An assessment of non-monotone linesearch techniques for unconstrained optimization

Philippe L. Toint

Report 94/14 May 19, 1994

Department of Mathematics
Facultés Universitaires ND de la Paix
B-5000 Namur, Belgium

Current reports available by anonymous ftp from the directory `pub/reports` on thales.math.fundp.ac.be (internet 138.48.4.14).

Keywords: non-monotone algorithms, linesearch, unconstrained optimization.

An assessment of non-monotone linesearch techniques for unconstrained optimization

Philippe L. Toint

May 19, 1994

Abstract

The purpose of this paper is to discuss the potential of non-monotone techniques for enforcing convergence of unconstrained minimization algorithms from starting points distant from the solution. Linesearch based algorithms are considered for both small and large problems, and extensive numerical experiments show that this potential is sometimes considerable. A new variant is introduced in order to limit some of the identified drawbacks of the existing techniques. This variant is again numerically tested and appears to be competitive. Finally, the impact of preconditioning on the considered methods is examined.

1 Introduction

In this paper, we consider algorithms for the numerical solution of the problem

$$\min_{x \in \mathbf{R}^n} \quad f(x), \tag{1.1}$$

where $f(x)$ is a real-valued smooth function, which we assume is bounded below. Most widely available algorithms for solving the *nonlinear unconstrained minimization problem* (1.1) are, today, *descent methods*, in that they generate a sequence of iterates $\{x_k\}$ such that the associated sequence of objective function values $\{f(x_k)\}$ is monotonically decreasing. This is the case of all methods described in Dennis and Schnabel (1983), a now classical reference on this subject, as that of many packages for solving unconstrained problems, as UNCMIN of Koontz *et al.* (1985), TENMIN of Schnabel and Chow (1991), LANCELOT of Conn *et al.* (1992) and the Harwell Subroutines VA15 by Liu and Nocedal (1989), VE08 by Toint (1983) and VE10 by Toint (1987), to cite just a few. This monotonicity property may be considered as natural, since one wishes to find a point with the lowest possible objective function value, and has been extensively used, both in theory and practice, to ensure that convergence to a minimizer occurs, even when the minimization is started far away from such a point. The reader is referred to Byrd *et al.* (1992), Conn *et al.* (1988), Burke *et al.* (1990) or Toint (1988) for examples of convergence theory heavily relying on this important property.

It has however been noted that imposing monotonicity of the sequence of objective function values is not without drawbacks. A typical case where monotonicity can cause severe loss of efficiency is when the objective function features deep narrow curved valleys: once trapped near the bottom of such a valley, the sequence of iterates must follow the valley's floor rather closely, which can result in very short steps or even undesired zigzaging. This observation has led researchers to propose algorithms that do not always guarantee the monotonicity properties. In particular, Grippo, Lampariello and Lucidi published a few papers (see Grippo *et al.* (1986) and Grippo *et al.* (1991)) where they demonstrated on some examples that substantial efficiency gains could be achieved in the solution of (1.1) if one is ready to abandon, at least to some extent, the constraint of generating monotonically decreasing sequences of objective function values. They also provided the convergence analysis to support their algorithmic proposals. In all fairness, Grippo, Lampariello and Lucidi were not the only researchers interested in non-monotone techniques for nonlinear optimization: the watchdog technique of Chamberlain *et al.* (1982) is also a strategy of this type, although intended for the solution of constrained problems.

Surprisingly, and despite their obvious potential practical implications, the findings of Grippo and his coauthors didn't lead to any substantial validation by other researchers, at least as far as the author is aware. These ideas have been sometimes used in the litterature (see Kanzow (1994), for instance), but have not yet been incorporated in the mainstream research on nonlinear unconstrained minimization. It is the first purpose of this paper to re-examine their proposals and contribute to their evaluation. The second purpose is to present an improved variant of their original methods, also based on a succession of unidimensional (approximate) minimizations, usually known as "linesearches". The third and final aim of the paper is to examine the impact of preconditioning on the considered non-monotone algorithms.

The paper is organized as follows. The original proposals of Grippo *et al.* are considered in Section 2, where a specific implementation of these proposals is compared to a more classical monotone linesearch algorithm on a large collection of both small and larger test problems from the CUTE collection (see Bongartz *et al.* (1994)). An improved variant of these methods is introduced and tested in Section 3. Section 4 is devoted to the numerical investigation of the effect of preconditioning. Finally, we present some conclusions and perspectives in Section 5.

2 The algorithms of Grippo, Lampariello and Lucidi

2.1 The original proposals

The classical framework in which Grippo, Lampariello and Lucidi discuss their proposals is that of a "gradient related" method using an Armijo linesearch. If we denote the usual Euclidean inner product and norm by $\langle \cdot, \cdot \rangle$ and $\| \cdot \|$, respectively, and define $g_k = \nabla f(x_k)$, we may describe this basic method as follows.

Basic Linesearch Algorithm (BLS)

0. Initialisation. A starting point x_0 is given, as well as constants $\beta \in (0,1)$ and $\gamma \in (0,1)$. Set $k = 0$.

1. Test for convergence. Stop if the chosen convergence criteria are met.

2. Determination of a search direction. Find a direction d_k such that

$$\langle d_k, g_k \rangle \leq -\kappa_1 \|g_k\|^2 \tag{2.1}$$

and

$$\|d_k\| \leq \kappa_2 \|g_k\| \tag{2.2}$$

for some positive constants κ_1 and κ_2.

