Efficiency and robustness of conjugate gradient methods subject to the procedures for stepsize computation

Neculai Andrei
Center for Advanced Modeling and Optimization,
Academy of Romanian Scientists,
54, Splaiul Independenţei, Sector 5,
Bucharest, ROMANIA
E-mail: neculaiandrei70@gmail.com

Technical Report 2/2021
January 11, 2021

Abstract. It is known that the conjugate gradient methods are very sensitive to the stepsize computation. This technical report presents some numerical results of HS (Hestenes-Stiefel), PRP+ (Polak-Ribière-Polyak) and DY (Dai-Yuan) conjugate gradient methods implemented with two procedures for stepsize computation. The first one implements the standard Wolfe line search with cubic interpolation. The second one implements the standard Wolfe line search with a simple bisection method. Intensive numerical experiments with 800 unconstrained optimization problems, with the number of variables in the range [1000, 10000], show that the performances of conjugate gradient methods are very dependent by the procedure for stepsize computation. Particularly, all these conjugate gradient methods equipped with cubic Wolfe line search is way more efficient and more robust.

1. Introduction

For solving the problem

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

where $f:\mathbb{R}^n \to \mathbb{R}$ is a continuously differentiable function lower bounded and $x \in \mathbb{R}^n$, the conjugate gradient methods are defined by the following algorithm:

$$x_{k+1} = x_k + \alpha_k d_k, \quad k = 0,1,\ldots, \tag{2}$$

where d_k is the search direction computed as

$$d_{k+1} = -g_k + \beta_k d_k, \quad k = 0,1,\ldots, \tag{3}$$

with $d_0 = -g_0$. In (3) β_k is a scalar known as the conjugate gradient parameter defined by different formulae involving some elements of the algorithm and $g_k = \nabla f(x_k)$. The search direction d_k, assumed to be a descent one, plays the main role in these methods. On the other hand, the stepsize α_k guarantees the global convergence in some cases and is crucial in

efficiency. For nonlinear functions different parameters β_k determine conjugate gradient algorithms with different performances. Details on conjugate gradient methods including their definition and properties, as well as their performances for solving a large class of unconstrained optimization problems with different structures and complexities are presented in Andrei (2020).

It is known that in conjugate gradient algorithms, the search directions tend to be poorly scaled and consequently the line search must perform more function evaluations in order to obtain a suitable stepsize α_k. In conjugate gradient methods, the stepsizes differ from 1 in a very unpredictable way. They can be larger or smaller than 1, depending on how the problem is scaled. This is in very sharp contrast to the Newton and the quasi-Newton methods, including the limited memory quasi-Newton methods, which accept the unit stepsize most of the time along the iterations and therefore they usually require only few function evaluations per search direction. In this technical report we are interested to see the performances of the conjugate gradient methods subject to different procedures for stepsize α_k computation. In the following we consider two procedures for computing the stepsize α_k. Both of them implement the standard Wolfe line search:

$$f(x_k + \alpha_k d_k) - f(x_k) \le \rho \alpha_k g_k^T d_k, \tag{4}$$

$$g_{k+1}^T d_k \ge \sigma g_k^T d_k, \tag{5}$$

where $0 < \rho < 1/2 \le \sigma < 1$. The first Wolfe condition (4) is called the *sufficient reduction* of the minimizing function values, which ensures a sufficient reduction of function values. The second Wolfe condition (5) is the *curvature condition*, which ensures unacceptable short stepsizes. The first procedure for computing the stepsize α_k implements the Wolfe line search conditions (4) and (5) with cubic interpolation. The second procedure implements the Wolfe line search conditions (4) and (5) with a simple bisection method. The following theorem proves that there exists a value α_k satisfying both Wolfe line search conditions (4) and (5).

Theorem 1. *Suppose that the function f is continuously differentiable. Let d_k be a descent direction at point x_k and assume that f is bounded from below along the ray $\{x_k + \alpha d_k : \alpha > 0\}$. Then, if $0 < \rho < \sigma < 1$, there exists an interval of stepsizes α satisfying the Wolfe conditions.*

Proof Let us define $\varphi(\alpha) = f(x_k + \alpha d_k)$. Since $\varphi(\alpha)$ is bounded from below for all $\alpha > 0$, the line $l(\alpha) = f(x_k) + \alpha\rho\nabla f(x_k)^T d_k$ must intersect the graph of φ at least once. Let $\alpha' > 0$ be the smallest intersection value of α, i.e.

$$f(x_k + \alpha' d_k) = f(x_k) + \alpha'\rho\nabla f(x_k)^T d_k < f(x_k) + \rho\nabla f(x_k)^T d_k. \tag{6}$$

Hence, a sufficient decrease holds for all $0 < \alpha < \alpha'$.
Now, by the mean value theorem, there exists $\alpha'' \in (0, \alpha')$ so that

$$f(x_k + \alpha' d_k) - f(x_k) = \alpha' \nabla f(x_k + \alpha'' d_k)^T d_k. \tag{7}$$

Since $\rho < \sigma$ and $\nabla f(x_k)^T d_k < 0$, from (6) and (7) we get

$$\nabla f(x_k + \alpha'' d_k)^T d_k = \rho \nabla f(x_k)^T d_k > \sigma \nabla f(x_k)^T d_k. \quad (8)$$

Therefore, α'' satisfies the Wolfe line search conditions (4) and (5) and the inequalities are strict. By smoothness assumption on f, there is an interval around α'' for which the Wolfe conditions hold. ♦

The structure of this technical report is as follows. Section 2 is dedicated to present an implementation of Wolfe line search with cubic interpolation. Section 3 presents an implementation of Wolfe line search with bisection method. Section 4 illustrates the performances of some conjugate gradient methods with these two procedures for stepsize computation based on the Wolfe line search (4) and (5).

