SCALCG: A Nonlinear Scaled 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,
E-mail: nandrei@ici.ro

SCALCG is a package of Fortran 77 subroutines for solving large-scale unconstrained optimization problems by means of a scaled conjugate gradient method. In this algorithm the searching direction is a combination of the Newton direction and that corresponding to the conjugate gradient method with scalled gradient, which satisfy the quasi-Newton equation. The steplength is computed using the Wolfe line search conditions. The computational scheme is embedded into the restart philosophy of Beale-Powell [4,23]. The parameter scaling the gradient is selected as spectral gradient, as suggested by Raydan [24], or in an anticipative manner by means a formula using the function values in two successive points [1]. In very mild conditions it is shown that, for strongly convex functions, the algorithm is global convergent. Computational results and performance profile by Dolan and Moré [8], for a test set consisting of 700 unconstrained optimization problems (some of them from CUTE [6]), show that this new scaled nonlinear conjugate gradient algorithm substantially outperforms known conjugate gradient methods including: spectral conjugate gradient SCG by Birgin and Martinez [5], scaled Fletcher and Reeves, and Polak and Ribière, CG_DESCENT by Hager and Zhang [11,12] and CONMIN by Shanno and Phua [27].

Categories and Subject Descriptors: G 1.6 [**Numerical Analysis**]: Optimization - *Unconstrained optimization, gradient methods*; G.4 [**Mathematics of Computing**]: Mathematical Software - *Algorithm design and analysis; Efficiency; Reliability and robustness.*

General Terms: Algorithms

Additional Key Word and Phrases: Conjugate gradient method, quasi-Newton method, scaling, Rayleigh quotient

1. INTRODUCTION

In [1] we introduced a new scaled nonlinear conjugate gradient method for solving large-scale unconstrained optimization problem:

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

where $f: R^n \to R$ is continuously differentiable and its gradient is available. SCALCG is a double-precision Fortran package for solving (1) implementing this algorithm. The searching direction is a combination of the Newton direction and that corresponding to the conjugate gradient method with scalled gradient, which satisfies the quasi-Newton equation. The steplength is computed to satisfy the Wolfe line search conditions. The computational scheme is embedded into the restart philosophy of Beale-Powell [4,23]. The parameter scaling the gradient is selected as spectral gradient, as suggested by Raydan [24], or in an anticipative manner by means of a formula using the function values in two successive points [1]. The algorithm in SCALCG has been developed for functions with a large number of variables. To access the package, the user must supply a subroutine for function of the problem and its derivative calculations, as well as a driver program for calling the package with appropriate options specifications and input parameters. The performance of the package depends on the initial guess and on the input parameters, all of them being provided by the user.

In the next section, we describe the method and prove that the direction is a descent one. In Section 3 we present the algorithm. Section 4 is dedicated to present the convergence analysis for strongly convex functions, and in Section 5 we provide some details of the package as well as a numerical study and comparisons with conjugate gradient algorithms Fletcher-Reeves [9] and Polak-Ribière [21] and Polyak [22], and packages: SCG by Birgin and Martinez [5], CG_DESCENT by Hager and Zhang [12] and CONMIN by Shanno and Phua [27].

2. THE METHOD

The algorithm generate 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, and β_k is a scalar parameter. 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 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 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 Martinez [5] 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 Martinez [5] in their SCG (spectral conjugate gradient) package for unconstrained optimization, where θ_{k+1} is selected in a spectral manner, as suggested by Raydan [24]. The following particularizations can be remarked. If $\theta_{k+1} = 1$, then (6) is the direction considered by Perry [20]. At the same time we see that (6) is the direction given by Dai and Liao [7] 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 (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 [21]. If $s_j^T g_{j+1} = 0,\ j = 0,1,\ldots,k,$ 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 [9]. Therefore, (6) is a general formula for direction computation in a conjugate gradient manner including the classical Fletcher and Reeves [9], and Polak and Ribière [21] formulas. We see that the direction given by (6) can be written as:

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

where

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

If $\theta_{k+1} = 1,$ we have:

$$d_{k+1} = -\left[I - \frac{s_k y_k^T}{y_k^T s_k} + \frac{s_k s_k^T}{y_k^T s_k}\right] g_{k+1}, \tag{11}$$

which is exactly the *Perry formula*. A major difficulty with (9) is that the matrix Q_{k+1}, defined by (10), is not symmetric and hence not positive definite. Thus, the directions d_{k+1} from (9) are not necessarily descent directions and therefore numerical instability can result. Besides, another difficulty arising from this lack of symmetry is that the true quasi-Newton equation $H_{k+1} y_k = s_k$ is not satisfied.

In order to overcome this difficulty and to get a true quasi-Newton updating we first make the matrix Q_{k+1} from (10) symmetric as follows:

$$Q_{k+1} = \theta_{k+1} I - \theta_{k+1} \frac{s_k y_k^T + y_k s_k^T}{y_k^T s_k} + \frac{s_k s_k^T}{y_k^T s_k}. \tag{12}$$

Now, we force Q_{k+1} to satisfy the quasi-Newton equation $H_{k+1} y_k = s_k$ yielding the following symmetric update:

$$Q_{k+1}^* = \theta_{k+1} I - \theta_{k+1} \frac{y_k s_k^T + s_k y_k^T}{y_k^T s_k} + \left[1 + \theta_{k+1} \frac{y_k^T y_k}{y_k^T s_k}\right] \frac{s_k s_k^T}{y_k^T s_k}. \tag{13}$$

By direct computation it is very easy to proof that Q_{k+1}^* satisfy the quasi-Newton equation, i.e. $Q_{k+1}^* y_k = s_k$. Notice that

$$d_{k+1} = -Q^*_{k+1} g_{k+1} \tag{14}$$

does not actually require the matrix Q^*_{k+1}, i.e. the direction d_{k+1} can be computed as:

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \theta_{k+1}\left(\frac{g^T_{k+1}s_k}{y^T_k s_k}\right) y_k - \left[\left(1+\theta_{k+1}\frac{y^T_k y_k}{y^T_k s_k}\right)\frac{g^T_{k+1}s_k}{y^T_k s_k} - \theta_{k+1}\frac{g^T_{k+1}y_k}{y^T_k s_k}\right] s_k, \tag{15}$$

involving only 4 scalar products. Again observe that if $g^T_{k+1}s_k = 0$, then (15) reduces to:

$$d_{k+1} = -\theta_{k+1} g_{k+1} + \theta_{k+1}\frac{g^T_{k+1}y_k}{y^T_k s_k} s_k. \tag{16}$$

Thus, in this case, the effect is simply one of multiplying the Hestens and Stiefel [14] search direction by a positive scalar.

In order to ensure the convergence of the algorithm (2), with d_{k+1} given by (15), we need to constrain the choice of α_k. We consider line searches that satisfy the Wolfe conditions [29,30]:

$$f(x_k + \alpha_k d_k) - f(x_k) \le \sigma_1 \alpha_k g^T_k d_k, \tag{17}$$

$$\nabla f(x_k + \alpha_k d_k)^T d_k \ge \sigma_2 g^T_k d_k, \tag{18}$$

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

Theorem 1. *Suppose that α_k in (2) satisfies the Wolfe conditions (17) and (18), then the direction d_{k+1} given by (15) is a descent direction.*

Proof: Since $d_0 = -g_0$, we have $g^T_0 d_0 = -\|g_0\|^2 \le 0$. Multiplying (15) by g^T_{k+1}, we have

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

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 $u = (s^T_k y_k) g_{k+1}$ and $v = (g^T_{k+1}s_k) y_k$ we get:

$$g^T_{k+1}d_{k+1} \le -\frac{(g^T_{k+1}s_k)^2}{y^T_k s_k}. \tag{19}$$

But, by Wolfe condition (18), $y^T_k s_k > 0$. Therefore, $g^T_{k+1}d_{k+1} < 0$ for every $k = 0,1,\ldots$ ■

Observe that the second Wolfe condition (18) is crucial for the descent character of direction (15). Moreover, we see that the estimation (19) is independent by 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 [23] is to test if there is very little orthogonality left between the curent gradient and the previous one. At step r when:

$$\left|g^T_{r+1}g_r\right| \ge 0.2\|g_{r+1}\|^2, \tag{20}$$

we restart the algorithm using the direction given by (15). Another restarting procedure, considered by Birgin and Martinez [5], consists of testing if the angle between the current direction and $-g_{k+1}$ is not acute enough. Therefore, at step r when:

$$d^T_r g_{r+1} > -10^{-3}\|d_r\|_2 \|g_{r+1}\|_2, \tag{21}$$

the algorithm is restarted using the direction given by (15).