3. Linesearch. Determine the smallest integer j_k such that, if one defines

$$\alpha_k = \beta^{j_k}, \tag{2.3}$$

then

$$f(x_k + \alpha_k d_k) \leq f(x_k) + \gamma \alpha_k \langle g_k, d_k \rangle. \tag{2.4}$$

Then, set

$$x_{k+1} = x_k + \alpha_k d_k. \tag{2.5}$$

4. Prepare for the next iteration. Increment k by one and go to Step 1.

There are several possible ways to determine the step d_k in Step 2 of this prototype algorithm, in order to ensure that conditions (2.1) and (2.2) are satisfied. Grippo *et al.* propose to use the "modified Cholesky" factorization technique of Gill and Murray (1974). Other techniques will be discussed in Section 2.3, in relation with our numerical experiments.

For future reference, we will refer to the linesearch calculation of Step 3 as a procedure for obtaining x_{k+1} from the *base point* x_k (which determines the gradient $g_k = \nabla f(x_k)$), the *search direction* d_k and the *reference value* $f(x_k)$ (the first term of the right-hand side of (2.4)). More concisely, we will express this relationship by the statement

$$x_{k+1} = LS_{\beta,\gamma}(x_k, d_k, f(x_k)), \tag{2.6}$$

which is therefore an alternative equivalent specification of Step 3.

It is now easy to describe the original idea of Grippo *et al.* (1986): instead of using $f(x_k)$ as the reference value in Step 3, they propose to use an objective function value corresponding to a previous iterate. More precisely, they propose to define

$$r_k = \max_{j=0,\ldots,p} f(x_{k-j}) \tag{2.7}$$

for some fixed non-negative integer p and then to replace Step 3 in the basic algorithm by

$$x_{k+1} = LS_{\beta,\gamma}(x_k, d_k, r_k) \tag{2.8}$$

instead of (2.6). We will call the resulting algorithm the *GLL1 Algorithm.* As a consequence of (2.7), the strict monotonicity of the sequence $\{f(x_k)\}$ is no longer enforced, giving the algorithm more flexibility in the way in which successive iterates are obtained. Grippo *et al.* in fact describe and analyze several possible rules for the definition of the reference value r_k, but only provide numerical examples for the choice (2.7).

In a subsequent paper, Grippo *et al.* (1991) introduce a further relaxation of the monotonicity property. Briefly speaking, they argue that some steps could be *automatically* accepted, without even checking any condition like (2.4), provided that these steps are sufficiently short (they require the subsequence of the norms of these steps to be geometrically converging to zero). To avoid convergence to undesirable maxima or saddle points, the value of the objective function is checked once in a while: if a suitable decrease has not been obtained since the last such verification, the algorithm is then restarted from the iterate corresponding to this last acceptable value. The precise manner in which the reference value r_k is redefined and this idea realized is a little more complex than the modification leading to Algorithm GLL1. It can be casted in the framework of Algorithm BLS as follows.

$\boxed{\text{Algorithm GLL2}}$

0. Initialisation. A starting point x_0 is given, as well as constants $\beta \in (0,1)$, $\gamma \in (0,1)$, $\eta \in (0,1)$, $\mu > 0$, $\xi \in (0,1)$, $\Delta_0 > 0$, $p \geq 1$ and $N \geq 1$. Set $k = \ell = 0$ and $f_0 = r_0 = f(x_0)$.

1. Test for convergence. Compute g_k and stop if $\|g_k\|$ is sufficiently small.

2. Determination of a search direction. Find d_k such that (2.1) and (2.2) hold and such that

$$\|d_k\| \geq \kappa_3 \|g_k\|. \tag{2.9}$$

If x_k has been obtained by Step 5 or if $k = 0$, define $d_\ell = d_k$. Then, if $k = \ell + N$ go to Step 3. Otherwise, go to Step 4.

3. Tests after N automatic step acceptances. Compute $f(x_k)$ if necessary. If $f(x_k) \geq r_k$, set

$$x_k = x_\ell, \quad d_k = d_\ell, \quad \Delta_{k+1} = \Delta_k \tag{2.10}$$

and go to Step 5. Otherwise, increment ℓ by one and define

$$x_\ell = x_k, \quad d_\ell = d_k, \quad f_\ell = f(x_k) \text{ and } r_k = \max_{j=0,\ldots,p} f_{\ell-j}. \tag{2.11}$$

Then, if $\|d_k\| \leq \Delta_k$, set

$$x_{k+1} = x_k + d_k, \quad \Delta_{k+1} = \eta \Delta_k, \quad r_{k+1} = r_k \tag{2.12}$$

and go to Step 6. Otherwise go to Step 5.

4. Tests before N automatic step acceptances. If $\|d_k\| \leq \Delta_k$, perform Step 4.a. Otherwise, perform Step 4.b.

Step 4.a. If $\|d_k\| > \|d_{k-1}\|$, compute $f(x_k + d_k)$ and $f(x_k)$ if necessary. Then test if $f(x_k + d_k) - f(x_0) \geq \mu(f(x_0) - f(x_k))$. If this latter inequality holds, set

$$x_k = x_\ell, \quad d_k = d_\ell, \quad \Delta_{k+1} = \xi \Delta_k \tag{2.13}$$

and go to Step 5. In all other cases, set x_{k+1}, Δ_{k+1} and r_{k+1} according to (2.12) and go to Step 6.

Step 4.b. Compute $f(x_k)$ if necessary. Then, if $f(x_k) \geq r_k$, set x_k, d_k and Δ_{k+1} using (2.10) and go to Step 5. Otherwise, increment ℓ by one, recompute x_ℓ, d_ℓ, f_ℓ and r_k using (2.11) and go to Step 5.

