpu time, SCALCG is more robust than CG_DESCENT with approximate Wolfe line search. Tolerances lower than 10^{-5} do not change significantly the number of iterations or cpu time of SCALCG. On the other hand, CG_DESCENT is more sensitive to ε_g modification. Reducing ε_g we see that CG_DESCENT requires more iterations, and obviously more cpu time, to get the corresponding accuracy.
- Generally, we see that the number of iterations and cpu time corresponding to SCALCG are lower than the same elements required by CG_DESCENT. This once again illustrates that the crucial element of any unconstrained optimization algorithm is given by the search direction procedure.

5.7. Performance of SCALCG on MINPACK-2 applications

In Tables 11-16 we show the performance of SCALCG, SCG and scaled Polak-Ribière (sPR) algorithms for 6 unconstrained optimization applications from MINPACK-2 collection [2]. In all these numerical experiments we have considered the standard initial point, as it is recommended in MINPACK-2. SCALCG uses the Powell restart criterion and the iterations are stopped according to (26) criteria. SCG and sPR use the angle restart criterion and the iterations are stopped when (42) is satisfied. All these algorithms use the same implementation of the Wolfe line search procedure with the same values of parameters σ_1 and σ_2.

Table 11. Performance of SCALCG, SCG, sPR.
Elastic-Plastic Torsion Problem.

$nx = 100, ny = 100$, $c = 5.$, $n = 10000$.

Algorithm	# iter		# fg		cpu (s)		$f(x^*)$
	θ^s	θ^a	θ^s	θ^a	θ^s	θ^a	
SCALCG	217	255	284	338	12.20	14.45	-0.439163196
SCG	282	261	438	396	25.27	23.01	-0.439163173
sPR	265	235	420	375	24.06	21.42	-0.439162989

$nx = 200, ny = 200$, $c = 5.$, $n = 40000$.

Algorithm	# iter θ^s	# iter θ^a	# fg θ^s	# fg θ^a	cpu (s) θ^s	cpu (s) θ^a	$f(x^*)$
SCALCG	398	473	511	614	87.39	104.80	-0.439267742
SCG	472	425	746	668	170.92	153.18	-0.439265832
sPR	516	391	828	621	188.83	141.76	-0.439264713

Table 12. Performance of SCALCG, SCG, sPR.
Pressure Distribution in a Journal Bearing.

$nx = 100, ny = 100$, $ecc = 0.1$, $b = 10$, $n = 10000$.

Algorithm	# iter		# fg		cpu (s)		$f(x^*)$
	θ^s	θ^a	θ^s	θ^a	θ^s	θ^a	
SCALCG	433	461	567	620	23.51	25.71	-0.282840004
SCG	617	483	956	752	53.11	41.69	-0.282839980
sPR	523	454	838	705	45.86	38.94	-0.282840004

$nx = 200, ny = 200$, $ecc = 0.1$, $b = 10$, $n = 40000$.

Algorithm	# iter θ^s	# iter θ^a	# fg θ^s	# fg θ^a	cpu (s) θ^s	cpu (s) θ^a	$f(x^*)$
SCALCG	876	900	1143	1157	189.11	191.14	-0.282892918
SCG	1117	1133	1746	1779	384.21	390.96	-0.282892622
sPR	980	961	1556	1530	339.82	333.56	-0.282892453

Table 13. Performance of SCALCG, SCG, sPR.
Optimal Design with Composite Materials.

$nx = 100, ny = 100$, $\lambda = 0.008$,, $n = 10000$.

Algorithm	# iter		# fg		cpu (s)		$f(x^*)$
	θ^s	θ^a	θ^s	θ^a	θ^s	θ^a	
SCALCG	628	644	801	815	49.49	50.31	-0.011377244
SCG	1519	1249	2309	1907	168.62	139.19	-0.011377230
sPR	1319	1439	2064	2245	149.62	162.80	-0.011377226

$nx = 200, ny = 200$, $\lambda = 0.008$, $n = 40000$.

Algorithm	# iter θ^s	# iter θ^a	# fg θ^s	# fg θ^a	cpu (s) θ^s	cpu (s) θ^a	$f(x^*)$
SCALCG	1413	939	1753	1157	431.82	285.06	-0.011381240
SCG	3952	3184	6003	4828	1749.27	1407.30	-0.011380973
sPR	3628	4372	5680	6847	1644.47	1979.79	-0.011381028

Table 14. Performance of SCALCG, SCG, sPR.
Inhomogeneous Superconductors.
1-dimensional Ginzburg-Landau problem.