At step r when one of the two criteria (20) or (21) is satisfied, the direction is computed as in (15). For $k \geq r+1$, we consider the same philosophy used to get (13), 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 $k \geq r+1$, is computed using a double update scheme as:

$$d_{k+1} = -H_{k+1} g_{k+1}, \tag{22}$$

where

$$H_{k+1} = H_{r+1} - \frac{H_{r+1} y_k s_k^T + s_k y_k^T H_{r+1}}{y_k^T s_k} + \left[1 + \frac{y_k^T H_{r+1} y_k}{y_k^T s_k}\right] \frac{s_k s_k^T}{y_k^T s_k}. \tag{23}$$

and

$$H_{r+1} = \theta_{r+1} I - \theta_{r+1} \frac{y_r s_r^T + s_r y_r^T}{y_r^T s_r} + \left(1 + \theta_{r+1} \frac{y_r^T y_r}{y_r^T s_r}\right) \frac{s_r s_r^T}{y_r^T s_r}. \tag{24}$$

As above, observe that this computational scheme does not involve any matrix. Indeed, $H_{r+1} g_{k+1}$ and $H_{r+1} y_k$ can be computed as:

$$\begin{aligned} v \equiv H_{r+1} g_{k+1} &= \theta_{r+1} g_{k+1} - \theta_{r+1} \left(\frac{g_{k+1}^T s_r}{y_r^T s_r}\right) y_r \\ &+ \left[\left(1 + \theta_{r+1} \frac{y_r^T y_r}{y_r^T s_r}\right) \frac{g_{k+1}^T s_r}{y_r^T s_r} - \theta_{r+1} \frac{g_{k+1}^T y_r}{y_r^T s_r}\right] s_r, \end{aligned} \tag{25}$$

and

$$\begin{aligned} w \equiv H_{r+1} y_k &= \theta_{r+1} y_k - \theta_{r+1} \left(\frac{y_k^T s_r}{y_r^T s_r}\right) y_r \\ &+ \left[\left(1 + \theta_{r+1} \frac{y_r^T y_r}{y_r^T s_r}\right) \frac{y_k^T s_r}{y_r^T s_r} - \theta_{r+1} \frac{y_k^T y_r}{y_r^T s_r}\right] s_r, \end{aligned} \tag{26}$$

involving 6 scalar products. With these the direction (22), at any nonrestart step, can be computed as:

$$d_{k+1} = -v + \frac{(g_{k+1}^T s_k) w + (g_{k+1}^T w) s_k}{y_k^T s_k} - \left(1 + \frac{y_k^T w}{y_k^T s_k}\right) \frac{g_{k+1}^T s_k}{y_k^T s_k} s_k, \tag{27}$$

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 (27) is well defined and it is always a descent direction.

In the following we shall 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:

$$\theta_{k+1} = \frac{s_k^T s_k}{y_k^T s_k}. \tag{28}$$

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 (28) is well defined.

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

$$f(x_{k+1}) = f(x_k) + \alpha_k g_k^T d_k + \frac{1}{2} \alpha_k^2 d_k^T \nabla^2 f(z) d_k, \tag{29}$$

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 enough small 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:

$$\gamma_{k+1} = \frac{2}{d_k^T d_k} \frac{1}{\alpha_k^2} \left[f(x_{k+1}) - f(x_k) - \alpha_k g_k^T d_k \right]. \tag{30}$$

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 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 that γ_{k+1} to be positive. To get a value for η_k let us select a real $\delta > 0$, "enough small", but comparable with the value of the function, and consider

$$\eta_k = \frac{1}{g_k^T d_k} \left[f(x_k) - f(x_{k+1}) + \alpha_k g_k^T d_k + \delta \right], \tag{31}$$

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

$$\gamma_{k+1} = \frac{2}{d_k^T d_k} \frac{1}{(\alpha_k - \eta_k)^2} \left[f(x_{k+1}) - f(x_k) - (\alpha_k - \eta_k) g_k^T d_k \right]. \tag{32}$$

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

$$\theta_{k+1} = \frac{1}{\gamma_{k+1}}, \tag{33}$$

where γ_{k+1} is given by (30) or (32).

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},

$$\gamma_{k+1} \leq 2L. \tag{34}$$

Proof: From (30) 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 (33) 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.

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 (17) and (18). 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 (28) or an anticipative (33) approach.
Step 5. Restart direction. Compute the (restart) direction d_k as in (15).
Step 6. Line search. Compute the initial guess of the step-length as:

$$\alpha_k = \alpha_{k-1}\|d_{k-1}\|_2 / \|d_k\|_2.$$

Using this initialization compute α_k satisfying the Wolfe conditions (17) and (18). 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 (20), or the angle restart criterion (21), is satisfied, then go to step 4 (a restart step); otherwise continue with step 10 (a normal step).
Step 10. Normal direction. Compute:

$$v = \theta g_k - \theta\left(\frac{g_k^T s}{y^T s}\right) y + \left[\left(1 + \theta \frac{y^T y}{y^T s}\right)\frac{g_k^T s}{y^T s} - \theta \frac{g_k^T y}{y^T s}\right] s,$$

$$w = \theta y_k - \theta\left(\frac{y_{k-1}^T s}{y^T s}\right) y + \left[\left(1 + \theta \frac{y^T y}{y^T s}\right)\frac{y_{k-1}^T s}{y^T s} - \theta \frac{y_{k-1}^T y}{y^T s}\right] s,$$

and

$$d_k = -v + \frac{(g_k^T s_{k-1})w + (g_k^T w)s_{k-1}}{y_{k-1}^T s_{k-1}} - \left(1 + \frac{y_{k-1}^T w}{y_{k-1}^T s_{k-1}}\right)\frac{g_k^T s_{k-1}}{y_{k-1}^T s_{k-1}} s_{k-1}.$$

Step 11. Line search. Compute the initial guess of the step-length as:

$$\alpha_k = \alpha_{k-1}\|d_{k-1}\|_2 / \|d_k\|_2.$$

Using this initialization compute α_k satisfying the Wolfe conditions (17) and (18). 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 steplength 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, considered for the first time by Shanno and Phua in CONMIN [27], prove to be one of the best. Concerning the stopping criterion used in steps 3, 8 and 12 one could consider any of the following tests:

$$\|g_k\|_\infty \leq \varepsilon_g \quad \text{or} \quad \alpha_k |g_k^T d_k| \leq \varepsilon_f |f(x_{k+1})|, \tag{35a}$$

$$\|g_k\|_\infty \leq max\{\varepsilon_g, \varepsilon_f \|g_0\|_\infty\}, \tag{35b}$$

$$\|g_k\|_2 \leq \varepsilon_g, \tag{35c}$$

where ε_f and ε_g are tolerances specified by the user. ε_f specify the desired accuracy of the computed solution, ε_g specify the desired accuracy for the gradient norm. The second criterion in (35a) says that the estimated change in the function value is insignificant compared to the function value itself.

4. CONVERGENCE ANALYSIS FOR STRONGLY CONVEX FUNCTIONS

Assume that f is strongly convex and Lipschitz continuous on the level set

$$L_0 = \{x \in R^n : f(x) \leq f(x_0)\}. \tag{36}$$

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

$$(\nabla f(x) - \nabla f(y))^T (x-y) \geq \mu \|x-y\|^2 \tag{37}$$

and

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x-y\|, \tag{38}$$

for all x and y from L_0. For the convenience of the reader we include here the following lemma (see [12]).

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

$$\|\nabla f(x) - \nabla f(x_k)\| \leq L\|x - x_k\|, \tag{39}$$

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 (18), then

$$\alpha_k \geq \frac{1-\sigma_2}{L}\frac{|g_k^T d_k|}{\|d_k\|^2}. \tag{40}$$

Proof: Subtracting $g_k^T d_k$ from both sides of (18) and using the Lipschitz condition we have

$$(\sigma_2 - 1) g_k^T d_k \le (g_{k+1} - g_k)^T d_k \le L\alpha_k \|d_k\|^2. \tag{41}$$

Since d_k is a descent direction and $\sigma_2 < 1$, (40) follows immediately from (41). ■

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 (15) satisfies:*

$$\|d_{k+1}\| \le \left(\frac{2}{\mu} + \frac{2L}{\mu^2} + \frac{L^2}{\mu^3} \right) \|g_{k+1}\|. \tag{42}$$

Proof: By Lipschitz continuity (38) we have

$$\|y_k\| = \|g_{k+1} - g_k\| = \|\nabla f(x_k + \alpha_k d_k) - \nabla f(x_k)\| \le L\alpha_k \|d_k\| = L\|s_k\|. \tag{43}$$

On the other hand, by strong convexity (37)

$$y_k^T s_k \ge \mu \|s_k\|^2. \tag{44}$$

Selecting θ_{k+1} as in (28), it follows that

$$\theta_{k+1} = \frac{s_k^T s_k}{y_k^T s_k} \le \frac{\|s_k\|^2}{\mu \|s_k\|^2} = \frac{1}{\mu}. \tag{45}$$

Now, using the triangle inequality and the above estimates (43)-(45), after some algebra on $\|d_{k+1}\|$, where d_{k+1} is given by (15), we get (42). ■

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 (15) satisfies:*

$$\|d_{k+1}\| \le \left(\frac{1}{m} + \frac{2L}{m\mu} + \frac{1}{\mu} + \frac{L^2}{m\mu^2} \right) \|g_{k+1}\|. \tag{46}$$

Proof: By strong convexity on L_0, there exists the constant $m > 0$, such that $\nabla^2 f(x) \ge mI$, for all $x \in L_0$. Therefore, for every k, $\gamma_{k+1} \ge m$. Now, from (33) we see that, for all k,

$$\theta_{k+1} \le \frac{1}{m}. \tag{47}$$

With this, like in lemma 2, we get (46). ■

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 (15) and the step length α_k selected to satisfy the Wolfe conditions (17) and (18), then either $g_k = 0$ for some k, or* $\lim_{k \to \infty} g_k = 0$.