5. Linesearch. Define

$$x_{k+1} = LS_{\beta,\gamma}(x_k, d_k, r_k), \tag{2.14}$$

increment ℓ by one and define

$$x_\ell = x_{k+1}, \quad f_\ell = f(x_{k+1}) \text{ and } r_{k+1} = \max_{j=0,\ldots,p} f_{\ell-j}. \tag{2.15}$$

6. Prepare for the next iteration. Increment k by one and go to Step 1.

We now briefly comment on this algorithm.

1. Condition (2.9) is introduced by Grippo *et al.* and makes the length of the step an adequate measure of criticality. Section 2.3 shows that it does not constitute a strong additional requirement in a practical algorithm.

2. The step d_k is automatically accepted (see Step 4) if its norm is at most equal to Δ_k and has not increased compared to that of d_{k-1}. In this case, the function value is not even computed. Observe also that the index ℓ denotes the last iterate at which the value of the objective function has been evaluated and has decreased enough. As indicated above, the algorithm "restarts" from this point if the objective function has not sufficiently diminished after N automatic acceptances.

3. The idea behind Step 4.a is that it might be advisable to enforce closer control of the monotonicity of the objective function values if these become significantly larger than the initial value $f(x_0)$: this behaviour combined with large steps may indeed indicate that the iterates are leaving the region of interest.

4. The mechanism for updating the reference value r_k is essentially identical to that used in GLL1, except that it only uses the last p function values that have been computed and which do not exceed their corresponding reference values. This distinction is necessary because, as indicated in the previous paragraph, some function values may have not been calculated when short steps are automatically accepted.

Grippo *et al.* (1991) indicate that Algorithm GLL2 may be more efficient than GLL1 and often compare favourably with the modified Newton routine E04LBF from the NAG library.

2.2 Convergence theory

Grippo and his co-authors proved that every limit point of the sequence of iterates produced by Algorithms GLL1 and GLL2 is a stationary point for problem (1.1). In order to obtain this result, they assume that the objective function is twice continuously differentiable on the compact level set $\{x \mid f(x) \le f(x_0)\}$. Their argument is no longer based on the (abandoned) decreasing nature of the sequence of objective function values $f(x_k)$, but rather on the monotonicity of the references values r_k. The analysis is slightly more involved for Algorithm GLL2, since the effect of restarting the algorithm after an "unsuccessful" series of N automatically accepted steps must be taken into account. The reader is referred to Grippo *et al.* (1986) and Grippo *et al.* (1991) for details.

Note that the same convergence result is automatically obtained for Algorithm BLS by viewing this algorithm as a special case of Algorithm GLL1 where $p = 0$, but may also be obtained by directly exploiting (2.1)–(2.5). In particular, (2.1) and (2.2) imply that BLS produces "gradient-related" search directions (see Ortega and Rheinboldt (1970), Section 14.3).

2.3 An implementation

Our implementation of Algorithms BLS, GLL1 and GLL2 in Fortran 77 is rather straighforward.

- Termination occurs when $\|g_k\| \le 10^{-5}$.
- The determination of the search direction d_k at Step 2 is achieved by solving a "perturbed" Newton system of the form

$$(H_k + M_k)d_k = -g_k + h_k \tag{2.16}$$

where H_k denotes $\nabla^2 f(x_k)$, where M_k is a symmetric positive semidefinite perturbation matrix such that the eigenvalues of $H_k + M_k$ are in the interval $[\kappa_4, \kappa_5]$ (for some positive constants κ_4 and κ_5 independent of k) and where the "residual" h_k satisfies the conditions

$$\langle d_k, h_k \rangle = 0 \tag{2.17}$$

and

$$\|h_k\| \le \|g_k\| \min(0.1, \|g_k\|^{1/2}). \tag{2.18}$$

Amongst the many possible ways of selecting the perturbation matrix M_k, we have chosen the modified conjugate-gradients (CG) method proposed by Arioli *et al.* (1993). This method guarantees both (2.17) and the uniform positive definite and bounded character of the matrices $H_k + M_k$ by construction. Note that the modified Cholesky factorization technique used by Grippo and co-authors in their papers also guarantees the same conditions (with $h_k = 0$), as would other types of modified factorizations (see Schnabel and Eskow (1991), Gill *et al.* (1992), Conn *et al.* (1992) or Schlick (1993), for instance). However, since our approach allows for nonzero h_k, we have to enforce (2.18) explicitly: this is simply achieved in our code by stopping the modified conjugate-gradients method whenever this condition is satisfied.

It is now easy to verify that (2.1), (2.2) and (2.9) follow from (2.17), (2.18) and the requirements imposed on the matrices $H_k + M_k$. We note first that (2.18) and (2.16) imply that

$$\|(H_k + M_k)d_k\| \geq \|g_k\| - \|h_k\| \geq 0.9\|g_k\|, \tag{2.19}$$

which yields that

$$\|d_k\| \geq \frac{0.9}{\kappa_5}\|g_k\|, \tag{2.20}$$

and (2.9) is satisfied. Using now (2.17), one can prove (see Griewank and Toint (1982)) that

$$\langle g_k, d_k\rangle \leq -[\mathrm{cond}(H_k + M_k)]^{-2}\|g_k\|\,\|d_k\| \tag{2.21}$$

where cond(X) is the condition number of the matrix X. Since the eigenvalues of $H_k + M_k$ are contained in $[\kappa_4, \kappa_5]$, its condition number is bounded above by κ_5/κ_4, which, together with (2.9), gives (2.1). For the same reason, one immediately obtains that (2.2) holds. Indeed, using (2.16),

$$\|d_k\| \leq \frac{1}{\kappa_4}(\|g_k\| + \|h_k\|) \leq \frac{1.1}{\kappa_4}\|g_k\|, \tag{2.22}$$

where we used (2.18) to derive the second inequality.

