DOTTORATO DI RICERCA IN
INFORMATICA, SISTEMI E TELECOMUNICAZIONI

CICLO XXVII

COORDINATORE Prof. Chisci Luigi

New classes of adaptive cubic regularization algorithms for unconstrained optimization

Settore Scientifico Disciplinare MAT/09

Dottorando
Dott. Bianconcini Tommaso

Tutori
Prof. Sciandrone Marco

Prof. Schoen Fabio

Coordinatore
Prof. Chisci Luigi

Anni 2012/2014

Contents

1 Introduction **5**

2 Adaptive cubic regularization algorithms **9**
- 2.1 ARC algorithms and convergence properties 9
- 2.2 Worst complexity case bound 13
- 2.3 Parallelism between ARC and Trust Region methods 17

3 An ARC algorithm using iterative methods and early stopping **21**
- 3.1 An iterative method in ARC framework 21
- 3.2 The early stopping criterion 22
- 3.3 The minimization scheme of the cubic model 23
- 3.4 Numerical results 26
  - 3.4.1 Implementation details 26
  - 3.4.2 Numerical results on CUTEr problems 28
  - 3.4.3 Numerical results on Luksan's problems 34
  - 3.4.4 On the influence of Algorithm 2.2 on overall performance 42
  - 3.4.5 General comments on the comparison between ARC-GLRT and ARC-NMGRAD 43

4 A non-monotone ARC algorithm **45**
- 4.1 Non-monotone method sketch 45
- 4.2 Convergence analysis 49
- 4.3 Numerical results 56
  - 4.3.1 Implementation details 56
  - 4.3.2 Numerical results with solver GLRT 56
  - 4.3.3 Numerical results with solver NMGRAD 62
  - 4.3.4 General comments and comparison between solvers GLRT and NMGRAD 67

Chapter 1

Introduction

Adaptive regularized methods have been recently studied as an alternative to classical globalization techniques for nonlinear constrained and unconstrained optimization [1–3, 7–9, 15, 16, 20, 21, 23]. This thesis is devoted to the numerical solution of the unconstrained optimization problem

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

where $f : \mathbb{R}^n \to \mathbb{R}$ is a smooth function, by means of the ARC framework [8, 9, 16, 21, 23].

The adaptive cubic regularization of Newton method for (1.1) gives rise to a local cubic overestimator of the objective function f which is employed for computing the step from one iterate to the next one. Under mild assumptions, ARC iterates converge to first-order critical points; by strengthening the conditions on the acceptance of the trial step, second-order variants of ARC show global and fast convergence to second-order critical points [9, 16, 21].

Besides the good numerical performance of ARC compared to a standard trust-region approach [9], its distinguishing features from linesearch and trust-region techniques are the results on worst-case iteration and gradient evaluation complexity [8, 21]. Specifically, a worst-case iteration count of order $\epsilon^{-3/2}$ has been established to drive the norm of the gradient of f below a prefixed accuracy ϵ. This bound is sharp and represents a substantial improvement over the Newton's method which may be as slow as the steepest descent method (in the worst case) and require a number of iterations of order ϵ^{-2} [6].

The good complexity bound of ARC can be achieved by minimizing the cubic model approximately within some suitable accuracy requirement and under conditions that can all be ensured if the step taken at each iteration is a global minimizer of the model in a subspace of $\mathbb{R}^n$ [8]. The second-order variant of ARC, denoted as $\mathrm{ARC}_{(\mathrm{S})}$ in [8], algorithmically ensures the conditions required and one possible implementation relies on linear algebra. In fact, in ARC$_{(S)}$ the approximate minimization of the cubic model can be performed over evolving subspaces by using the Lanczos method until a suitable termination criterion is met.

Despite the good complexity properties, the practical efficiency of ARC depends on the way the cubic model is minimized. Therefore the main objective of this thesis is to investigate how to preserve the complexity properties of the ARC$_{(S)}$ framework when procedures alternative to the Lanczos method are used for minimizing the cubic model.
In particular the interest is focused on iterative descent methods, such as gradient methods, limited memory Quasi-Newton methods, conjugates gradients methods, which are matrix-free and either reduce or avoid the cost of the linear algebra phase.