Proof: Suppose $g_k \ne 0$ for all k. By strong convexity we have

$$y_k^T d_k = (g_{k+1} - g_k)^T d_k \ge \mu\alpha_k \|d_k\|^2. \tag{48}$$

By theorem 1, $g_k^T d_k < 0$. Therefore, the assumption $g_k \ne 0$ implies $d_k \ne 0$. Since $\alpha_k > 0$, from (48) 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 (17) we have

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

Considering the lower bound for α_k given by (40) in lemma 1, and having in view that d_k is a descent direction, it follows that

$$\sum_{k=1}^{\infty} \frac{\left|g_k^T d_k\right|^2}{\left\|d_k\right\|^2} < \infty. \tag{49}$$

Now, from (19), using the inequality of Cauchy and (44) we get

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

Therefore, from (49) it follows that

$$\sum_{k=0}^{\infty} \frac{\left\|g_k\right\|^4}{\left\|d_k\right\|^2} < \infty. \tag{50}$$

Now, inserting the upperbound (42), or (46), for d_k in (50) yields

$$\sum_{k=0}^{\infty} \left\|g_k\right\|^2 < \infty,$$

which completes the proof. ■

For general functions the convergence of the algorithm is comming from theorem 1 and the restart procedure. Our algorithm is very close to Perry/Shanno computational scheme. Therefore, for convex functions and under inexact line search it is global convergent. If restarts are employed, the algorithm is convergent, but the speed of convergence can decrease. On the other hand, for general functions that are bounded from below with bounded level sets and bounded second partial derivatives, the convergence of the proposed algorithm can be established using exactly the same analysis given by Shanno in [26].

5. NUMERICAL STUDY AND COMPARISONS

In this section we present a numerical study concerning the performance of *SCALCG - scaled conjugate gradient algorithm* on a number of 700 test unconstrained optimization problems. At the same time, we compare the performance of SCALCG to *SCG* (betatype=1,Perry-M1) *the best spectral conjugate gradient algorithm* (6) by Birgin and Martinez [5], to *CG_DESCENT - a conjugate gradient method with guaranteed descent* by Hager and Zhang [11,12], to *scaled Polak-Ribière algorithm* (7) and *scaled Fletcher-Reeves algorithm* (8) and to *CONMIN conjugate gradient algorithm* by Shanno and Phua [27].

The SCALCG code is authored by Andrei, while the SCG is co-authored by Birgin and Martinez and CG_DESCENT is co-authored by Hager and Zhang. Polak-Ribiere and Fletcher-Reeves algorithms are implemented into the frame of SCG. CONMIN is authored by Shanno and Phua [27]. All codes are written in Fortran and compiled with f77 (default compiler settings) on a work station 1.8Ghz.

SCALCG code implements both the scaled conjugate gradient with spectral choice of scaling parameter θ_{k+1}, as well as with anticipative choice of this parameter, using Powell restart criterion. Basically, SCALCG can be described by the following components: an user-supplied driver, the main subroutine implementing SCALCG algorithm, a line-search subroutine implementing the Wolfe line search, a subroutine specifying the function to be minimized and its gradient, and a subroutine with the initial guess. The line search is implemented in the same manner as that given by Shanno and Phua in their CONMIN package.

In order to compare SCALCG with SCG, scaled Polak-Ribière and scaled Fletcher-Reeves we manufactured a new SCG code of Birgin and Martinez by introducing a sequence of code to compute the θ_{k+1} anticipative according to (33) and the Powell restart criterion. Therefore, SCG code implements both restart criteria: Powell (20) and angle (21). CG_DESCENT code contains both variants implementing the Wolfe line search and the variant corresponding to approximate Wolfe conditions, using restarting at every n iterations. On the other hand, CONMIN implements a BFGS modification of Perry method [20] with Wolfe line search and Powell restart criterion.

The test problems are the unconstrained problems in the CUTE [6] library, along with other large-scale optimization test problems. We selected 70 large-scale unconstrained optimization test problems (22 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, number of restart iterations, 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/scalcgt

In the following we present the numerical performances of all these codes, including the performance profiles of Dolan and Moré [8] subject to the number of iterations, the number of function evaluations and cpu time metrics, respectively. For SCALCG and CONMIN we present their performances on some applications from MINPACK-2 library.

5.1. SCALCG with θ^s versus SCALCG with θ^a

In SCALCG code the computations are terminated as soon as (35a) is 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 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 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 700 test problems, refering to the total number of iterations, total number of function evaluations and total cpu time for these algorithms.

Table 1. Global characteristics of SCALCG with θ^s versus SCALCG with θ^a. 700 problems.

Global characteristics	θ^s	θ^a
Total number of iterations	644287	628485
Total number of function evaluations	900365	837906
Total cpu time (seconds)	3056.13	2830.91

Out of 700 problems solved in this set of experiments the criterion $\|g_k\|_\infty \le \varepsilon_g$ in (35a) stopped the iterations for 614 problems, i.e. 87.7%, in case of SCALCG with θ^s, and for 587 problems, i.e. 83.8%, in case of SCALCG with θ^a.

Table 2 shows the number of problems, out of 700, for which SCALCG with θ^s and SCALCG with θ^a achieved the minimum number of iterations, minimum number of function evaluations and the minimum cpu time, respectively.

Table 2. Performance of SCALCG algorithms.
700 problems.

Performance criterion	# of problems
SCALCG (θ^s) achieved minimum # of iterations in	423
SCALCG (θ^a) achieved minimum # of iterations in	488
SCALCG (θ^s) and SCALCG (θ^a) achieved the same # of iterations in	211
SCALCG (θ^s) achieved minimum # of function evaluations in	416
SCALCG (θ^a) achieved minimum # of function evaluations in	474
SCALCG (θ^s) and SCALCG (θ^a) achieved the same # of function evaluations in	190
SCALCG (θ^s) achieved minimum cpu time in	384
SCALCG (θ^a) achieved minimum cpu time in	484
SCALCG (θ^s) and SCALCG (θ^a) achieved the same cpu time in	168

Observe that the total number in Table 2 exceeds 700 due to ties for some problems. The performance of these algorithms have been evaluated using the profiles of Dolan and Moré [8]. 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 700, 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.

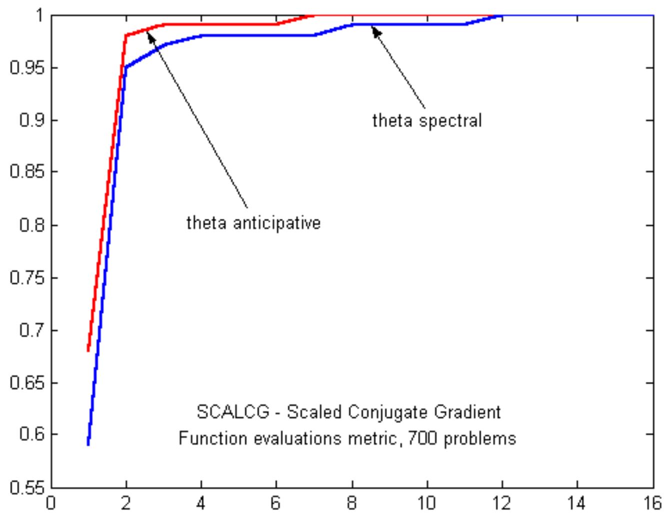

Fig.1. Function evaluations performance profile of SCALCG.
θ^s versus θ^a .