- In our code, the constants $\mu = 10^6$ and $\xi = 10^{-3}$ are used in Step 4.a, as recommended by Grippo *et al.* (1991).

- After some experimenting, a value of $p = 10$ was chosen in (2.11) and (2.15), the initial step threshold Δ_0 was set to unity and the maximum number of successive automatic step acceptances N was fixed at 10.

- The linesearch procedures follows Step 3 of Algorithm BLS nearly exactly (with $\gamma = 0.01$ and $\beta = 0.5$), except that doubling the stepsize is attempted if the unit step satisfies (2.4) and M_k nonzero, in an attempt to make a step as big as possible along a direction of (possibly) negative curvature.

- A maximum number of 10000 iterations and a maximum CPU time of 5 hours were allowed.

2.4 Numerical comparisons

We now present the results of an extensive numerical investigation of Algorithms BLS, GLL1 and GLL2. We have run these three methods on a set of 327 unconstrained problems available[1] from the CUTE collection (see Bongartz *et al.* (1994)). All tests were performed on a DEC 3000/500 under OSF and using version 3.3-35 of the Fortran compiler with the default optimizing option. As the total number of test runs discussed in this paper exceeds 2200, it is impossible to include their detailed results in this text and we only present statistical summaries. The complete results are however available as a separate report[2] (Toint (1994)).

[1] In March 1994.

[2] This report can be obtained by anonymous `ftp` from the directory `pub/reports` on `thales.math.fundp.ac.be` (internet 138.48.4.14).

In order to clarify the comparison, we divided our set of 327 problems in two subsets: the first contains all problems of dimension at most 500 (226 problems), and the second all larger cases (101 problems). This will allows us to distinguish the effects of the non-monotone strategies for small and large problems.

We first consider the relative reliability of our three algorithms on both problem sets and note that BLS failed on eleven small and six large problems, while GLL1 failed on nine small and five large problems and GLL2 on ten small and five large ones. Failures were caused by excessive number of iterations, excessive CPU time or by the occurence of an arithmetic error during the computation of the objective function. The latter did often happen in the early iterations where a large step was attempted that lead outside the region where the problem functions are well defined on the computer.

Amongst the remaining 212 small and 93 large problems that could be solved by all three algorithms, six small ones and one large exhibited more than one local minimizer and were therefore rejected from our test sets, leaving 206 small and 92 large problems for further analysis. Table 1 shows the cumulative statistics for the three methods on these problems. In this table, "iters" stands for the total number of iterations needed to solve all problems, "evals" for the total number of objective function evaluations, "CG-its" for the total number of iterations within the modified conjugate-gradients procedure defining the search direction and "time" for the total CPU time (rounded to the second). Note that the objective function's gradient and Hessian are evaluated once per iteration.

Method	206 small problems				92 large problems			
	iters	evals	CG-its	time	iters	evals	CG-its	time
BLS	8173	17926	134099	1752	5041	9231	541429	35233
GLL1	8249	22793	153943	1288	7564	17135	966247	22661
GLL2	8089	21637	151501	1300	7504	16795	968919	23586

Table 1: Cumulative statistics for BLS, GLL1 and GLL2

These cumulative measures tend to suggest that GLL1 and GLL2 provide an average improvement in CPU time on BLS for small problems. This is obtained at the expense of more CG iterations and function evaluations. The situation is similar for the larger problems.

In order to refine our conclusions, we now look at more disaggregate performance measures and report, in Table 2, how many times each of the three considered method has been best, second or last. In these comparisons, two CPU times are reputed equal is they correspond to runs yielding identical iterates or if differ by at most 5% of the best or by at most half a second. We immediately notice that the performance of GLL1 and GLL2 is much better than suggested by Table 1. In particular, we observe that GLL2 is most often the best method in CPU time, in CG iterations and, even more clearly, in function evaluations.

Finally, these results indicate very few reasons why one should use GLL1 instead of GLL2. The latter method is admittedly more complex to code, but very often preferable to GLL1, as is

Performance measure	Method	Small problems 1st	2nd	3rd	Large problems 1st	2nd	3rd
Iterations	BLS	174	20	12	80	7	5
	GLL1	153	51	2	67	20	5
	GLL2	162	39	5	68	20	4
Evaluations	BLS	57	110	39	30	48	14
	GLL1	41	140	25	24	56	12
	GLL2	166	37	3	69	19	4
CG iterations	BLS	160	30	16	69	14	9
	GLL1	156	45	5	72	15	5
	GLL2	166	36	4	77	12	3
Time	BLS	196	9	1	72	13	7
	GLL1	197	9	0	74	15	3
	GLL2	199	7	0	80	11	1

Table 2: Rankings for BLS, GLL1 and GLL2 on 206 small and 92 large problems

indicated by the more detailed comparison between these methods presented in Table 3. In this table, we restrict our attention to 215 small and 94 large problems that were coherently solved by both methods, that is for which both methods converged to critical points with identical objective function values. Columns three to five of the tables indicate how many times each method was best and how many ties occured. Columns six to ten describe the distribution of differences in values of the relevant criterion (iterations, function evaluations, CG iterations or time) for the two methods, *when these values are different*: we report the minimum of this distribution (min), its first quartile (q1), its median (med), its third quartile (q3) and its maximum (max).