Following the approach described in [4] , a strategy that fits into ARC$_{(S)}$ framework is introduced, thus retaining its worst-case complexity count. This can be done with any arbitrarily computed approximate minimizer for the mode.l To achieve this goal the approximate minimization of the model is combined, if necessary, with the exact optimization of the model in a suitable one-dimensional space. Further, is presented a new termination criterion (called early stopping) which monitors the value of f along the minimization of the model and can prevent an "over-solving" phenomenon when the objective function is not adequately represented by the cubic model. Also this stopping criterion has been introduced in [4].

The resulting variant of ARC has been extensively tested using a nonmonotone gradient method as iterative solver for step computation. This proposal showed to be a viable alternative to the implementation of ARC using the GLRT routine, available in the GALAHAD library [14], for step computation.

This thesis also investigates the use of nonmonotone techniques within the ARC framework. A nonmonotone variant of the ARC algorithm is introduced, its convergence properties are discussed and some numerical results on the comparison against the monotone version are presented.

This thesis is organized as follows. Chapter 2 reviews the ARC framework, investigates the worst-case complexity bounds for ARC and compares ARC methods with TR methods. In Chapter 3 we discuss the minimization of the cubic model by iterative descent methods and their use in conjunction with the early stopping criterion. The new variant of ARC presented in [4] that preserves the good iteration complexity is discussed. In Chapter 4 we present a non monotone variant of ARC algorithm, which aims to improve the performances of the monotone version.

Notations. The gradient $\nabla_x f(x)$ of f and the Hessian $\nabla_{xx} f(x)$ of f are denoted by $g(x)$ and $H(x)$ respectively. The 2-norm is denoted by $\|x\|$. The identity matrix of appropriate dimension is indicated by I.

Chapter 2

Adaptive cubic regularization algorithms

2.1 ARC algorithms and convergence properties

Adaptive cubic regularization of Newton method for unconstrained optimization gives rise to globally convergent procedures recently investigated in many papers, see e.g. [8,9,16,21]. The key feature of this approach is the computation of the step from one iterate to the next by minimizing a cubic overestimator of the objective function f.

If the Hessian H of f is Lipschitz continuous (with constant $2L$), the Taylor expansion of f around x_k gives

$$\begin{aligned} f(x_k+p) &= f(x_k)+p^T g(x_k)+\frac{1}{2}p^T H(x_k)p+ \\ &\quad +\int_0^1 (1-\tau)p^T(H(x_k+\tau p)-H(x_k))p\,d\tau \\ &\leq f(x_k)+p^T g(x_k)+\frac{1}{2}p^T H(x_k)p+\frac{1}{3}L||p||^3 \overset{\text{def}}{=} m_k^C(p), \end{aligned}$$

for all $p \in \mathbb{R}^n$. Thus, for every step p such that $m_k^C(p) \leq m_k^C(0) = f(x_k)$, the point x_k+p improves f.

In order to define a model of practical interest, the constant L may be replaced by a dynamic positive parameter σ_k and $H(x_k)$ may be approximated by a symmetric matrix B_k. This gives rise to the model

$$m_k(p) = f(x_k)+p^T g(x_k)+\frac{1}{2}p^T B_k p+\frac{1}{3}\sigma_k\|p\|^3, \tag{2.1}$$

which is employed in the ARC algorithm proposed by Cartis et al. in [8,9].

The k-th iteration of the ARC method is sketched in Algorithm 2.1.

```
Algorithm 2.1: k-th iteration of ARC
Given $x_k$ and the scalars $\sigma_k > 0$, $1 > \eta_2 \geq \eta_1 > 0$, $\gamma_2 \geq \gamma_1 > 1$.

1. Compute an approximate minimizer $p_k$ of $m_k$ such that
       $m_k(p_k) \leq m_k(p_k^c),$                                   (2.2)
   where $p_k^c$ is the Cauchy point
       $p_k^c = -\alpha_k g(x_k), \qquad \alpha_k = \operatorname*{argmin}_{\alpha \geq 0} m_k(-\alpha g(x_k)).$   (2.3)

2. Compute
       $\rho_k = \dfrac{f(x_k) - f(x_k + p_k)}{f(x_k) - m_k(p_k)}.$      (2.4)

3. Set
       $x_{k+1} = \begin{cases} x_k + p_k & \text{if } \rho_k \geq \eta_1, \\ x_k & \text{otherwise.} \end{cases}$

4. Set
       $\sigma_{k+1} \in \begin{cases} (0, \sigma_k] & \text{if } \rho_k \geq \eta_2 & \text{(very successful iteration),} \\ [\sigma_k, \gamma_1 \sigma_k) & \text{if } \eta_1 \leq \rho_k \leq \eta_2 & \text{(successful iteration),} \\ [\gamma_1 \sigma_k, \gamma_2 \sigma_k] & \text{otherwise} & \text{(unsuccessful iteration).} \end{cases}$
```