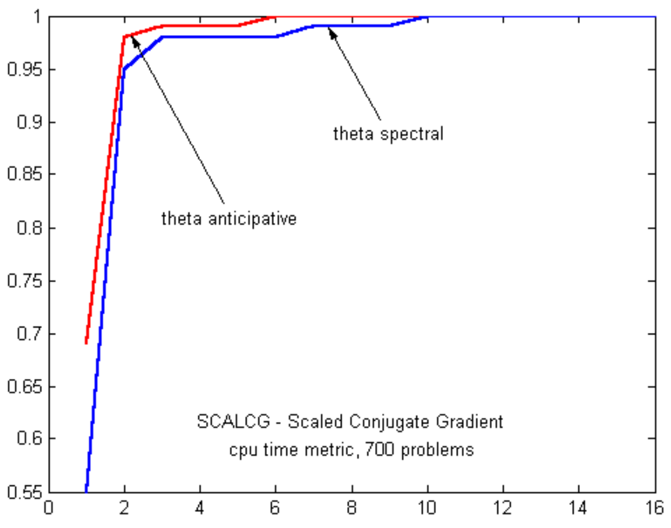

Fig. 2. CPU time performance profile of SCALCG.
θ^s versus θ^a.

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 almost all values of τ. Therefore, we see that the anticiaptive selection of the scaling parameter compares favourable, being slightly better then spectral approach.

As we know, different stopping criteria change the performances of the algorithms, as well as the accuracy of the solution. For example, considering the stopping criteria (35a)-(35c), then the performance of SCALCG (θ^a), for solving this set of 700 test problems, is given as in Table 3.

Table 3. Global characteristics of SCALCG (θ^a) subject to different stopping criteria. 700 problems.

Stopping criteria	# iter	# fg	cpu (s)
35a	628485	837906	2830.91
35b	637296	852515	3062.24
35c	819160	1102347	4225.28

5.2. SCG with θ^s versus SCG with θ^a

The direction in SCG code by Birgin and Martinez [5] is computed as in (6), where θ_{k+1} is determined in a spectral (28), or in an anticipative (33), manner. 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 when (35a) is satisfied. The original stopping criterion used in SCG was: $\|\nabla f(x_k)\|_2 \le \varepsilon_g max\{1, |f(x_{k+1}|\}$. However, except 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 stop the iterations almost immediately, far from the optimum. Therefore, we changed the stopping criteria as in (35a). 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 $\alpha_{k-1}\|d_{k-1}\|_2 / \|d_k\|_2$. Tables 4 and 5 show the global characteristics corresponding to these 700 test problems, where the iterations are restarted using the Powell (20) or angle (21) restart criterion, respectively.

Table 4. Global characteristics of SCG with θ^s versus SCG with θ^a. Powell restart. 700 problems.

Global characteristics	θ^s	θ^a
Total number of iterations	555874	543574
Total number of function evaluations	894892	1131209
Total cpu time (seconds)	4948.08	6750.37

Table 5. Global characteristics of SCG with θ^s versus SCG with θ^a. Angle restart. 700 problems.

Global characteristics	θ^s	θ^a
Total number of iterations	953032	954041
Total number of function evaluations	1266262	1278089
Total cpu time (seconds)	7801.14	7868.20

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 700 problems solved by SCG with θ^a in this numerical experiment, only for 178 (i.e. 25.42%) γ_{k+1} in (30) was negative in case of Powell restart, and for only 177 (i.e. 25.2% γ_{k+1} in (30) was negative in case of angle restart. From Tables 4 and 5 we see that SCG with Powell restart criterion, in both implementations using θ^s or θ^a, is slightly better than SCG with angle criterion. For example, for solving this set of 700 problems SCG with Powell restart and θ^s is with 2853 seconds fastest than SCG with angle restart and θ^s.

5.3. CG_DESCENT with Wolfe line search (W) versus CG_DESCENT with approximate Wolfe line search (aW)

Recently, for solving (1), Hager and Zhang [12] presented a *new conjugate gradient algorithm with guaranteed descent* and the performance of the Fortran 77 package CG_DESCENT which implements it. The directions d_k are computed by the following rule:

$$d_{k+1} = -g_{k+1} + \beta_k d_k, \tag{51}$$

$$\beta_k = \frac{1}{d_k^T y_k}\left(y_k - 2\frac{y_k^T y_k}{d_k^T y_k} d_k \right)^T g_{k+1}, \tag{52}$$

$d_0 = -g_0$. We see that the above direction d_{k+1} of Hager and Zhang, given in (51)-(52), denoted d_{k+1}^{HZ}, can be expressed as:

$$d_{k+1}^{HZ} = -g_{k+1} - \frac{y_k^T y_k}{s_k^T y_k}\left[2\frac{s_k^T g_{k+1}}{s_k^T y_k} - \frac{y_k^T g_{k+1}}{y_k^T y_k} \right] s_k. \tag{53}$$

On the other hand, the direction generated by Shanno, let call it d_{k+1}^S, is given as (see (2) in [26]):

$$d_{k+1}^S = -\frac{s_k^T y_k}{y_k^T y_k} g_{k+1} - \left[2\frac{s_k^T g_{k+1}}{s_k^T y_k} - \frac{y_k^T g_{k+1}}{y_k^T y_k} \right] s_k + \frac{s_k^T g_{k+1}}{y_k^T y_k} y_k. \tag{54}$$

After some simple algebra we see that the direction d_{k+1}^{HZ} of Hager and Zhang is related to the direction d_{k+1}^{S} of Shanno in the following way:

$$d_{k+1}^{S} = \frac{s_k^T y_k}{y_k^T y_k}\left(d_{k+1}^{HZ} + \frac{s_k^T g_{k+1}}{s_k^T y_k} y_k \right) \quad \text{or} \quad d_{k+1}^{HZ} = \frac{y_k^T y_k}{s_k^T y_k} d_{k+1}^{S} - \frac{s_k^T g_{k+1}}{s_k^T y_k} y_k .$$

With other words, the direction d_{k+1}^{HZ} is obtained from d_{k+1}^{S} by deleting from this direction the term

$$\frac{s_k^T g_{k+1}}{y_k^T y_k} y_k . \tag{55}$$

In their algorithm Hager and Zhang restrict β_k to be nonnegative. This is motivated by the work of Gilbert and Nocedal [10] who modified the Polak and Ribière updating formula as $\beta_k^+ = max\{\beta_k, 0\}$ and proved the global convergence of this computational scheme for general nonlinear functions. Similar to the approaches considered by Gilbert and Nocedal [10], Han, Liu, Sun and Yin [13], and Wang, Han and Wang [28] in their studies on the Polak-Ribière version of the conjugate gradient method, Hager and Zhang prove the convergence for general nonlinear functions by restricting the lower bound of β_k in the following manner:

$$d_{k+1} = -g_{k+1} + \bar{\beta}_k d_k , \tag{56}$$

$$\bar{\beta}_k = \max\{\beta_k, \eta_k\}, \tag{57}$$

$$\eta_k = \frac{-1}{\|d_k\| min\{\eta, \|g_k\|\}}, \tag{58}$$

where β_k is given by (52), and the parameter $\eta > 0$ is a user specified constant. (Suggested value: $\eta = 0.01$, considered in all numerical experiments).

An important innovation given by Hager and Zhang in their approach consists of a new efficient and highly accurate line search procedure. This is based on the Wolfe conditions (17)-(18) and on a very fine interpretation of the numerical issue concerning the first Wolfe condition (17). In CG_DESCENT the iterations are stopped whenever $\|g_k\|_\infty \le \varepsilon_g$, where $\varepsilon_g = 10^{-6}$. The Wolfe line search conditions are implemented in CG_DESCENT with: $\sigma_1 = 0.1$ and $\sigma_2 = 0.9$. These values represent a compromise between the desire for a rapid termination of 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. Table 6 shows the global characteristics of CG_DESCENT with Wolfe line search (W), as well as of CG_DESCENT with approximate Wolfe line search (aW), corresponding to these 700 test problems.

Table 6. Global characteristics of CG_DESCENT.
700 problems.

Global characteristics	W	aW
Total number of iterations	460920	484429
Total number of function evaluations	941037	963294
Total cpu time (seconds)	5006.28	5967.41

In Figures 3 and 4 we compare the performance profiles of CG_DESCENT with Wolfe line search and CG_DESCENT with approximate Wolfe line search refering to the number of function evaluations and cpu time, respectively.

The best performance, relative to the function evaluations metric or cpu time metric, was obtained by CG_DESCENT with approximate Wolfe line search, the top curve in Figures 3 and 4. In CG_DESCENT the search directions are always descent directions, independent of the accuracy in the line search [12]. Since both codes generate the same direction, it follows that the CG_DESCENT with approximate Wolfe line search is more performant, subject to these metrics, because it generates a more accurate step length.