	Performance measure	GLL1 wins	tie	GLL2 wins	GLL1 - GLL2 (when different) min	q1	med	q3	max
215 small problems	Iterations	10	188	17	-775	-3	1	11	97
	Evaluations	21	27	167	-1561	2	3	4	294
	CG iterations	11	183	21	-3859	-3	12	93	1034
	Time	1	211	3	-1	3	3	4	4
94 large problems	Iterations	7	77	10	-16	-5	1	11	26
	Evaluations	12	14	67	-22	1	2	4	57
	CG iterations	6	76	12	-10676	-23	83	338	3262
	Time	4	79	11	-1131	-2	2	16	100

Table 3: Detailed comparison of GLL1 and GLL2

Table 3 generally favors GLL2: this method indeed wins more often for all criteria and the medians of all difference distributions are positive. However, we observe that these distributions are rather dispersed. In particular, we note the striking difference of 10676 CG iterations in Table 3 for a large problem (`MSQRTBLS` ($n = 1024$)) in favor of GLL1, which results in an advantage of 1131 seconds for GLL1 on the same problem.

We conclude this analysis of the ideas of Grippo, Lampariello and Lucidi presenting a detailed comparison of the best non-monotone method, GLL2, with our basic linesearch algorithm (BLS). This comparison is given in Table 4 for the 206 small and 88 large problems coherently solved by both methods. The format of this table is identical to that of Table 3.

	Performance measure	BLS wins	tie	GLL2 wins	BLS - GLL2 (when different) min	q1	med	q3	max
206 small problems	Iterations	42	132	32	-694	-10	-2	9	499
	Evaluations	32	146	28	-6747	1	3	6	1696
	CG iterations	37	46	123	-53770	-23	4	51	18779
	Time	7	189	10	-151	-1	1	6	597
92 large problems	Iterations	22	59	11	-2633	-23	-7	7	192
	Evaluations	21	11	60	-9744	-1	2	6	514
	CG iterations	13	57	22	-649584	-96	43	3749	101631
	Time	10	64	18	-5250	-3	14	109	11066

Table 4: Detailed comparison of BLS and GLL2

A few conclusions can be drawn from this comparison.

- The conclusions of Grippo, Lampariello and Lucidi are confirmed in that a non-monotone procedure appears to be often better than the basic linesearch method, and sometimes by a quite substantial amount. However, GLL1 and GLL2 occasionally perform quite badly compared to BLS. In particular, this is the case for problem `FMINSURF` ($n = 1024$), which corresponds to the large differences at the minimum of the distributions of BLS - GLL2 in Table 4.

- The dispersed character of most reported performance difference distributions indicates that a large part of the variations between methods, as appearing in Table 1, are explained by their behaviour on a relatively small number of more difficult problems. The sometimes dramatical differences between BLS, GLL1 and GLL2 are caused by the strongly nonlinear nature of these problems. However, the trend is rather clear overall, indicating that one might indeed loose in performance by using a non-monotone method on a very nonlinear problem, but with the potential of winning substantially more.

- It also results from Tables 3 and 4 that performance differences are relatively small for at least half the test problems used, but that these small differences also often favour GLL2, over both GLL1 and BLS.

- At variance with previously reported numerical experience, the results presented here allow to distinguish the relative merits of non-monotone techniques for small and large problems separetely. As expected the potential benefits and losses in CPU time are more important for the large problems, at least on our test set where the cost of computing the objective function and its derivatives is often small or modest compared to that of the involved linear algebra.

  Furthermore, a remarkable effect seems to occur when combining the non-monotone GLL1 and GLL2 methods and the truncated modified CG technique on some large problems. Because of the non-monotonicity, the iterates do not appear to follow the bottom of curved valleys so closely for these two algorithms, but rather "stay higher on the slope". This implies that the average size of g_k is larger for these methods than for BLS, which in turn implies, because of (2.18), that the modified CG can be stopped sooner, which finally results in a smaller number of CG steps on average. As a large proportion of the computational effort for large problems is concentrated in this part of the algorithms, this effect yields a noticeable reduction in the overall CPU time for such problems.

3 A new non-monotone variant

3.1 Motivation and specification

In order to alleviate the difficulties of GLL2 revealed, for instance, by the `FMINSURF` problem, a more careful analysis of the performance of GLL2 was carried out for this case and a few others. It then appeared that the quadratic model implicit in Newton's method is often not well suited to represent the objective function, and that the stepsizes determined by the linesearch procedure are very often much smaller than one. Moreover, this behaviour is accentuated for GLL1 and GLL2 compared with BLS. It is therefore natural to think of controlling the amount of non-monotonicity allowed by a measure indicating how well adapted the truncated Newton step is to the true objective function. It also appeared in the tests that it is often advantageous to enforce monotonicity in the first few iterations. The purpose of this section is to show that these simple ideas can indeed produce additional benefits.

We therefore modified Algorithm GLL2 as follows.

- The equality $r_k = f(x_k)$ is enforced for the first 10 iterates.

- For all subsequent iterations, r_k is defined by

$$r_k = f(x_k) + \left(\prod_{j=1}^{j_k} \alpha_{k(\ell-j)} \right)^{\frac{1}{j_k}} [f_{\ell-j_k} - f(x_k)] \tag{3.1}$$

  where

$$j_k = \arg \max_{j=1,\ldots,p} f_{\ell-j} \tag{3.2}$$

  and $\alpha_{k(\ell-j)}$ is the stepsize corresponding to the $(\ell - j)$-th accepted step.

In this proposal, we have chosen to measure the adequacy of the Newton step by examining how close the past stepsizes α_k are to unity. The amount of non-monotonicity allowed in the method is then controlled by this measure: it is indeed easy to see that r_k as given by (3.1) is equal to the value calculated by Algorithm GLL2 whenever the last j_k accepted stepsizes are equal to one. Because most stepsizes do not exceed one, it is typically smaller in other cases. In our implementation, p is chosen to be 10, as for GLL1 and GLL2. Of course, the convergence theory developed for GLL2 still holds for this variant, which we call Algorithm NMLS (for Non-Monotone LineSearch).