2. Wolfe line search with cubic interpolation

In the following, let us describe *a variant* of a line search procedure which is simple enough to generate safeguarded stepsizes satisfying the standard Wolfe conditions (4) and (5) (see Shanno, 1983). Suppose that we are at the iteration k. To have a simple interpretation of the procedure and a clear description, a Fortran version of it is presented in Figure 1. The inputs of this procedure are: n the number of variables, $x = x_k$ a vector with the current values of variables, $f = f(x_k)$ the value of the minimizing function in x, $d = d_k$ the current search direction, $gtd = \nabla f(x_k)^T d_k$ the scalar product of the current gradient and the search direction, $dnorm = \|d_k\|$ the l_2-norm of the search direction. The outputs of the procedure are: $alpha = \alpha_k$ the stepsize satisfying the standard Wolfe line search conditions, $xnew = x_{k+1} = x_k + \alpha_k d_k$ the new point, $fnew = f(x_{k+1})$ the function value in new point, $gnew = \nabla f(x_{k+1})$ the gradient of the minimizing function in the new point, $fgcnt$ the number of function and its gradient calls, $lscnt$ indicates that the line search procedure performed a number of iterations, $lsflag$ indicates that the number of iterations in the line search procedure is greater than a prespecified threshold.

```
      subroutine LineSearch (n,x,f,d,gtd,dnorm,alpha,xnew,fnew,gnew,
     +                       fgcnt,lscnt,lsflag)

C     SCALAR ARGUMENTS
      integer n,fgcnt,lscnt,lsflag
      double precision f,gtd,dnorm,alpha,fnew
C     ARRAY ARGUMENTS
      double precision x(n),d(n),xnew(n),gnew(n)
C     LOCAL SCALARS
      integer i,lsiter, max$ls
      double precision alphap,alphatemp,fp,dp,gtdnew,a,b
      common/acca/epsm

      lsflag = 0
* Maximum number of iterations in LineSearch is max$ls (now is 20)
      max$ls=20

      alphap = 0.0d0
      fp     = f
      dp     = gtd

      do i = 1,n
        xnew(i) = x(i) + alpha * d(i)
      end do
c1
      call evalfg(n,xnew,fnew,gnew)
      fgcnt = fgcnt + 1
```

```
      gtdnew = 0.0d0
      do i = 1,n
        gtdnew = gtdnew + gnew(i) * d(i)
      end do

      lsiter = 0

 10   if ( alpha * dnorm .gt. 1.0d-30 .and. lsiter .lt. max$ls .and.
     +     .not. ( gtdnew .eq. 0.0d0 .and. fnew .lt. f ) .and.
     +     ( ( fnew .gt. f + 1.0d-04 * alpha * gtd .or.
     +     dabs( gtdnew / gtd ) .gt. 0.9d0 ) .or. ( lsiter .eq. 0 .and.
     +     dabs( gtdnew / gtd ) .gt. 0.5d0 ) ) ) then

 20       if ( alpha * dnorm .gt. 1.0d-30 .and. fnew .gt. f .and.
     +         gtdnew .lt. 0.0d0 ) then

              alpha = alpha / 3.0d0
              do i = 1,n
                xnew(i) = x(i) + alpha * d(i)
              end do
c2
              call evalfg(n,xnew,fnew,gnew)
              fgcnt = fgcnt + 1
              gtdnew = 0.0d0
              do i = 1,n
                gtdnew = gtdnew + gnew(i) * d(i)
              end do
              alphap = 0.0d0
              fp     = f
              dp     = gtd
              goto 20
          end if

          a = dp + gtdnew - 3.0d0 * ( fp - fnew ) / ( alphap - alpha )
          b = a ** 2 - dp * gtdnew
          if ( b .gt. epsm ) then
              b = sqrt( b )
          else
              b = 0.0d0
          end if

          alphatemp = alpha - ( alpha - alphap ) * ( gtdnew + b - a ) /
     +                ( gtdnew - dp + 2.0d0 * b )

          if ( gtdnew / dp .le. 0.0d0 ) then
              if ( 0.99d0 * dmax1( alpha, alphap ) .lt. alphatemp .or.
     +             alphatemp .lt. 1.01d0 * dmin1( alpha, alphap ) ) then
                  alphatemp = ( alpha + alphap ) / 2.0d0
              end if
          else
              if ( gtdnew .lt. 0.0d0 .and.
     +             alphatemp .lt. 1.01d0 * dmax1( alpha, alphap ) ) then
                  alphatemp = 2.0d0 * dmax1( alpha, alphap )
              end if
              if ( ( gtdnew .gt. 0.0d0 .and.
     +             alphatemp .gt. 0.99d0 * dmin1( alpha, alphap ) ) .or.
     +             alphatemp .lt. 0.0d0 ) then
                  alphatemp = dmin1( alpha, alphap ) / 2.0d0
              end if
          end if

          alphap = alpha
          fp     = fnew
          dp     = gtdnew
          alpha = alphatemp
          do i = 1,n
            xnew(i) = x(i) + alpha * d(i)
          end do
c3
          call evalfg(n,xnew,fnew,gnew)
```

```
          fgcnt = fgcnt + 1
          gtdnew = 0.0d0
          do i = 1,n
            gtdnew = gtdnew + gnew(i) * d(i)
          end do

          lsiter = lsiter + 1

        goto 10
        end if

        if ( lsiter .ge. max$ls ) then
            lsflag = 1
        end if
        if ( lsiter .ne. 0 ) then
            lscnt = lscnt + 1
        end if

        return
        end
```