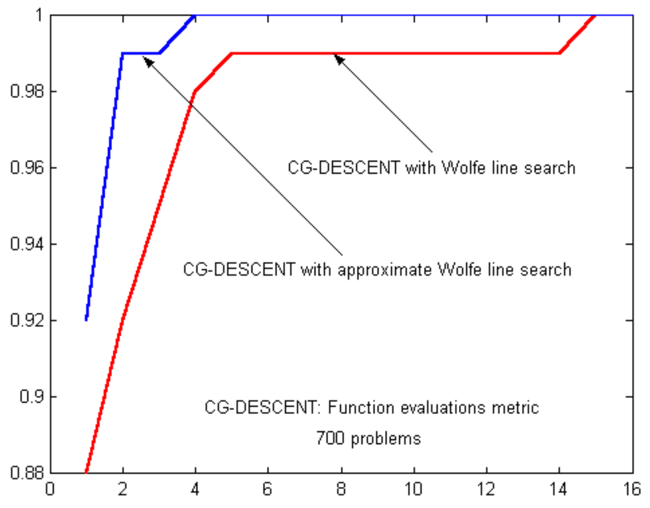

Fig. 3. CG_DESCENT: Performance based on number of function evaluations.

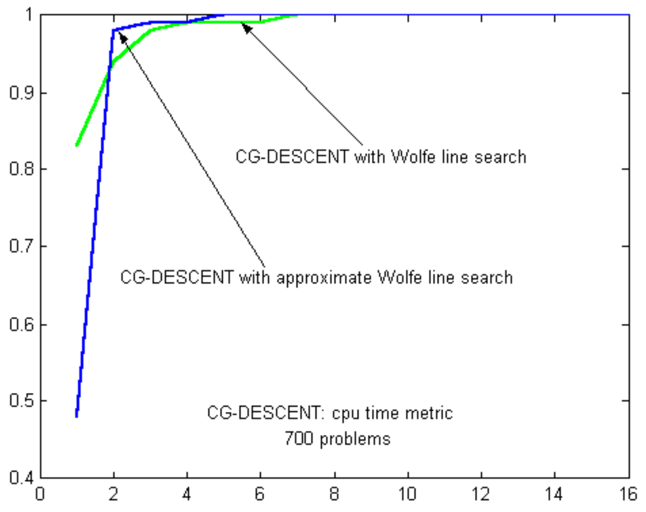

Fig. 4. CG_DESCENT: Performance based on CPU time.

Observe, in Figure 3, that the CG_DESCENT with approximate Wolfe line search code is the top performer, relative to the function evaluation metric for almost all values of τ . From Figure 4 we note that relative to the cpu time metric, both codes have similar performances for all values of τ , CG_DESCENT with approximate Wolfe line search being slightly better. Therefore, the overall poor performance of CG_DESCENT with Wolfe line search, both in the function evaluation and cpu time, is connected with the performance of the line search. Hager and Zhang [11,12] give computational evidence that their CG_DESCENT is better than L-BFGS - Limited Memory quasi-Newton method by Nocedal [16] and Liu and Nocedal [15], and than PRP+ version of the conjugate gradient method developed by Gilbert and Nocedal [10].

5.4. Scaled Polak-Ribière and scaled Fletcher-Reeves

The search direction in scaled Polak-Ribière or scaled Fletcher-Reeves algorithms is computed as in (7) or (8) respectively, where the scaling parameter θ_k is selected in a spectral or an anticipative manner. The iterations are stopped whenever (35a) is satisfied. 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 $\alpha_{k-1}\|d_{k-1}\|_2/\|d_k\|_2$. Tables 7 and 8 show the global characteristics corresponding to these 700 test problems, where both variants of the algorithms implement the Powell restart criterion.

Table 7. Global characteristics of Polak-Ribière.
700 problems. Powell restart.

Global characteristics	θ^s	θ^a
Total number of iterations	516246	505004
Total number of function evaluations	814305	863769
Total cpu time (seconds)	4662.16	4918.43

Table 8. Global characteristics of Fletcher-Reeves.
700 problems. Powell restart

Global characteristics	θ^s	θ^a
Total number of iterations	521889	500817
Total number of function evaluations	847201	823806
Total cpu time (seconds)	4497.10	4359.46

We see that while there are differences, both these variants of Polak-Ribière and Fletcher-Reeves algorithms have similar performances, Fletcher-Reeves with θ^a being slightly more competitive, at least for this set of test problems considered in this study.

5.5. Comparison: SCALCG versus SCG and CG_DESCENT

Motivated by the results given in the above Tables and Figures in the following we compare SCALCG (θ^a) with SCG (θ^s) and CG_DESCENT (aW). Both SCALCG and SCG restart iterations using the Powell criterion. Table 9 shows the global characteristics, corresponding to these 700 test problems, refering to the total number of iterations, total number of function evaluations and total cpu time for these algorithms.

Table 9. Global characteristics of SCALCG (θ^a), SCG (θ^s) and CG_DESCENT (aW).
Powell restart. 700 problems.

Global characteristics	SCALCG	SCG	CG_DESCENT
Total number of iterations	628485	555874	484429
Total number of function evaluations	837906	894892	963294
Total cpu time (seconds)	2830.91	4948.08	5967.41

Table 10 illustrates the number of problems, out of 700, for which the above algorithms achieved the minimum number of iterations, minimum number of function evaluations and minimum cpu time, respectively.

Table 10. Performance of SCALCG (θ^a), SCG (θ^s) and CG_DESCENT (aW). Powell restart. 700 problems.

Performance criterion	# of problems
SCALCG (θ^a) achieved minimum # of iterations in	451
SCG (θ^s) achieved minimum # of iterations in	105
CG_DESCENT (aW) achieved minimum # of iterations in	198
SCALCG (θ^a) achieved minimum # of function evaluations in	419
SCG (θ^s)achieved minimum # of function evaluations in	201
CG_DESCENT (aW) achieved minimum # of function evaluations in	215
SCALCG (θ^a) achieved minimum cpu time in	583
SCG (θ^s) achieved minimum cpu time in	62
CG_DESCENT (aW) achieved minimum cpu time in	117

In Figures 5 and 6 we compare the performance profiles of all these three algorithms subject to function evaluations and cpu time metrics, respectively.

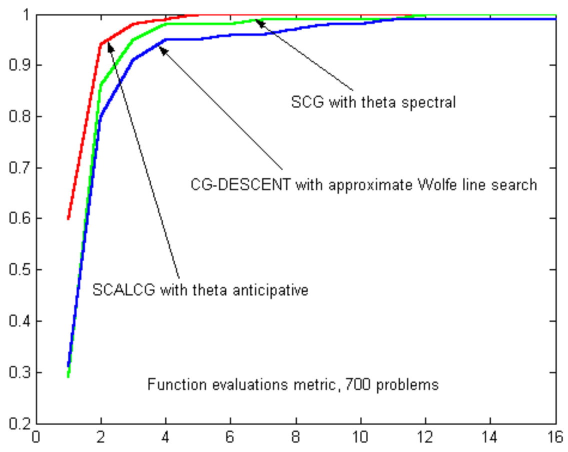

Fig. 5. Performance based on number of function evaluations. SCALCG (θ^a) versus SCG (θ^s) and CG_DESCENT (aW).

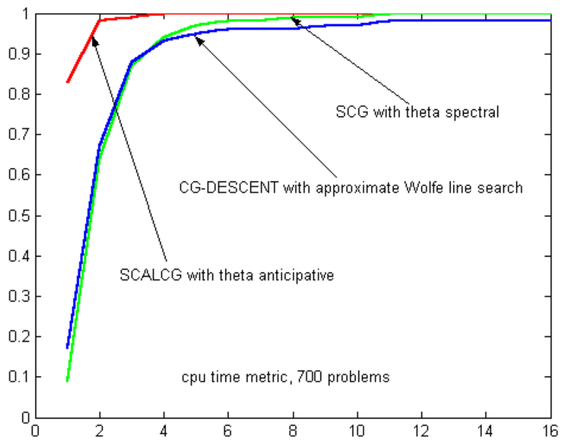

Fig. 6. Performance based on cpu time. SCALCG (θ^a) versus SCG (θ^s) and CG_DESCENT (aW).

Notice that relative to the function evaluations and cpu time metrics from Figures 5 and 6 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 SCG(θ^s) and followed by CG_DESCENT (aW).

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 10 we see that refering to the number problems solved in minimum number of iterations SCALCG is about 4.3 times more performant than SCG. Considering the function evaluations it is about twice more performant. Refering to the cpu time we see that SCALCG is about 9.4 times fastest than SCG.

In the function evaluation metric, from Table 10, we see that, at least for this set of 700 problems, with dimensions ranging from 10^3 to 10^4, SCALCG is about twice more performant than CG_DESCENT. In the cpu time metric SCALCG is about 5 times more performant than CG_DESCENT, i.e. the fraction of problems for which SCALCG achieved the best time is about 5 times greater than that corresponding to CG_DESCENT. In Figures 5 and 6 these performances are illustrated for $\tau = 1$. The salient point computationaly 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 700 problems, from Table 9 we see that SCALCG is almost twice fastest 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 satisfy the quasi-Newton equation in a restart environment. Although the update formulas (15) and (25)-(27) are more complicated than the computational scheme (52) and (56)-(58), 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 set.