3.2 Numerical comparison

In order to assess this new variant, we also ran it on the set of test problems described above. This algorithm succeeded to solve 222 of the 226 small problems and 97 of the 101 large ones, a slightly better result than GLL2. We also computed values of the cumulative performance statistics for the 206 small and 92 large problems coherently solved by BLS, GLL1, GLL2 and NMLS. The comparison shown in Table 5 indicates a very clear improvement on average for all criteria, compared to GLL1 and GLL2. In particular, the reduction in CG iterations is noticeable.

Method	206 small problems				92 large problems			
	iters	evals	CG-its	time	iters	evals	CG-its	time
BLS	8173	17926	134099	1752	5041	9231	541429	35233
GLL1	8249	22793	153943	1288	7564	17135	966247	22661
GLL2	8089	21637	151501	1300	7504	16795	968919	23586
NMLS	6620	13115	96571	1130	5057	11771	370379	20577

Table 5: Cumulative statistics for BLS, GLL1, GLL2 and NMLS

Examining the rankings presented in Table 6 confirms that NMLS is better than BLS, GLL1 or GLL2 not only on average, but also in a number of instances. This conclusion applies for both small and large problems. One interesting observation is that NMLS seems to require less iterations and less function evaluations than GLL2 on a significant number of problems.

If we now consider the detailed comparisons between NMLS, BLS and GLL2 presented in Tables 7 and 8, the following additional comments can be made.

- The observations made above about the relative importance of more difficult problems in the average performance applies to NMLS as well as to the previously analyzed methods. However, one notices that the cases that are most unfavourable to NMLS compared to BLS (see Table 8) are much less so than those reported for GLL2 (see Table 4). We note in particular, the very remarkable (although extreme) gain of 599569 CG iterations between GLL2 and NMLS on problem `FMINSURF` ($n = 1024$). This is not so surprising since NMLS is in fact designed to overcome the deficiency of GLL2 on that problem and similar ones. This big advantage in CG iterations of course results in a corresponding gain in CPU time.

Performance measure	Method	Small problems 1st	2nd	3rd	4th	Large problems 1st	2nd	3rd	4th
Iterations	BLS	169	17	17	7	76	7	6	3
	GLL1	148	34	24	0	65	9	15	3
	GLL2	154	35	13	4	65	14	10	3
	NMLS	168	31	7	0	70	17	4	1
Evaluations	BLS	44	118	24	20	26	48	9	9
	GLL1	39	124	25	18	21	53	10	8
	GLL2	155	29	19	3	63	20	6	3
	NMLS	166	28	8	4	67	16	8	1
CG iterations	BLS	154	25	15	12	67	9	10	6
	GLL1	150	34	20	2	69	9	12	2
	GLL2	159	31	13	3	71	15	4	2
	NMLS	161	31	14	0	72	13	6	1
Time	BLS	195	9	1	1	69	11	9	3
	GLL1	197	5	4	0	70	13	8	1
	GLL2	199	4	3	0	75	14	2	1
	NMLS	200	6	0	0	76	11	4	1

Table 6: Rankings for BLS, GLL1, GLL2 and NMLS on 206 small and 92 large problems

- The smaller differences in performance (on the easier problems) also appear to favour NMLS over GLL2 and, more clearly, over BLS.
- As expected, the potential gain in CPU time between NMLS and GLL2 is more considerable for large problems and parallels, in this case, the potential gain in CG iterations. The same comment applies, although to a lesser extent, to the comparison between NMLS and BLS.
- We note that it is advantageous to use NMLS rather than BLS or even than GLL2 when the objective function is costly to evaluate.

	Performance measure	GLL2 wins	tie	NMLS wins	GLL2 - NMLS (when different) min	q1	med	q3	max
210 small problems	Iterations	15	158	37	-64	-1	4	13	475
	Evaluations	19	156	35	-189	-6	2	19	4390
	CG iterations	26	152	32	-577	-25	6	86	43688
	Time	4	200	6	-3	-3	1	2	104
95 large problems	Iterations	9	67	19	-29	-4	8	22	2208
	Evaluations	11	68	16	-73	-14	1	37	4738
	CG iterations	13	67	15	-8880	-285	9	281	599569
	Time	10	73	12	-941	-39	1	21	4810

Table 7: Detailed comparison of GLL2 and NMLS

	Performance measure	BLS wins	tie	NMLS wins	BLS - NMLS (when different) min	q1	med	q3	max
213 small problems	Iterations	31	143	39	-243	-4	2	17	499
	Evaluations	24	27	162	-2357	2	3	6	1508
	CG iterations	27	137	49	-10082	-9	12	120	19664
	Time	3	199	11	-46	-3	2	8	635
93 large problems	Iterations	17	61	15	-425	-13	-1	13	190
	Evaluations	18	11	64	-5006	1	2	8	451
	CG iterations	12	60	21	-50015	-70	27	2602	92751
	Time	12	63	18	-440	-11	1	55	10124

Table 8: Detailed comparison of BLS and NMLS

4 Preconditioning

We now examine the impact of preconditioning the iterative solution of the system (2.16) on the relative performance of BLS and NMLS, our best non-monotone method. The preconditioner used is constructed by extracting from H_k its main 11-banded symmetric submatrix, which is then modified, if necessary, to ensure its positive definiteness. This modification is carried out according to the proposal of Schnabel and Eskow (1991). We note that this preconditioner has not been formally analyzed, and thus that its effect is essentially unknown, although often beneficial. It is used in the default variant of trust region based LANCELOT package, although it is applied here in a context different from that of trust-region methods. This difference is quite significant because the typically longer steps determined by the modified conjugate-gradients procedure make the preconditioner's adequacy more crucial. The same preconditioner is also used by Conn *et al.* (1994) in their study of Iterated Subspace Minimization (ISM) methods. The preconditioned methods corresponding to BLS and NMLS are called PBLS and PNMLS, respectively.