Fig. 1. Subroutine LineSearch which generate safeguarded stepsizes satisfying the standard Wolfe line search with cubic interpolation

In Figure 1, **max$ls** is the maximum number of iterations in the line search procedure, **epsm** is the epsilon machine and **evalfg(n,xnew,fnew,gnew)** is the subroutine implementing the algebraic expressions of the minimizing function and its gradient. In input, this subroutine has: **n** as the number of variables and **xnew** as the new point. In output, it computes: **fnew** as the value of function f in the new point and **gnew** as the gradient of f in the new point.

We see that a line search procedure is complicated and to be reliable it must incorporate a lot of features. Firstly, observe that the standard Wolfe conditions are implemented in a complicated form, which takes into consideration both the ratio between the rate of decrease of f in the direction d_k at the new point and the rate of decrease in the direction d_k at the current point x_k, and also some precautions to avoid too small or too large values of the stepsize. Observe that in the selection phase of the procedure, the cubic interpolation is used. Cubic interpolation provides a good model for the minimizing function in the searching interval. Suppose we have an interval $[\bar{a},\bar{b}]$ containing the desirable stepsize and two previous stepsizes estimates α_{i-1} and α_i in this interval. The algorithm uses a cubic function to interpolate the values, $\varphi_k(\alpha_{i-1})$, $\varphi_k(\alpha_i)$, $\varphi_k'(\alpha_{i-1})$ and $\varphi_k'(\alpha_i)$, where $\varphi_k(\alpha) = f(x_k + \alpha d_k)$. The minimizer of this cubic function in $[\bar{a},\bar{b}]$, that is a new estimation of the stepsize, is either at one of the endpoints or in the interior, case in which it is given by

$$\alpha_{i+1} = \alpha_i - (\alpha_i - \alpha_{i-1})\left[\frac{\varphi_k'(\alpha_i) + b - a}{\varphi_k'(\alpha_i) - \varphi_k'(\alpha_{i-1}) + 2b}\right], \tag{9}$$

where

$$a = \varphi_k'(\alpha_{i-1}) + \varphi_k'(\alpha_i) - 3\frac{\varphi_k(\alpha_{i-1}) - \varphi_k(\alpha_i)}{\alpha_{i-1} - \alpha_i}, \tag{10}$$

$$b = \left(a^2 - \varphi_k'(\alpha_{i-1})\varphi_k'(\alpha_i)\right)^{1/2}. \tag{11}$$

In Figure 1 the new estimate α_{i+1} is computed as **alphatemp.** The interpolation process can be repeated by discarding the data at one of the stepsizes α_{i-1} or α_i and replacing it by $\varphi_k(\alpha_{i+1})$ and

$\varphi_k'(\alpha_{i+1})$. Observe that the interpolation step that determines a new estimation to the stepsize is safeguarded in order to ensure that the new stepsize is not too close to the endpoints of the interval. Some more details may be found, for example, in (Dennis & Schnabel, 1983), (Shanno, 1983).

3. Wolfe line search with a simple bisection method

The bisection method is a simple method to find a zero of a continuous function for which two values of opposite sign are known. To determine a stepsize α_k which satisfies the Wolfe line search conditions (4) and (5) the following simple algorithm based on bisection concept may be used.

Algorithm WSB.

Step 1. Choose $\alpha_k > 0$ and set $\alpha_k^{low} = \alpha_k^{high} = 0$.

Step 2. If α_k satisfies (4), then go to Step 4.

Step 3. Else (if α_k do not satisfy (4)), then set: $\alpha_k^{high} = \alpha_k$ and $\alpha_k = (\alpha_k^{low} + \alpha_k^{high})/2$ and go to Step 2.

Step 4. If α_k satisfies (5), then stop.

Step 5. Otherwise (if α_k do not satisfy (5)), then set: $\alpha_k^{low} = \alpha_k$ and

$$\alpha_k = \begin{cases} 2\alpha_k^{low}, & if\ \alpha_k^{high} = 0, \\ (\alpha_k^{low} + \alpha_k^{high})/2, & if\ \alpha_k^{high} > 0, \end{cases}$$

and go to Step 2. ♦

For the very beginning let us proof that the above algorithm WSB determines a value for α_k which satisfy both the Wolfe line search conditions (4) and (5) in a finite number of steps.