5.6. Comparisons: SCALCG versus SCG, CG_DESCENT and CONMIN

In the following we compare SCALCG (θ^a) with SCG (θ^s), CG_DESCENT (aW) and CONMIN by Shanno and Phua. Table 11 shows the general characteristics, corresponding to these 700 test problems, refering to the total number of iterations, the total number of function evaluations and the total cpu time for these algorithms.

Table 11. Global characteristics of SCALCG (θ^a), SCG (θ^s), CG_DESCENT (aW) and CONMIN. 700 problems.

	Global characteristics		
Algorithm	**# iterations**	**# function evals**	**cpu time (seconds)**
SCALCG (θ^a)	628485	837906	2830.91
SCG (θ^s)	555874	894892	4948.08
CG_DESCENT (aW)	484429	963294	5967.41
CONMIN	434349	1509312	9127.65

In Figures 7 and 8 we compare the performance profiles of these algorithms subject to the number of function evaluations and cpu time, respectively.

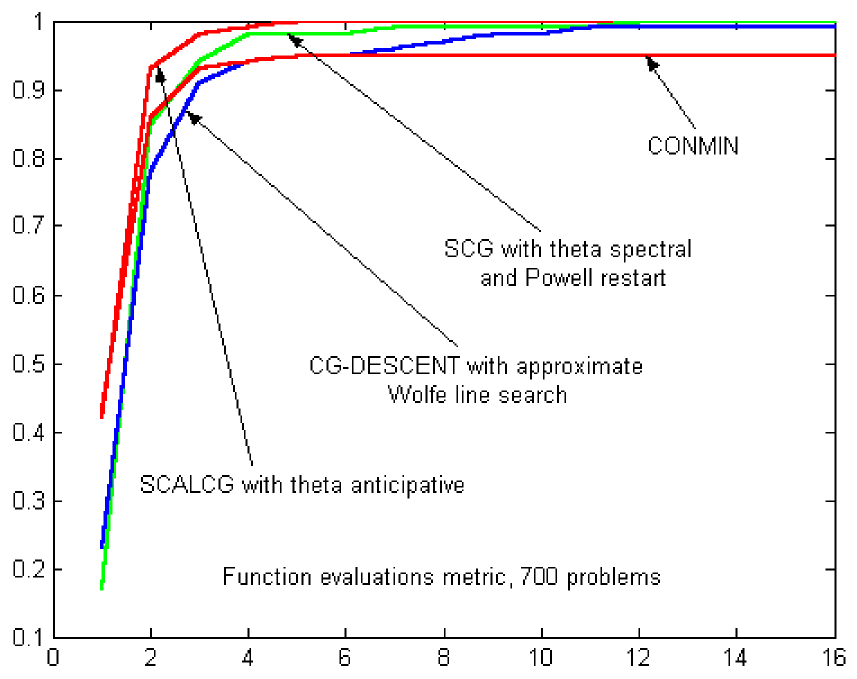

Fig. 7. Performance based on number of function evaluations.

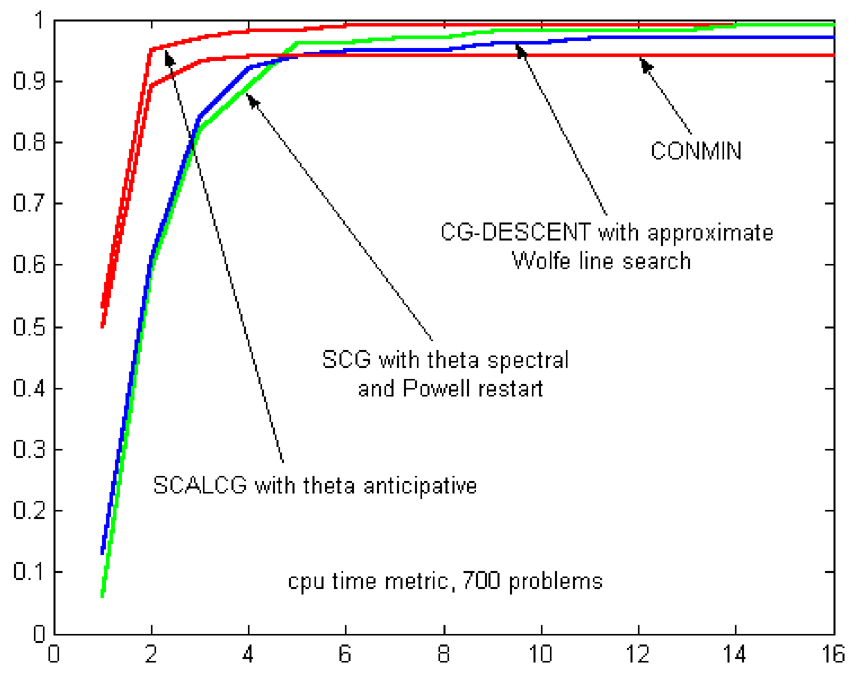

Fig. 8. Performance based on cpu time.

The top curve correspond to the algorithm that solved the most problems in a number of function evaluations (Figure 7) or in a cpu time (Figure 8) that was within a given factor of the best number function evaluations and cpu time, respectively. Since the top curve in

Figures 7 and 8 corresponds to SCALCG, this algorithm is clearly the best among these algorithms considered in this study.

Relative to the function evaluations and cpu time metrics, from Figures 7 and 8, we see that for $\tau = 1$ SCALCG and CONMIN have similar performances, however SCALCG (θ^a) is top performer for almost all values of τ. Concerning the robustness, we see that SCALCG (θ^a) is more robust, followed by SCG (θ^s), CG_DESCENT (aW) and CONMIN, in this order. Since all these algorithms use the same Wolfe line search, the same restart criterion (Powell), and the same stopping criteria, mainly these algorithms differ in their procedures for search direction computation. Therefore, Figures 7 and 8 give a computational evidence that SCALCG generates a better direction.

The comparison between SCALCG and CONMIN, subject to function evaluations and cpu time metrics, reveals that SCALCG is more performant than CONMIN. An explanation of this behaviour seems to be as follows. As we know Oren [17], Oren and Luenberger [18] and Oren and Spedicato [19] modified the Broyden class of quasi-Newton methods by introducing a scalar parameter in order to make the sequence of inverse Hessian invariant under multiplication of function f by a scalar constant. For this scaling parameter Shanno [25] suggests the value $s_k^T y_k / y_k^T y_k$ as that value minimizing the condition number of $H_k^{-1} H_{k+1}$. This scaling factor is used in CONMIN. On the other hand, in SCALCG we use another value for scaling parameter θ_{k+1}, as a scalar approximation of the inverse Hessian given by (28) or (33) yealding to a more efficient direction. This factor greatly increase both the computational stability and efficiency as the problem size increase, explaining the numerical behaviour of SCALCG in comparison with CONMIN subject to function evaluations and cpu time metrics.

5.7. Comparisons: SCALCG versus SCG, CG_DESCENT, CONMIN and Polak-Ribière

Figures 9 and 10 show the performance profiles of SCALCG (θ^a), SCG (θ^s), CG_DESCENT (aW), CONMIN and spectral Polak-Ribière, subject to number of function evaluations and cpu time metrics, respectively. Again we notice that, at least for this set of 700 test unconstrained optimization problems, SCALCG with θ^a is the top performer both subject to function evaluations and cpu time metrics. We see that, for $\tau = 1$, the algorithms clasifies in two groups with similar performances. One group include SCALCG and CONMIN, and the second one contains SCG, CG_DESCENT and Polak-Ribière. This separation is more evident for cpu time metric. As we have already said, these algorithms differ in their procedures for search direction computation. Compared to SCG, CG_DESCENT and Polak-Ribière, the direction in SCALCG and CONMIN is more elaborated. It satisfy the quasi-Newton equation using a double update scheme in a restart environment given by Powell procedure.

Computationally, the salient point is that the scaled Polak-Ribière algorithm in this implementation has the best performance profile, compared to SCG and CG_DESCENT. Concerning the robustness, we see that scaled Polak-Ribière classifies the second, immediately after SCALCG. Scaled Polak-Ribière algorithm possess a built-in restart feature that addresses the jamming problem. When $s_k = x_{k+1} - x_k$ is small, the factor y_k in the numerator of (7) tends to zero. Hence, the new search direction d_{k+1} given by (7) is essentially the scaled steepest descent direction $-\theta_{k+1} g_{k+1}$, where the scaling factor θ_{k+1} is a scalar approximation to the inverse Hessian. Generally, the scaled Polak-Ribière method automatically adjust the parameter β_k to avoid jamming. On the other hand, SCG does not have this property. Even that y_k in the numerator of (6) tends to zero, the new search direction given by (6) is different by the scaled steepest descent direction $-\theta_{k+1} g_{k+1}$. Therefore, jamming can appear. However, in the numerator of (6) apperas the factor

$\theta_{k+1} y_k - s_k$ which is resemblig the quasi-Newton equation. Figures 9 and 10 give computational evidence that SCG is more robust than CG_DESCENT. CG_DESCENT was designed in order to ensure sufficient descent, independent of the accuracy of the line search. When d_k is small, from (52) we see that the new search direction essentially is the Hestens and Stiefel direction which avoids jamming.