A first observation is that preconditioning slightly deteriorates reliability on our set of test problems: PBLS succeeds on 215 of the 226 small problems and PNMLS on 214. Of the 101 large problems, PBLS solves 89 and PNMLS 86. Besides the reasons of failure discussed above for the unpreconditioned case, we note that the search direction computed with the preconditioner sometimes causes the iterates to leave the region where the problem's functions are safely defined, causing arithmetic errors. Another effect is that the test (2.18) is not always adapted to the preconditioned system and may cause very early termination of the modified conjugate-gradients method, yielding a search direction identical or close to steepest descent. When this happens, convergence is typically very slow.

On the other hand, the preconditioner fulfills its purpose on most problems: it reduces the number of CG iterations requested for solution (although it might indeed deteriorate speed due to their increased computational cost). We illustrate this effect by comparing the cumulative performances of the preconditioned algorithms and their unpreconditioned counterparts on the 193 small and 77 large problems that were coherently solved by all four methods. PNMLS appears to be the best of the two preconditioned methods on average for all criteria and all problem sizes.

method	193 small problems				77 large problems			
	iters	evals	CG-its	time	iters	evals	CG-its	time
BLS	8439	19897	130984	1746	4765	8768	511981	32814
NMLS	6851	14450	95688	1126	4729	11278	337616	17826
PBLS	12211	21611	87203	1871	6695	13136	307064	24628
PNMLS	7870	18109	71963	1374	5373	11942	257701	18166

Table 9: Cumulative statistics for BLS, NMLS, PBLS and PNMLS

Table 10 presents the rankings computed for BLS, NMLS, PBLS, and PNMLS. These rankings confirm the comments already made. Again, PNMLS is most often the best of the preconditioned algorithms.

The detailed comparisons between PBLS and PNMLS presented in Table 11 reinforce our previous conclusions on the relative merits of monotone *vs.* non-monotone methods. We may also note that, on this set of test problems, preconditioning does not globally improve the speed of NMLS, which still dominates PNMLS in CPU time. However, it seems to help in reducing the number of CG iterations. We finally note that the gap between monotone and non-monotone methods is smaller when preconditioning is used than in the unpreconditioned case.

Performance	Method	Small problems				Large problems			
measure		1st	2nd	3rd	4th	1st	2nd	3rd	4th
Iterations	BLS	88	77	14	14	28	37	7	5
	NMLS	87	79	25	2	29	38	8	2
	PBLS	106	53	18	16	54	9	6	8
	PNMLS	120	35	27	11	52	13	9	3
Evaluations	BLS	14	82	60	37	5	26	32	14
	NMLS	94	62	30	7	31	24	17	5
	PBLS	29	64	57	43	18	26	16	17
	PNMLS	109	24	38	22	47	11	16	3
CG iterations	BLS	28	119	21	25	6	53	9	9
	NMLS	34	112	38	9	9	55	10	3
	PBLS	129	38	13	13	56	8	7	6
	PNMLS	135	33	22	3	63	8	4	2
Time	BLS	178	10	3	2	40	24	11	2
	NMLS	182	9	2	0	46	23	6	2
	PBLS	162	18	12	1	41	24	8	4
	PNMLS	165	18	8	2	42	29	5	1

Table 10: Rankings for BLS, NMLS, PBLS and PNMLS on 193 small and 77 large problems

5 Conclusion and perspectives

In this paper, we have revisited Grippo, Lampariello and Lucidi's proposals to use non-monotone techniques in the framework of linesearch algorithms for unconstrained optimization. At variance with the original experiences reported by these authors, we have used a (preconditioned) truncated conjugate-gradients technique to compute the search directions and have applied the resulting methods to a large collection of both small and large dimensional test problems, totalizing more than 2200 test runs.

Using the results of these experiments, we have assessed the potential of non-monotone linesearch techniques as proposed by Grippo *et al.*. The effects of problem size and preconditioning

	Performance measure	PBLS wins	tie	PNMLS wins	PBLS - PNMLS (when different) min	q1	med	q3	max
212 small problems	Iterations	35	131	46	-431	-12	2	12	1419
	Evaluations	32	28	152	-4513	1	3	7	2594
	CG iterations	32	127	53	-31999	-28	3	62	15700
	Time	9	191	12	-34	-2	0	2	505
84 large problems	Iterations	13	57	14	-285	-33	1	97	3121
	Evaluations	10	59	15	-3402	1	2	5	6334
	CG iterations	7	56	21	-27248	-2	109	439	64647
	Time	7	59	18	-3329	-2	5	97	7406

Table 11: Detailed comparison of PBLS and PNMLS

have also been specifically pointed out. We finally proposed a new improved variant that uses similar techniques.

The general conclusions of these tests and proposals can be summarized as follows.

- The impact of non-monotone linesearch techniques is clear, and most noticeable on the more difficult and nonlinear problems.
- This impact is often favourable, in that non-monotone methods show superior efficiency when compared to standard linesearch algorithms. This is especially true if preconditioning is not used. However, unfavourable problem dependent effects also appear, but seem to be outweighted by the favourable ones.
- As expected, the differences in CPU time between monotone and non-monotone methods are most important for the large problems. This distinction is however irrelevant when the number of problem function evaluations is considered: the gains provided by some of the analyzed non-monotone methods are consistent, irrespective of problem size.
- The new proposed variant appears to yield the highest potential for substantial efficiency gains while best limiting the probability of losses, at least on the set of test problems used in this comparison.