Theorem 2. *Suppose that the function f is continuously differentiable on $\mathbb{R}^n$ and bounded below on the half-line $\{x_k + \alpha d_k : \alpha > 0\}$. Then, the algorithm WSB terminates in finite time and generates a value for α_k that satisfies Wolfe conditions (4) and (5).*

Proof Let us define $\varphi(\alpha) = f(x_k + \alpha d_k)$ and introduce the following two sets:

$$S_1 = \{\alpha > 0 :\ (4)\ \text{holds}\},$$
$$S_2 = \{\alpha > 0 :\ (5)\ \text{holds}\}.$$

Observe that both S_1 and S_2 are closed in $\mathbb{R}_+$. Moreover, for α sufficiently small, because φ' is continuous and $\rho < 1$,

$$\varphi(\alpha) = \varphi(0) + \int_0^\alpha \varphi'(\tau)d\tau < \varphi(0) + \int_0^\alpha \rho\varphi'(0)d\tau.$$

Therefore, there exists $\delta_1 > 0$ such that $[0, \delta_1] \subset S_1$, i.e. there exists $\alpha > 0$ satisfying the first Wolfe condition (4). Now, consider the second Wolfe condition (5). Let $\alpha > 0$ and two sequences $\{\alpha_k^{low,[i]}\}_{\mathbb{N}} \subset S_1$ and $\{\alpha_k^{high,[i]}\}_{\mathbb{N}} \subset S_2$ be such that

$$\alpha_k^{low,[i]} < \alpha \quad \text{for any } i \in \mathbb{N}, \quad \alpha_k^{low,[i]} \underset{i\to\infty}{\to} \alpha, \tag{12}$$

$$\alpha_k^{high,[i]} > \alpha \quad \text{for any } i \in \mathbb{N}, \quad \alpha_k^{high,[i]} \underset{i\to\infty}{\to} \alpha. \tag{13}$$

With this let us prove that $\alpha \in S_2$. For this assume that $\alpha \notin S_2$ and hence $\varphi'(\alpha) \le \sigma\varphi'(0)$. Then, since φ' is continuous and $\rho < \sigma$, there exists a value $\delta_2 > 0$ such that for any $\theta \in [0, \delta_2]$ it follows that $\varphi'(\alpha + \theta) \le \sigma\varphi'(0)$. Therefore, for all $\theta \in [0, \delta_2]$, it follows that

$$\varphi(\alpha + \theta) = \varphi(\alpha) + \int_{\alpha}^{\alpha+\theta} \varphi'(\tau) d\tau < \varphi(0) + (\alpha + \theta)\sigma\varphi'(0).$$

But, since $\alpha_k^{high,[i]}$ converges to α from the right, it follows that there exists an index j large enough so that $\alpha_k^{high,[j]} \in [\alpha, \alpha + \delta_2]$, thus contradicting the assumption that $\alpha_k^{high,[j]} \in S_1$. Therefore, $\alpha \in S_2$.

Now, let us prove that the algorithm terminates in finite time. If the algorithm terminates in a finite time, then α_k generated by it satisfies both Wolfe line search conditions. For this let us define $\alpha_k^{low,[i]}$, $\alpha_k^{high,[i]}$ and $\alpha_k^{[i]}$ as the values of α_k^{low}, α_k^{high} and α_k at the beginning of iteration i of the algorithm. The following properties can be observed:

1) Observe that for all i it is impossible that $\alpha_k^{low,[i]} = 0$. This is because in this case $\alpha_k^{[i]} = 2^{-i}\alpha_k^{[0]}$, and hence $\alpha_k^{[i]} \in [0, \delta_1]$ and α_k^{low} is updated to $\alpha_k^{[i]} > 0$ (see Step 5 of the algorithm).

2) The sequence $\{\alpha_k^{low,[i]}\}_{\mathbb{N}}$ is an increasing one in S_1 such that for all i, $\alpha_k^{low,[i]} < \alpha_k^{[i]}$. α_k^{low} can only be updated in Step 5 of the algorithm WSB. It will be increased to the strictly larger value $\alpha_k^{low,[i+1]} = \alpha_k^{[i]}$ and as we can see in Step 5 of the algorithm $\alpha_k^{[i+1]}$ takes on a strictly larger value than $\alpha_k^{[i]}$.

3) Initially, for a few iterations $\alpha_k^{high,[i]} = 0$. But, once it takes on a value $\alpha_k^{[i_0]} > 0$ at some iteration i_0, then this can only happen in Step 3 of the algorithm WSB. Starting with the iteration i_0, the elements of the sequence $\alpha_k^{high,[i]}$ with $i \ge i_0$ decrease in S_1, because α_k^{high} is only updated in Step 3 of the algorithm to a value of α_k that is strictly smaller than α_k^{high}, and α_k is itself updated to a strictly smaller value.