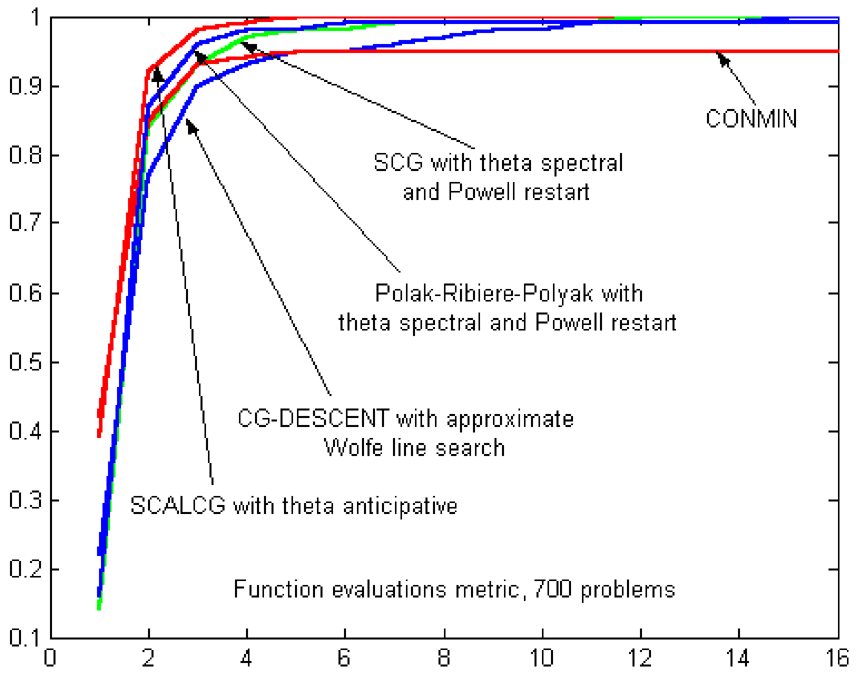

Fig. 9. Performance based on number of function evaluations.

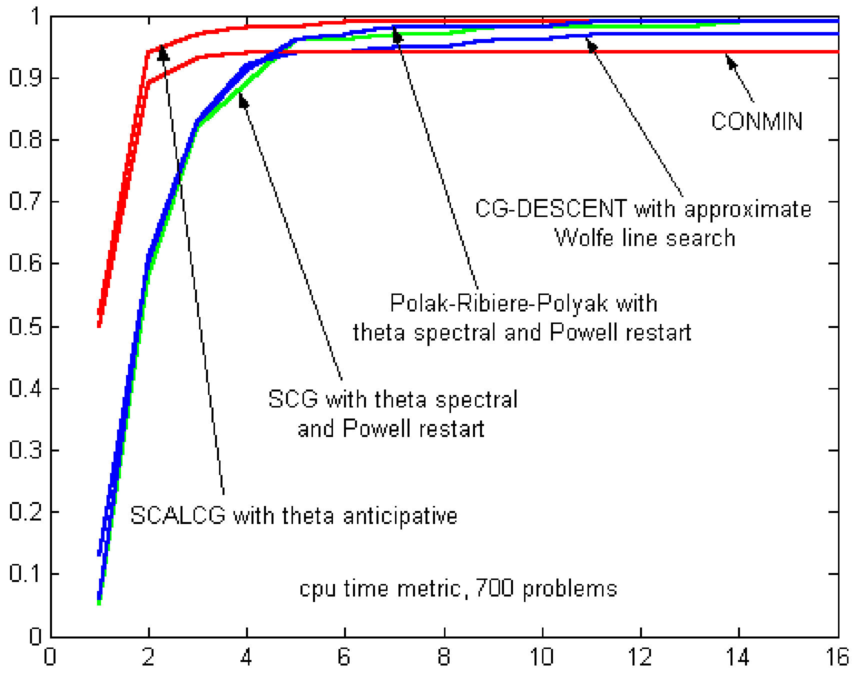

Fig. 10. Performance based on cpu time.

5.8. Comparisons: SCALCG versus SCG, scaled Polak-Ribière-Polyak, scaled Fletcher-Reeves and CONMIN on MINPACK-2 applications

In Tables 12-17 we present the performance of SCALCG, SCG, scaled Polak-Ribière-Polyak (sPRP) and scaled Fletcher-Reeves (sFR) for 6 unconstrained optimization applications from MINPACK-2 collection [2]. For comparison, in Table 18 we present the performances of CONMIN on same problems. In all these numerical experiments we have considered the standard initial point, as it is recommended in MINPACK-2 and the iterations was stopped according to (35a) criteria.

Table 12. Performance of SCALCG, SCG, sPRP and sFR: **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
sPRP	265	235	420	375	24.06	21.42	-0.439162989
sFR	289	245	452	385	25.92	22.03	-0.439163155
$nx = 200, ny = 200$, $c = 5.$, $n = 40000$.							
SCALCG	398	473	511	614	87.39	104.80	-0.439267742
SCG	472	425	746	668	170.92	153.18	-0.439265832
sPRP	516	391	828	621	188.83	141.76	-0.439264713
sFR	442	445	697	698	159.12	159.56	-0.439267086

Table 13. Performance of SCALCG, SCG, sPRP and sFR: **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
sPRP	523	454	838	705	45.86	38.94	-0.282840004
sFR	495	569	777	889	42.73	49.05	-0.282839956
$nx = 200, ny = 200$, $ecc = 0.1$, $b = 10$, $n = 40000$.							
SCALCG	876	900	1143	1157	189.11	191.14	-0.282892918
SCG	1117	1133	1746	1779	384.21	390.96	-0.282892622
sPRP	980	961	1556	1530	339.82	333.56	-0.282892453
sFR	874	1069	1399	1687	303.30	368.44	-0.282892680

Table 14. Performance of SCALCG, SCG, sPRP and sFR: **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
sPRP	1319	1439	2064	2245	149.62	162.80	-0.011377226
sFR	1184	1420	1854	2221	134.02	160.50	-0.011377160

$nx = 200, ny = 200, \lambda = 0.008,\ n = 40000$.							
SCALCG	1413	939	1753	1157	431.82	285.06	-0.011381240
SCG	3952	3184	6003	4828	1749.27	1407.30	-0.011380973
sPRP	3628	4372	5680	6847	1644.47	1979.79	-0.011381028
sFR	4398	3945	6788	6111	1964.08	1765.58	-0.011381183

Table 15. Performance of SCALCG, SCG, sPRP and sFR:
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
sPRP	10001	10001	15948	15922	234.81	237.72	0.646897e-04
sFR	10001	10001	15881	15869	235.19	231.07	0.103695e-04

Table 16. Performance of SCALCG, SCG, sPRP and sFR:
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
sPRP	2553	2917	4058	4591	637.91	721.99	-6621.582203
sFR	2327	3142	3669	5802	576.50	907.58	-6612.839493

Table 17. Performance of SCALCG, SCG, sPRP and sFR:
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
sPRP	388	360	611	570	58.60	54.60	-0.070086368
sFR	240	278	380	439	36.25	41.96	-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
sPRP	664	555	1061	875	404.47	334.17	-0.070085849
sFR	517	538	821	843	312.53	321.70	-0.070085770

Table 18. Performance of CONMIN on MINPACK-2 applications.

Problem	n	# iter	# fg	cpu (s)	$f(x^*)$
Elastic-Plastic Torsion	10000	217	439	17.90	-0.43916320
	40000	242	486	78.76	-0.43926781
Pressure Distribution in a Journal Bearing	10000	396	801	32.74	-0.2828400078
	40000	819	1657	270.23	-0.282892943
Optimal Design with Composite Materials	10000	359	729	40.76	-0.011377240
	40000	675	1370	305.28	-0.011381291
Inhomogeneous Superconductors	1000	9881	20001	270.12	0.11079927e-03
Leonard-Jones Cluster	3000	880	1819	507.07	-6605.45868939
Steady State Combustion Solid fuel ignition	10000	171	346	28.40	-0.0700863684
	40000	467	948	310.60	-0.0700863743

From Tables 12-18 we see that for solving these 10 MINPACK-2 applications, the top performer is SCALCG (θ^a) followed immediatelly by CONMIN. Even that SCALCG and CONMIN have a lot of algebra in common, we see that SCALCG is more performant than CONMIN. The total number of function evaluations of SCALCG (θ^a) is 14749 compared to 28596 required by CONMIN for solving these 10 applications. At the same time we see that SCALCG is with 479.53 seconds faster than CONMIN. This is in accordance with performance profiles from Figures 9 and 10, for $\tau = 1$.