As is the case for all practical comparisons between optimization algorithms, the results presented here do not pretend to be either complete or decisive. Further confirmation by continued experience is of course necessary for a true validation of non-monotone linesearch techniques, but the trends outlined in this study are definitely encouraging. The application of these techniques in the context of merit function minimization for the solution of constrained problems and in the framework of Iterated Subspace Minimization (see Conn *et al.* (1994)) are of particular interest. Finally, the extension of the non-monotone approach to trust-region algorithm is the object of ongoing research.

6 Acknowledgements

The author is indebted to N. Gould for kindly providing a Fortran 77 code for the modified conjugate-gradient technique and to A. Sartenaer for many useful comments.

References

[Arioli *et al.*, 1993] M. Arioli, T.F. Chan, I.S. Duff, N.I.M. Gould, and J.K. Reid. Computing a search direction for large-scale linearly constrained nonlinear optimization calculations. Technical Report TR/PA/93/34, CERFACS, Toulouse, France, 1993.

[Bongartz *et al.*, 1994] I. Bongartz, A. R. Conn, Nick Gould, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software*, (to appear), 1994.

[Burke *et al.*, 1990] J. V. Burke, J. J. Moré, and G. Toraldo. Convergence properties of trust region methods for linear and convex constraints. *Mathematical Programming, Series A*, 47(3):305–336, 1990.

[Byrd *et al.*, 1992] R. H. Byrd, D.C Liu, and J. Nocedal. On the behaviour of broyden's class of quasi-Newton methods. *SIAM Journal on Optimization*, 2(4):533–557, 1992.

[Chamberlain *et al.*, 1982] R. M. Chamberlain, C. Lemaréchal, H. C. Pedersen, and M. J. D. Powell. The watchdog technique for forcing convergence in algorithms for constrained optimization. *Mathematical Programming Studies*, 16:1–17, 1982.

[Conn *et al.*, 1988] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal on Numerical Analysis*, 25:433–460, 1988. See also same journal 26:764–767, 1989.

[Conn *et al.*, 1992] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT*: a Fortran package for large-scale nonlinear optimization (Release A)*. Number 17 in Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York, 1992.

[Conn *et al.*, 1994] A. R. Conn, Nick Gould, A. Sartenaer, and Ph. L. Toint. On iterated-subspace minimization methods for nonlinear optimization. Technical Report 94/13, Department of Mathematics, FUNDP, Namur, Belgium, 1994.

[Dennis and Schnabel, 1983] J. E. Dennis and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. Prentice-Hall, Englewood Cliffs, USA, 1983.

[Gill and Murray, 1974] P. E. Gill and W. Murray. Newton-type methods for unconstrained and linearly constrained optimization. *Mathematical Programming*, 28:311–350, 1974.

[Gill *et al.*, 1992] P. E. Gill, W. Murray, D. B. Ponceléon, and M. A. Saunders. Preconditioners for indefinite systems arising in optimization. *SIAM Journal on Matrix Analysis and Applications*, 13:292–311, 1992.

[Griewank and Toint, 1982] A. Griewank and Ph. L. Toint. Local convergence analysis for partitioned quasi-Newton updates. *Numerische Mathematik*, 39:429–448, 1982.

[Grippo *et al.*, 1986] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM Journal on Numerical Analysis*, 23(4):707–716, 1986.

[Grippo *et al.*, 1991] L. Grippo, F. Lampariello, and S. Lucidi. A class of nonmonotone stabilization methods in unconstrained optimization. *Numerische Mathematik*, 59:779–805, 1991.

[Kanzow, 1994] Ch. Kanzow. Some tools allowing interior-point methods to become noninterior. Technical Report A-79, Institute of Applied Mathematics, University of Hamburg, D-20146 Hamburg, Germany, 1994.

[Koontz *et al.*, 1985] J.E. Koontz, R.B. Schnabel, and B.E. Weiss. A modular system of algorithms for unconstrained minimization. *ACM Transactions on Mathematical Software*, 11:419–440, 1985. Also available as Technical Report CU-CS-240-82, Department of Computer Science, University of Colorado, Boulder, CO.

[Liu and Nocedal, 1989] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming, Series B*, 45(1):503–528, 1989.

[Ortega and Rheinboldt, 1970] J. M. Ortega and W. C. Rheinboldt. *Iterative solution of nonlinear equations in several variables.* Academic Press, New York, 1970.

[Schlick, 1993] T. Schlick. Modified Cholesky factorizations for sparse preconditioners. *SIAM Journal on Scientific and Statistical Computing*, 14(2):424–445, 1993.

[Schnabel and Chow, 1991] R.B. Schnabel and T.-T. Chow. Tensor methods for unconstrained optimization using second derivatives. *SIAM Journal on Optimization*, 1(3):293–315, 1991.

[Schnabel and Eskow, 1991] R. B. Schnabel and E. Eskow. A new modified Cholesky factorization. *SIAM Journal on Scientific and Statistical Computing*, 11:1136–1158, 1991.

[Toint, 1983] Ph. L. Toint. VE08AD, a routine for partially separable optimization with bounded variables. *Harwell Subroutine Library*, 2, 1983.

[Toint, 1987] Ph. L. Toint. VE10AD, a routine for large scale nonlinear least squares. *Harwell Subroutine Library*, 2, 1987.

[Toint, 1988] Ph. L. Toint. Global convergence of a class of trust region methods for nonconvex minimization in Hilbert space. *IMA Journal of Numerical Analysis*, 8:231–252, 1988.

[Toint, 1994] Ph. L. Toint. An assessment of non-monotone linesearch techniques for unconstrained optimization: the complete numerical results. Technical Report 94/15, Department of Mathematics, FUNDP, Namur, Belgium, 1994.