4) All in all only two possibilities can appear: Either $\alpha_k^{high,[i]} = 0$ for all i, and then $\alpha_k^{low,[i]} = 2^{i-1}\alpha_k^{[0]}$ for all i, in which case the algorithm WSB finds that the function f is unbounded. However, this case must be excluded since in the assumptions of theorem f is bounded. Or, there exists an index $i_0 \in \mathbb{N}$ such that $\alpha_k^{high,[i_0]} > 0$, and therefore $\alpha_k^{[i]} = (\alpha_k^{low,[i]} + \alpha_k^{high,[i]})/2$, the sequence $\{\alpha_k^{low,[i]}\}_{\mathbb{N}}$ is increasing, the sequence $\{\alpha_k^{high,[i]}\}_{\mathbb{N}}$ is decreasing and the interval $[\alpha_k^{low,[i]}, \alpha_k^{high,[i]}]$ is halved in length in every iteration. Therefore, it follows that $\alpha_k^{low,[i]}$ converges to a point α_k from S_1 and $\alpha_k^{high,[i]}$ converges to the same point in S_1. In conclusion $\alpha_k \in S_1 \cap S_2$. Therefore, $\alpha_k^{low,[i]} \in S_1 \cap S_2$ for i sufficiently large, proving that the algorithm terminates with this value. ♦

Observe that this is a very simple procedure for computing a value of α_k which satisfies the standard Wolfe line search conditions (4) and (5) without any safeguarding to the too large or too small values of α_k. A Fortran implementation of this algorithm is presented in Figure 2.

```
        subroutine LSbis(n,x,f,d,gtd,alpha,xnew,fnew,gnew,fgcnt, nexp)

        parameter(ia=50000)
        double precision x(n), d(n), xnew(n), gnew(n)
        double precision xtemp(ia), ftemp, gtemp(ia)
        double precision f, fnew, gtd, gtd1
        double precision alpha, alphalow, alphahigh
        integer fgcnt, nexp, ils

c       alpha     = 1.d0
        alphalow  = 0.d0
        alphahigh = 0.d0

c Parameters in Wolfe line search (cw1=rho, cw2=sigma)
        cw1 = 0.0001d0
        cw2 = 0.8d0

c ils is the number of iterations in this procedure.
C It is limited to 20.
        ils = 0

10      continue
        ils = ils + 1
        if(ils .gt. 20) return

        do i=1,n
          xtemp(i) = x(i) + alpha*d(i)
        end do
        call evalfg(n,xtemp,ftemp,gtemp, nexp)
        fgcnt = fgcnt + 1

c Test the first Wolfe line search
        if(ftemp .le. f + cw1*alpha*gtd) then
          go to 50

        else

          alphahigh = alpha
          alpha = (alphalow + alphahigh)/2.d0
          go to 10
        end if

50      continue

        gtd1 = 0.d0
        do i=1,n
          gtd1 = gtd1 + gtemp(i)*d(i)
        end do

c Test the second Wolfe line search
        if(gtd1 .ge. cw2*gtd) then
          go to 100
        else
          alphalow = alpha
          if(alphahigh .eq. 0.d0) alpha = 2.d0*alphalow
          if(alphahigh .gt. 0.d0) alpha = (alphalow+alphahigh)/2.d0
        end if

        go to 10

100     continue

C Compute: xnew, fnew and gnew as the outputs of the subroutine
        do i=1,n
```

```
      xnew(i) = x(i) + alpha*d(i)
    end do
    call evalfg(n,xnew,fnew,gnew, nexp)
    fgcnt = fgcnt + 1

    return
    end
```

Fig. 2. Subroutine LSbis which generate stepsizes satisfying the standard Wolfe line search with a simple bisection method.

For calling this subroutine a value for **alpha** must be given in its input. A possibility is to consider **alpha=1**. (Observe that this line is under comment.) In our numerical experiment we considered **alpha = alpha * dnormprev / dnorm,** where **dnormprev =** $\|d_{k-1}\|_2$ and **dnorm =** $\|d_k\|$.

4. Numerical experiments

In this section, let us report some numerical results obtained with a Fortran implementation of the accelerated conjugate gradient algorithms in which the stepsize is computed by using the standard Wolfe line search with cubic interpolation (see Figure 1) or by using the standard Wolfe line search with a simple bisection method (see Figure 2).

All algorithms have been coded in double precision Fortran and compiled with f77 (default compiler settings) and run on an Intel Pentium 4, 1.8 GHz workstation. The test functions are from the UOP collection (Andrei, 2018), which includes 80 functions. For each test function ten numerical experiments with the number of variables $n = 1000, 2000, ..., 10000$ have been considered, thus obtaining a number of 800 problems.

The algorithms compared in these numerical experiments find local solutions. Therefore, the comparisons of the algorithms are given in the following context. Let f_i^{ALG1} and f_i^{ALG2} be the optimal value found by ALG1 and ALG2 for problem $i = 1, \ldots, 800,$ respectively. We say that, in the particular problem $i,$ the performance of ALG1 was better than the performance of ALG2 if

$$\left| f_i^{ALG1} - f_i^{ALG2} \right| < 10^{-3} \tag{14}$$

and if the number of iterations (#iter), or the number of function-gradient evaluations (#fg), or the CPU time of ALG1 was less than the number of iterations, or the number of function-gradient evaluations, or the CPU time corresponding to ALG2, respectively.

The iterations are stopped if the inequality $\|g_k\|_\infty \leq 10^{-6}$ is satisfied, where $\|.\|_\infty$ is the maximum absolute component of a vector. All algorithms implement the standard Wolfe line search (4) and (5), where $\rho = 0.0001$ and $\sigma = 0.8.$ The maximum number of iterations was limited to 2000.