In Step 1 the trial step p_k is computed as an approximate minimizer of the model m_k guaranteeing a decrease in m_k greater than or equal to the reduction attained by the Cauchy point (2.3). Then, in Step 2 the ratio ρ_k is computed and in Step 3 p_k is accepted, and the new iterate x_{k+1} is set to $x_k + p_k$, if a sufficient decrease in the objective is achieved; otherwise, the step is rejected and x_{k+1} is set to x_k. Since the denominator in (2.4) is strictly positive whenever the current iterate is not a first-order critical point, then ARC algorithm is well defined and the generated sequence $\{f(x_k)\}$ is monotonically non-increasing. The rules in Step 4 for updating the parameter σ_k take into account the agreement between f and m_k and parallel those for updating the trust-region radius in trust-region methods [22].

Condition (2.2) on p_k imposes at least as much decrease in the model as that obtained by the Cauchy point p_k^c. A lower bound on the decrease achieved by p_k^c with respect to $m_k(0) = f(x_k)$ is given below.

Lemma 2.1. [9, Lemma 2.1] *Suppose that the step p_k satisfies (2.2). Then for $k \geq 0$, we have that*

$$f(x_k) - m_k(p_k) \geq f(x_k) - m_k(p_k^c) \geq \frac{\|g(x_k)\|}{6\sqrt{2}} \min\left(\frac{\|g(x_k)\|}{1+\|B_k\|}, \frac{1}{2}\sqrt{\frac{\|g(x_k)\|}{\sigma_k}}\right) \tag{2.5}$$

By imposing (2.2), global convergence to stationary points of problem (1.1) can be enforced as stated by the proposition below.

Proposition 2.1. [9, Corollary 2.6] *Let $f \in C^1(\mathbb{R}^n)$ and $\{x_k\}$ be the sequence generated by the ARC Algorithm. Suppose that $\|B_k\|$ is uniformly bounded for all $k \geq 0$ and that the gradient g is uniformly continuous on the sequence $\{x_k\}$. Then,*

$$\lim_{k\to\infty} \|g(x_k)\| = 0. \tag{2.6}$$

In order to enforce local fast convergence, more model reduction than (2.2) is sought and this requires to approximately minimize m_k. We now discuss the stationary points and values of m_k analyzed in [9, 16]. Any stationary point $\hat{p}$ of m_k satisfies

$$(B_k + \hat{\lambda} I)\hat{p} = -g(x_k), \tag{2.7}$$

$$\hat{\lambda} = \sigma_k \|\hat{p}\|. \tag{2.8}$$

A stationary point p_k^* is a global minimizer of m_k over $\mathbb{R}^n$ if and only if there exists a positive scalar λ_k^* such that the pair (p_k^*, λ_k^*) satisfies (2.7) and (2.8) and $B_k + \lambda_k^* I$ is positive semidefinite. Clearly, if $B_k + \lambda_k^* I$ is positive definite then p_k^* is unique.

The stationary values $m_k(\hat{p})$ are strictly decreasing in $\|\hat{p}\|$ and they are up to $2\ell + 1$ values if B_k has ℓ negative eigenvalues. If B_k is indefinite then the stationary values include one global minimum and possibly a second local minimum.

When the model is convex, equations (2.7) and (2.8) characterize any global minimizer. This occurrence is verified for k sufficiently large when the iterates converge to a point with positive definite Hessian and the approximate Hessian B_k becomes positive definite asymptotically. Another occurrence is when B_k is a ℓ-BFGS Hessian approximation. Finally, a convex model can be obtained from the application of ARC method to nonlinear least-squares problems [2, 15]. In this case $f(x) = \|F(x)\|^2$ for some vector-valued function F and the model used is a variant of (2.1) of practical interest for computing zero or small-residual solutions of the problem. Specifically, it consists of the Gauss-Newton model regularized by a cubic term

$$m_k(p) = \frac{1}{2}\|F(x_k) + J(x_k)p\|^2 + \frac{1}{3}\sigma_k \|p\|^3, \tag{2.9}$$

where J is the Jacobian matrix of F. For any positive σ_k the model (2.9) is strictly convex.