$t = 7, \ n = 1000$.

Algorithm	**# iter**		**# fg**		**cpu (s)**		$f(x^*)$
	θ^s	θ^a	θ^s	θ^a	θ^s	θ^a	
SCALCG	7422	5560	9499	7107	141.32	108.04	0.797584e-05
SCG	10001	10001	15724	15664	230.02	229.92	0.218839e-04
sPR	10001	10001	15948	15922	234.81	237.72	0.646897e-04

Table 15. Performance of SCALCG, SCG, sPR.
Leonard-Jones Cluster Problem.

ndim=3, natoms =1000, $n = 3000$.

Algorithm	**# iter**		**# fg**		**cpu (s)**		$f(x^*)$
	θ^s	θ^a	θ^s	θ^a	θ^s	θ^a	
SCALCG	1252	1583	2002	2007	383.16	384.37	-6622.567428
SCG	4552	4656	7103	7229	1117.24	1137.89	-6634.532271
sPR	2553	2917	4058	4591	637.91	721.99	-6621.582203

Table 16. Performance of SCALCG, SCG, sPR.
Steady State Combustion. Solid fuel ignition.

$nx = 100, ny = 100, \lambda = 0.07, \quad n = 10000$.

Algorithm	**# iter**		**# fg**		**cpu (s)**		$f(x^*)$
	θ^s	θ^a	θ^s	θ^a	θ^s	θ^a	
SCALCG	312	266	408	348	33.28	28.40	-0.070086367
SCG	340	288	527	445	50.75	42.85	-0.070086356
sPR	388	360	611	570	58.60	54.60	-0.070086368
$nx = 200, ny = 200, \ \lambda = 0.07, \ n = 40000$.							
SCALCG	546	460	720	586	233.65	190.05	-0.070086331
SCG	589	503	927	787	354.21	300.88	-0.070085961
sPR	664	555	1061	875	404.47	334.17	-0.070085849

From Tables 11-16 we see that for solving these 10 MINPACK-2 applications the top performer is SCALCG (θ^a).

6. Conclusion

We have presented a scaled memoryless BFGS preconditioned conjugate gradient algorithm, *SCALCG - scaled conjugate gradient algorithm*, for solving large-scale unconstrained optimization problems. SCALCG can be considered as a modification of the best algorithm by Birgin and Martínez [4], which is mainly a scaled variant of Perry's [16], and of the Dai and Liao [6] $(t = 1)$, in order to overcome the lack of positive definiteness of the matrix defining the search direction. This modification takes the advantage of the quasi-Newton BFGS updating formula. Using the restart technology of Beale-Powell, we get a scaled conjugate gradient algorithm in which the parameter scaling the gradient is selected as spectral gradient or in an anticipative manner by means of a formula using the function values in two successive points. Although the update formulas (9) and (16)-(18) are more complicated the scheme proved to be efficient and robust in numerical experiments. The algorithm implements the Wolfe conditions and we have proved that the steps are along the descent directions.

The performance profiles for our scaled conjugate gradient algorithm were higher than those of *SCG - spectral conjugate gradient method* by Birgin and Martínez, *CG_DESCENT - conjugate gradient with guaranteed descent* by Hager and Zhang and *scaled Polak-Ribière* for a test set consisting of 500 unconstrained optimization problems (some of them form CUTE) with dimensions ranging between 10^3 and 10^4. Relative to the function evaluations and cpu time metrics, SCALCG is the top performer. In the function evaluations metric SCALCG performs almost twice better than CG_DESCENT. Refering to the cpu time metric SCALCG is about three times faster than CG_DESCENT. The better performance of SCALCG, relative to SCG and CG_DESCENT, in the time and function evaluation metrics, is connected with the search direction procedure of selection. The direction d_{k+1} used in SCALCG is highly elaborate, it satisfies the quasi-Newton equation in a restart environment. However, CG_DESCENT with approximate Wolfe line search is more accurate than SCALCG. Using a line search based on the approximate Wolfe conditions, CG_DESCENT computes a solution with accuracy of the order of the machine epsilon. On the other hand, SCALCG, implementing the Wolfe line search, computes a solution with accuracy on the order of the square root of the machine epsilon.

Finally, as each of these four codes considered here are different in many respects, mainly in the amount of linear algebra required in each iteration, in the Wolfe line search parameters σ_1 and σ_2, in the stopping criteria and in the relative number of function and its gradient evaluations, it is quite clear that different codes will be superior in different problem sets. Generally, one needs to solve thousands different problems before trends begin to emerge. For any particular problem almost any method can win. However, we have a strong computational evidence that our SCALCG algorithm is the top performer among these conjugate gradient algorithms.