In the first set of numerical experiments the Hestenes-Stiefel (Hestenes & Stiefel, 1952) conjugate gradient algorithm, in which the conjugate gradient parameter is computed as

$$\beta_k^{HS} = g_{k+1}^T y_k \,/\, y_k^T d_k, \tag{15}$$

is considered, where $y_k = g_{k+1} - g_k.$ For this set of numerical experiments the total number of iterations, the total number of function and its gradient evaluations and the total CPU computing

time for solving this set of 800 unconstrained optimization problems using HS algorithm with these line search procedures are as in Table 1.

Table 1

HS with Cubic Wolfe Line Search (HSCLS)

```
Grand Total iterations      =   291234
Grand Total FG evaluations  =   705103
Grand Total time (seconds)  =   460.74
```

HS with Simple Bisection Wolfe Line Search (HSSBLS)

```
Grand Total iterations      =   359027
Grand Total FG evaluations  =  3100101
Grand Total time (seconds)  =  1666.81
```

To compare the performances of algorithms, the Dolan and Moré (2002) performance profiles are used. Figure 3 presents these performance profiles subject to the CPU time metric.

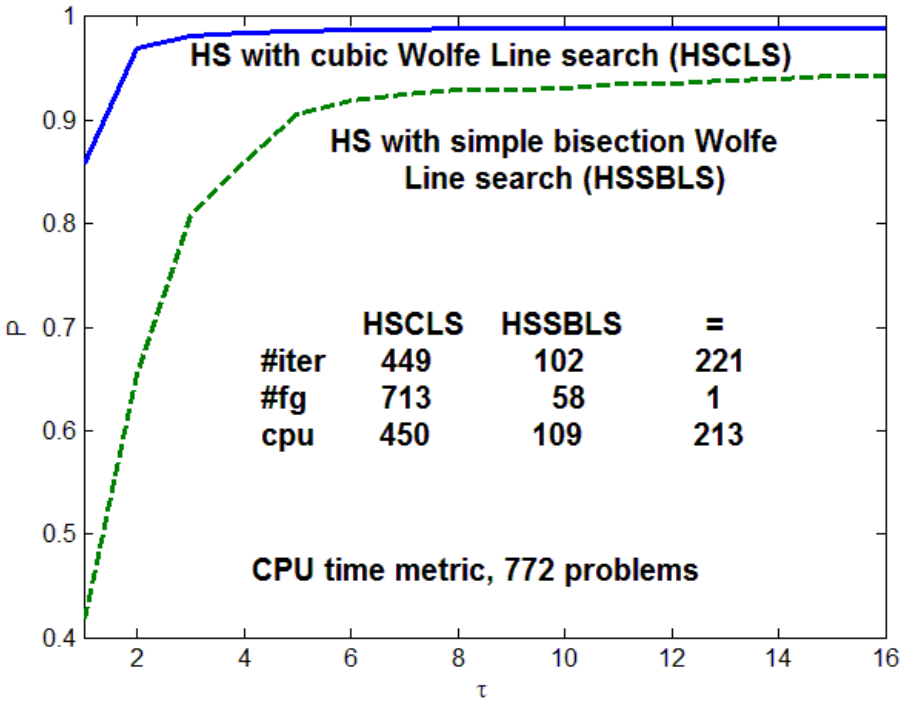

Fig. 3. Performance profiles of HS with cubic Wolfe line search versus HS with simple bisection Wolfe line search.

Comparing HSCLS versus HSSBLS (see Figure 3) subject to the number of iterations, we see that HSCLS was better in 449 problems (i.e. it achieved the minimum number of iterations in 449 problems), while HSSBLS was better only in 102 problems. Out of 800 problems considered in this set of numerical experiments, only for 772 problems did the criterion (14) hold. At the same time observe that HSCLS was faster for solving 450 problems. On the other hand, HSSBLS was faster only for solving 109 problems.

In the second set of numerical experiments let us consider the PRP+ conjugate gradient (Polak & Ribière, 1969), (Polyak, 1969), where the conjugate gradient parameter is computed as

$$\beta_k^{PRP+} = \max\{0, g_{k+1}^T y_k / g_k^T g_k\}. \tag{16}$$

In this case the total number of iterations, the total number of function and its gradient evaluations and the total CPU computing time for solving this set of 800 unconstrained optimization problems using PRP+ algorithm with these line search procedures are as in Table 2.

Table 2

PRP+ with Cubic Wolfe Line Search (PRPCLS)

```
Grand Total iterations     =   408895
Grand Total FG evaluations =   721314
Grand Total time (seconds) =   609.68
```

PRP+ with Simple Bisection Wolfe Line Search (PRPSBLS)

```
Grand Total iterations     =   362910
Grand Total FG evaluations =  3128826
Grand Total time (seconds) =  2053.80
```

Figure 4 presents the Dolan and Moré performance profiles of these algorithms subject to the CPU time metric.