The use of an approximate global minimizer is considered in [9] and a viable strategy for its approximate computation is derived using (2.7) and (2.8). We briefly review the main issues for approximating the optimal pair (p_k^*, λ_k^*). Let $p(\lambda)$ solve

$$(B_k + \lambda I)p(\lambda) = -g(x_k), \tag{2.10}$$

λ_k be an approximation to the optimal value λ_k^*, and $p_k = p(\lambda_k)$. The scalar λ_k can be obtained applying a root-finding solver to the so-called secular equation, i.e. the scalar nonlinear equation

$$\lambda - \sigma_k \|p(\lambda)\| = 0,$$

which can be reformulated as

$$\psi(\lambda) = \frac{1}{\|p(\lambda)\|} - \frac{\sigma_k}{\lambda} = 0. \tag{2.11}$$

Letting $\lambda_{min}(B_k)$ be the smallest eigenvalue of B_k and $\zeta = \max\{0, -\lambda_{min}(B_k)\}$, the function $\psi(\lambda)$ is concave and strictly increasing when $\lambda > \zeta$. Hence, either the Newton or the secant method applied to (2.11) converges globally and monotonically to the positive root λ_k^* for any initial guess in the open interval (ζ, λ_k^*) [9, Theorem 6.3]. Clearly, the application of these methods requires the solution of system (2.10) for various λ and the use of a Krylov method represents a relevant alternative in this respect.

Using the Lanczos method m_k can be minimized over evolving subspaces of $\mathbb{R}^n$. Specifically, the Lanczos method can be used to build an orthogonal basis $\{q_1, \dots, q_j\}$ for the Krylov space $\mathcal{K}_j = \{g(x_k), B_k g(x_k), \dots, B_k^{j-1} g(x_k)\}$. Then, letting $Q_j \in \mathbb{R}^{n \times j}$ be the matrix $Q_j = (q_1, \dots, q_j)$, the minimizer $p_{k,j}$ of m_k over $\mathcal{K}_j$ is the vector

$$p_{k,j} = Q_j y_j \quad \text{such that} \quad y_j = \operatorname*{argmin}_{y \in \mathbb{R}^j} m_k(Q_j y). \tag{2.12}$$

Solving the problem on each expanding subspace $\mathcal{K}_j$ is computationally convenient and the minimization process is carried out on evolving subspaces until a specified accuracy requirement is met. In particular, the procedure is repeated until a vector y_{j^*} satisfying

$$\|\nabla m_k(p_{k,j^*})\| = \|\nabla m_k(Q_{j^*} y_{j^*})\| \le \eta_k \|\nabla m_k(0)\|, \tag{2.13}$$

is computed for a given $\eta_k \in [0, 1)$.
Once y_{j^*} is computed, the approximate minimizer p_{k,j^*} can be evaluated either by recomputing the vectors q_j, $1 \le j \le j^*$, or by recovering them from memory. It has been found advantageous to store a small number t of the first q_j, $1 \le j \le t$, vectors and to start from $j = t$, if necessary, the so-called *second-pass*

iteration to determine p_{k,j^*}. Further economies can be made recording all the generated values $m_k(p_{k,j})$, $1 \le j \le j^*$, picking an iteration $h \le j^*$ which gives a specified fraction of the best value obtained, and then accepting $Q_h y_h$ as the required approximation [9, 12].

Under suitable assumptions, the sequence $\{x_k\}$ generated by the ARC algorithm shows superlinear or quadratic convergence rate if the steps satisfy (2.13) and $\eta_k \to 0$ as $k \to \infty$ [9, Corollary 4.8, 4.10]. For instance, superlinear convergence rate can be ensured provided that

$$\|\nabla m_k(p_k)\| \le \min\{\theta, \|g(x_k)\|^{1/2}\}\|g(x_k)\|, \tag{2.14}$$

and quadratic convergence rate can be achieved if

$$\|\nabla m_k(p_k)\| \le \min\{\theta, \|p_k\|\}\|g(x_k)\|, \tag{2.15}$$

with $\theta > 0$.

2.2 Worst complexity case bound

In this chapter, following [8], we preliminary recall the necessary ingredients to obtain the complexity bound in ARC. Then, we present the new procedure explained in [4] that uses a simple steepest descent method with backtracking for computing the step and that can be employed in connection with ARC framework attaining the same complexity bound. The procedure is matrix-free and does not require to store vectors or to recover them from memory, as in a Krylov method, whose number can be, in principle, equal to the dimension of the problem.