References

1. N. Andrei, "*A new gradient descent method for unconstrained optimization*", ICI Technical Report, March 2004.
2. B.M.Averick, R.G. Carter, J.J. Moré and G.L. Xue, "*The MINPACK-2 test problem collection*", Argonne National Laboratory, Preprint MCS-P153-0692, June 1992.
3. J. Barzilai and J.M. Borwein, "*Two point step size gradient method*", IMA J. Numer. Anal., 8, pp.141-148, 1988.
4. E. Birgin and J.M. Martínez, "*A spectral conjugate gradient method for unconstrained optimization*", Applied Math. and Optimization, 43, pp.117-128, 2001
5. I. Bongartz, A.R. Conn, N.I.M. Gould and P.L. Toint, "*CUTE: constrained and unconstrained testing environments*", ACM Trans. Math. Software, 21, pp.123-160, 1995.
6. Y.H. Dai and L.Z. Liao, "*New conjugate conditions and related nonlinear conjugate gradient methods*", Appl. Math. Optim., vol. 43 pp.87-101, 2001.
7. Y.H. Dai and Y. Yuan, "*Convergence properties of the Beale-Powell restart algorithm*", Science in China (series A) 41, (11), 1142-1150, 1998.
8. Y.H. Dai, J.Y. Yuan and Y-X. Yuan, "*Modified two-point stepsize gradient methods for unconstrained optimization*", Computational Optimization and Applications, 22, pp.103-109, 2002.
9. E.D. Dolan and J.J.Moré, "*Benchmarking optimization software with performance profiles*", Math. Programming, 91, pp. 201-213, 2002.
10. R. Fletcher and C.M. Reeves, "*Function minimization by conjugate gradients*", Comput. J. 7, pp. 149-154, 1964.
11. W.W. Hager and H. Zhang, "*A new conjugate gradient method with guaranteed descent and an efficient line search*", University of Florida, Department of Mathematics, November 17, 2003 (theory and comparisons), revised July 3, 2004.

12. W.W. Hager and H. Zhang, "*CG-DESCENT, A conjugate gradient method with guaranteed descent (algorithm details and comparisons)*", University of Florida, Department of Mathematics, January 15, 2004.
13. M.R. Hestenes and E. Stiefel, "*Methods of conjugate gradients for solving linear systems*", J. Research Nat. Bur. Standards Sec. B. 48, pp. 409-436, 1952.
14. S.S. Oren and D.G. Luenberger, "*Self-scaling variable metric algorithm. Part I*", Management Sci., 20, pp.845-862, 1976.
15. S.S. Oren and E. Spedicato, "*Optimal conditioning of self-scaling variable metric algorithms*", Math. Programming, 10, pp.70-90, 1976.
16. J.M. Perry, "*A class of conjugate gradient algorithms with a two step variable metric memory*", Discussion paper 269, Center for Mathematical Studies in Economics and Management Science, Northwestern University, 1977.
17. E. Polak and G. Ribière, "*Note sur la convergence de methods de directions conjugres*", Revue Francaise Informat. Reserche Opérationnelle 16, pp. 35-43, 1969.
18. M.J.D. Powell, "*Some convergence properties of the conjugate gradient method*". Math. Programming, 11, pp.42-49, 1976.
19. M.J.D. Powell, "*Restart procedures for the conjugate gradient method*", Math. Programming, 12, pp.241-254, 1977.
20. M. Raydan, "*The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem*", SIAM J. Optim., 7, 26-33, 1997.
21. D.F. Shanno, "*Conjugate gradient methods with inexact searches*", Mathematics of Operations Research, vol. 3, pp.244-256, 1978.
22. D.F. Shanno, "*On the convergence of a new conjugate gradient algorithm*", SIAM J. Numer. Anal. vol. 15, pp.1247-1257, 1978.
23. D.F. Shanno and K.H. Phua, "*Algorithm 500, Minimization of unconstrained multivariate functions [E4]*", ACM Trans. on Math. Soft., 2, pp.87-94, 1976.
24. P. Wolfe, "*Convergence conditions for ascent methods*", SIAM Rev., 11, pp.226-235, 1969.
25. P. Wolfe, "*Convergence conditions for ascent methods II: some corrections*", SIAM Rev. 13, pp.185-188, 1971.

September 8, 2005