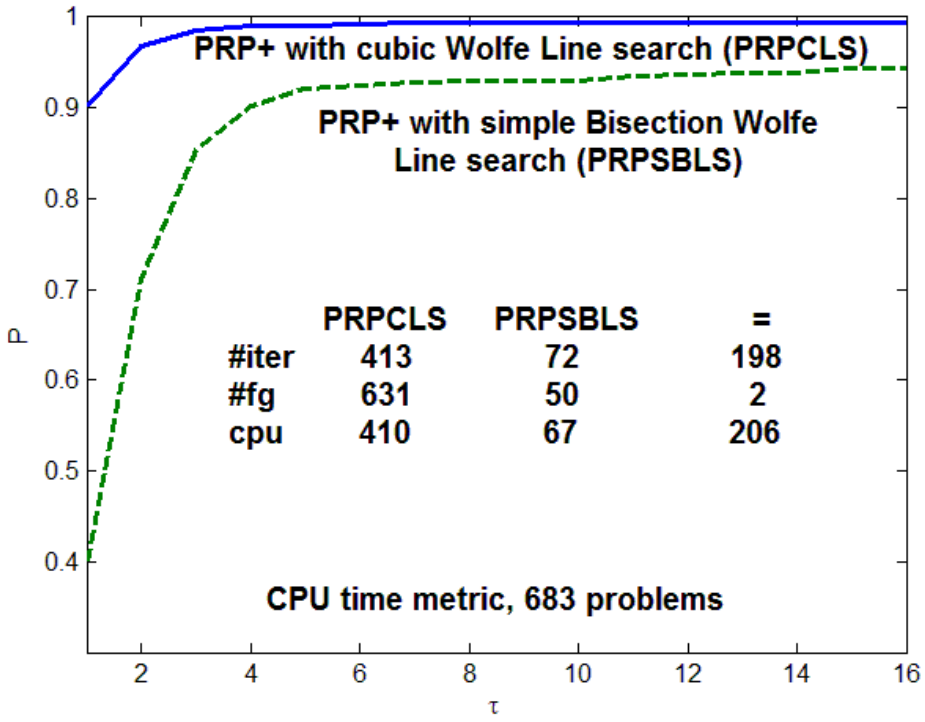

Fig. 4. Performance profiles of PRP+ with cubic Wolfe line search versus PRP+ with simple bisection Wolfe line search.

Comparing PRPCLS versus PRPSBLS (see Figure 4) subject to the number of iterations, we see that PRPCLS was better in 413 problems, while PRPSBLS was better in 72 problems, etc. Out of

800 problems considered in this set of numerical experiments, only for 683 problems did the criterion (14) hold. Observe that PRPCLS was faster for solving 410 problems, but PRPSBLS was faster only for solving 67 problems. Obviously, a more advanced line search procedure for computing the stepsize in conjugate gradient algorithms is more benefic.
In the third set of numerical experiments the DY conjugate gradient method is considered (Dai & Yuan, 1999), where the conjugate gradient parameter is computed as

$$\beta_k^{DY} = g_{k+1}^T g_{k+1} / y_k^T d_k. \tag{17}$$

Table 3 presents the global performances of DY for solving this set of problems.

Table 3

DY with Cubic Wolfe Line Search (DYCLS)

```
Grand Total iterations      =   288388
Grand Total FG evaluations  =   713280
Grand Total time (seconds)  =   415.74
```

DY with Simple Bisection Wolfe Line Search (DYSBLS)

```
Grand Total iterations      =   374945
Grand Total FG evaluations  =  3289585
Grand Total time (seconds)  =  1916.05
```

Figure 5 presents the Dolan and Moré performance profiles of these algorithms.

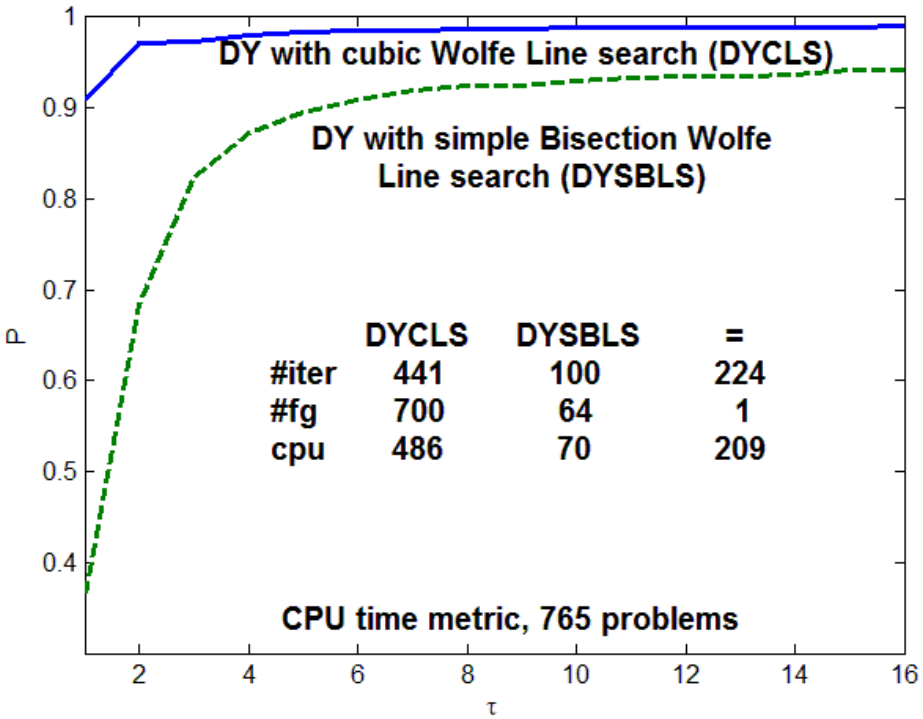

Fig. 5. Performance profiles of DY with cubic Wolfe line search versus DY with simple bisection Wolfe line search.