5.9. Accuracy comparisons

In the next series of experiments we explore the ability of the algorithms to *accurately* solve the problems. Since CG_DESCENT implements a new line search procedure based on approximate Wolfe conditions, we compare SCALCG and CG_DESCENT, and consider the following two problems:

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

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

with the initial point $x_0 = [1,\ldots,1]$. Tables 19a,b and 20a,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 19a. 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 19b. Iteration and solution time versus tolerance ε_g ,
CG_DESCENT, $f_1(x)$, $n = 10000.$

ε_g	W			aW		
	#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 20a. 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 20b. 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 was selected since they illustrate the typical behaviour that we 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 $\varepsilon_f \left| f(x_{k+1}) \right|$ for the error in function value used in the second stopping criterion in (35a) 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 way 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 [12].
- 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 of course 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.

6. CONCLUSION

We have presented a new conjugate gradient algorithm, *SCALCG - scaled conjugate gradient algorithm*, for solving large-scale unconstrained optimization problems. In a way SCALCG can be considered as a modification of the best algorithm by Birgin and Martinez [5], which mainly is a scaled variant of Perry's [20], 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 (15) and (25)-(27) are more complicated the scheme proved to be efficient and robust in numerical experiments. The algorithm implements the Wolfe conditions, and we prove that the steps are along the descent directions.

The performance profiles for our scaled conjugate gradient algorithm was higher than those of *SCG - spectral conjugate gradient method* by Birgin and Martinez [5], *CG_DESCENT - conjugate gradient with guaranteed descent* by Hager and Zhang [12], *scaled Polak-Ribière* and *Fletcher-Reeves*, and *CONMIN* by Shanno and Phua [27], for a test set consisting of 700 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, the number of problems for which SCALCG achieved the best number is about twice greater than that corresponding to CG_DESCENT. Similarly, the fraction of problems for which SCALCG achieved the best time is about 5 times greater than that corresponding to CG_DESCENT. The better performance of SCALCG, relative to SCG, CG_DESCENT and scaled Polak-Ribière-Polyak, 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 satisfy 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.

SCALCG and CONMIN belong to the same class of conjugate gradient algorithms. Both of them consider a BFGS modification of the Perry updating scheme which satisfy the quasy-Newton equation in a Beale-Powell restart environment. Mainly, the difference is in the scaling of gradient parameter. For this parameter SCALCG considers a scalar approximation of inverse Hessian, wich prove to be efficient and robust in numerical calculations. On the other hand, CONMIN uses a value minimizing the condition number of $H_k^{-1}H_{k+1}$.

It is important to notice that different stopping criteria have a dramatic influence on the performances of the algorithms, as well as on the accuracy of the solution. To get relevant conclusions, in this numerical experiment we tried to unify the stopping criteria in these codes. The criteria (35a) used in this numerical study seems to be the best in case of solving large-scale problems. The second criterion in (35a), which considers the change in function value, is important. In these experiments SCALCG terminates the iterations due to it for about 10-12% of problems.

Finally, since each of these codes, we have considered here in this numerical study, are different in many respects, mainly in the amount of linear algebra required in each iteration, in 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 set. However, we have a strong computational evidence that our SCALCG algorithm is one of the top performers among conjugate gradient algorithms.

REFERENCES

1. ANDREI, N. "*A new gradient descent method for unconstrained optimization*", ICI Technical Report, March 2004.
2. AVERICK, B.M., CARTER, R.G., MORÉ, J.J. and XUE, G.L. "*The MINPACK-2 test problem collection*", Argonne National Laboratory, Preprint MCS-P153-0692, June 1992.
3. BARZILAI, J. and BORWEIN, J.M. "*Two point step size gradient method*", IMA J. Numer. Anal., 8, pp.141-148, 1988.
4. BEALE, E.M.L. "*A derivation of conjugate gradients*" in: F.A. Lootsma, ed., *Numerical Methods for Nonlinear Optimization*, Academic Press, London, pp.39-43, 1972.
5. BIRGIN, E. and MARTINEZ, J.M. "*A spectral conjugate gradient method for unconstrained optimization*", Applied Math. and Optimization, 43, pp.117-128, 2001
6. BONGARTZ, I., CONN, A.R., GOULD, N.I.M., and TOINT, P.L. "*CUTE: constrained and unconstrained testing environments*", ACM Trans. Math. Software, 21, pp.123-160, 1995.
7. DAI, Y.H., and LIAO, L.Z. "*New conjugate conditions and related nonlinear conjugate gradient methods*", Appl. Math. Optim., vol. 43 pp.87-101, 2001.
8. DOLAN, E.D., and MORÉ, J.J. "*Benchmarking optimization software with performance profiles*", Math. Programming, 91, pp. 201-213, 2002.
9. FLETCHER, R., and REEVES, C.M. "*Function minimization by conjugate gradients*", Comput. J. 7, pp. 149-154, 1964.
10. GILBERT, J.C., and NOCEDAL, J. "*Global convergence properties of conjugate gradient methods for optimization*", SIAM J. Optim., 2, pp.21-42, 1992.
11. HAGER, W.W., and ZHANG, H. "*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. HAGER, W.W., and ZHANG, H. "*CG-DESCENT, A conjugate gradient method with guaranteed descent (algorithm details and comparisons)*", University of Florida, Department of Mathematics, January 15, 2004.
13. HAN, J.Y., LIU, G.H., SUN D.F., and YIN, H.X. "*On the global convergence of nonlinear conjugate gradient methods*", Technical Report 94-011, Inst. of Applied Math., Chinese Academy of Science, Beijing, 1994.
14. HESTENS, M.R., and STIEFEL, E. "*Methods of conjugate gradients for solving linear systems*", J. Research Nat. Bur. Standards Sec. B. 48, pp. 409-436, 1952.
15. LIU, D.C., and NOCEDAL, J. *"On the limited memory BFGS method for large scale*

optimization", Math. Programming, 45, pp.503-528, 1989.

16. NOCEDAL, J. "*Updating quasi-Newton matrices with limited storage*", Math. Comp., 35, pp.773-782, 1980.
17. OREN, S.S. "*Self-scaling variable metric algorithm. Part II*", Management Sci., 20, pp.863-874, 1974.
18. OREN S.S., and LUENBERGER, D.G. "*Self-scaling variable metric algorithm. Part I*", Management Sci., 20, pp.845-862, 1976.
19. OREN, S.S., and SPEDICATO, E. "*Optimal conditioning of self-scaling variable metric algorithms*", Math. Programming, 10, pp.70-90, 1976.
20. PERRY, J.M. "*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.
21. POLAK, E., and RIBIÈRE, G. "*Note sur la convergence de methods de directions conjugres*", Revue Francaise Informat. Reserche Opérationnelle 16, pp. 35-43, 1969.
22. POLYAK, B.T. "*The conjugate gradient method in extreme problems*", USSR Comp. Math. Math. Phys., 9, pp.94-112, 1969.
23. POWELL, M.J.D. "*Restart procedures for the conjugate gradient method*", Math. Programming, 12, pp.241-254, 1977.
24. RAYDAN, M. "*The Barzilai and Borwein gradient method for the large scale unconstrained minimization problem*", SIAM J. Optim., 7, 26-33, 1997.
25. SHANNO, D.F. "*Conjugate gradient methods with inexact searches*", Mathematics of Operations Research, vol. 3, pp.244-256, 1978.
26. SHANNO, D.F. "*On the convergence of a new conjugate gradient algorithm*", SIAM J. Numer. Anal. vol. 15, pp.1247-1257, 1978.
27. SHANNO, D.F., and PHUA, K.H. "*Algorithm 500, Minimization of unconstrained multivariate functions [E4]*", ACM Trans. on Math. Soft., 2, pp.87-94, 1976.
28. WANG, C., HAN, J., and WANG, L. "*Global convergence of the Polak-Ribière and Hestens-Stiefel conjugate gradient methods for the unconstrained nonlinear optimization*", OR Transactions, 4, pp.1-7, 2000.
29. WOLFE, P. "*Convergence conditions for ascent methods*", SIAM Rev., 11, pp.226-235, 1969.
30. WOLFE, P. "*Convergence conditions for ascent methods II: some corrections*", SIAM Rev. 13, pp.185-188, 1971.

June 28, 2005

-----oooooOooooo-----