The percentage of problems for which an algorithm is the best is given on the left side of the plot. On the other hand, the right side of the plot gives the percentage of the problems that are successfully solved. In other words, for a given algorithm, the plot for $\tau = 1$, represents the fraction of problems for which the algorithm was the most efficient over all algorithms. The plot for $\tau = \infty$ represents the fraction of problems solved by the algorithm irrespective of the required effort. Therefore, the plot for $\tau = 1$ is associated to the *efficiency* of the algorithm, while the plot for $\tau = \infty$ is associated to the *robustness* of the algorithm. Observe that the conjugate gradient algorithms with standard Wolfe line search using the cubic interpolation are more efficient and more robust than the same algorithms using a simple bisection Wolfe line search.

5. Conclusion

Conjugate gradient methods are very sensitive to the procedure for computing the stepsize α_k. More advanced procedures for stepsize computation in conjugate gradient methods, more efficient and more robust algorithms. In conjugate gradient methods computing the search direction d_k is very simple, the difficulty is determination of the stepsize α_k. In this technical report we compared two implementations of the standard Wolfe line search in the frame of conjugate gradient methods. One, very sophisticated, using cubic interpolation with safeguarding the values of α_k, and another one very simple based on the bisection method. The convergence of the Wolfe line search with simple bisection algorithm (see Theorem 2) is only linear. Its efficiency is very modest.

It is remarkable to notice that even if the conjugate gradient methods are implemented using the acceleration scheme (see Andrei (2020), chapter 5) which modifies the stepsize determined by the Wolfe line search conditions in a multiplicative manner, this does not eliminate the efforts to compute a stepsize as accurate as possible to get efficient and robust algorithms. The procedure for stepsize computation is a critical point in conjugate gradient methods.

The determination of the stepsize continue to be a very intensive research activity. Some other algorithms for stepsize determination are described by Hager and Zhang (2005), Dai and Kou (2013), Zhang and Hager (2004), Gu and Mo (2008), Ou and Liu (2017). No comparisons among these line search algorithms are known.

References

1. Andrei, N., (2020). *Nonlinear Conjugate Gradient Methods for Unconstrained Optimization*. Springer Optimization and Its Applications, vol. 158, Springer.
2. Shanno, D.F., (1983). *CONMIN – A Fortran subroutine for minimizing an unconstrained nonlinear scalar valued function of a vector variable x either by the BFGS variable metric algorithm or by a Beale restarted conjugate gradient algorithm*. Private communication, October 17, 1983.
3. Dennis, J.E., & Schnabel, R.B., (1983). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewoods Cliffs, New Jersey. [Reprinted as Classics in Applied Mathematics 16, SIAM, Philadelphia, USA, 1996.]
4. Andrei, N., (2018). *UOP - A collection of 80 unconstrained optimization test problems*. (Technical Report No. 7/2018, November 17, Research Institute for Informatics, Bucharest, Romania).
5. Hestenes, M.R., & Stiefel, E., (1952). *Methods of conjugate gradients for solving linear systems.* Journal of Research of the National Bureau of Standards, 49, 409-436.
6. Dolan, E.D., & Moré, J.J., (2002). *Benchmarking optimization software with performance profiles.* Mathematical Programming, 91, 201-213.
7. Polak, E., & Ribiére, G., (1969). *Note sur la convergence de méthods de direction conjugées*. Revue Francaise d'Informatique et de Recherche Opérationnelle, 16, 35-43.

8. Polyak, B.T., (1969). *The conjugate gradient method in extremal problems*. USSR Computational Mathematics and Mathematical Physics, 9, 94-112.
9. Dai, Y.H., & Yuan, Y., (1999). *A nonlinear conjugate gradient method with strong global convergence property*. SIAM Journal on Optimization, 10, 177-182.
10. Hager, W.W., & Zhang, H., (2005). *A new conjugate gradient method with guaranteed descent and an efficient line search.* SIAM Journal on Optimization, 16, 170-192.
11. Dai, Y.H., & Kou, C.X., (2013). *A nonlinear conjugate gradient algorithm with an optimal property and an improved Wolfe line search.* SIAM Journal on Optimization, 23(1) 296-320.
12. Zhang, H., & Hager, W.W., (2004). *A nonmonotone line search technique and its application to unconstrained optimization.* SIAM Journal on Optimization, 14, 1043-1056.
13. Gu, N.Z. & Mo, J.T., (2008). *Incorporating nonmonotone strategies into the trust region method for unconstrained optimization*, Computers and Mathematics with Applications, 55, 2158-2172.
14. Ou, Y., & Liu, Y., (2017). *A memory gradient method based on the nonmonotone technique*. Journal of Industrial and Management Optimization, 13(2), 857-872.

-----ooooo0ooooo-----