Global convergence properties and worst case complexity of the ARC algorithm have been established in [8]. The worst case complexity bound of order $\epsilon^{-3/2}$ was shown to be sharp and represents a substantial improvement over the Newton's method which may be as slow as the steepest descent method (in the worst case) and requires a number of iterations of order ϵ^{-2} [6].

The complexity analysis is based on the fact that, for the ARC algorithm, it is possible to bound the cardinality of any subset of successful iteration indices provided that, at the iterates of the subsequence, the step p_k yields a sufficient predicted reduction. This is shown in the following proposition.

Proposition 2.2. *[8, Theorem 2.2] Let $\{f(x_k)\}$ be bounded from below and K_s an index set of successful iterates generated by* ARC *algorithm defined as*

$$K_s = \{k \ge 0 : k \textit{ successful or very successful in Algorithm 2.1}, \ \|g(x_k)\| \ge \epsilon\},$$

for some positive ϵ. Assume that

$$f(x_k) - m_k(p_k) \ge \rho\epsilon^{3/2} \qquad \forall k \in K_s, \tag{2.16}$$

where ρ is a positive constant independent of k. Then, the cardinality $|K_s|$ of K_s satisfies

$$|K_s| \leq C\epsilon^{-3/2},$$

for some $C > 0$.

The key point in Proposition 2.2 is condition (2.16) which can be accomplished, for instance, by requiring that, for all $k \geq 0$, p_k satisfies (2.15) and

$$g(x_k)^T p_k + p_k^T B_k p_k + \sigma_k \|p_k\|^3 = 0, \tag{2.17}$$
$$p_k^T B_k p_k + \sigma_k \|p_k\|^3 \geq 0. \tag{2.18}$$

The variant ARC$_{(\mathrm{S})}$ introduced in [8, Algorithm 4.1] for all $k \geq 0$ uses a step such that (2.15), (2.17) and (2.18) are met and (2.2) remains satisfied.
A situation in which such conditions hold is when the step p_k is computed by approximately minimizing m_k over nested subspaces as in (2.12) and termination criterion (2.15) is adopted. This feature is a consequence of the following result.

Lemma 2.2. *[8, Lemma 4.1] Suppose that p_k is the global minimizer of $m_k(p)$, for $p \in \mathcal{L}_k$, where $\mathcal{L}_k$ is a subspace of $\mathbb{R}^n$. Then p_k satisfies* (2.17) *and* (2.18).

On the other hand, if the step p_k is computed by a procedure other than the minimization of m_k over evolving subspaces, conditions (2.17) and (2.18) may not hold thus making the analysis in [8] useless and, possibly, loosing the complexity property of the ARC algorithm. For this reason, in the following Algorithm 2.2 we introduce a strategy that produces an approximate minimizer for m_k satisfying conditions (2.2), (2.15), (2.17) and (2.18). Thus, condition (2.16) holds and the properties in terms of worst case complexity bound are ensured.

```
Algorithm 2.2: A variant of Step 1 of ARC.
At each iteration of ARC algorithm, perform Step 1 as follows

  a. Compute an approximate minimizer $p_{k,0}$ of $m_k(p)$, such that
         $m_k(p_{k,0}) \leq m_k(p_k^c)$.        (2.19)
  b. Set $d_{k,0} = p_{k,0}$.
  c. For $j = 0, 1, \ldots$
      c.1 Compute
             $p_{k,j+1} = \beta d_{k,j}, \qquad \beta = \operatorname{argmin}_{\beta \in R} m_k(\beta d_{k,j})$.
      c.2 If $p_{k,j+1}$ satisfies (2.15)
              set $p_k = p_{k,j+1}$ and stop.
          Else
              apply a backtracking linesearch and find
                 $z_{k,j+1} = p_{k,j+1} - \zeta_{k,j+1} \nabla m_k(p_{k,j+1})$,        (2.20)
              such that
                 $m_k(z_{k,j+1}) \leq m_k(p_{k,j+1}) - \mu \zeta_{k,j+1} \|\nabla m_k(p_{k,j+1})\|^2$.
      c.3 Set $d_{k,j+1} = z_{k,j+1}$, $j = j + 1$.
```

The for-loop at Step c of Algorithm 2.2 alternates between an exact minimization of m_k over a one-dimensional subspace (Step c.1), and an inexact minimization of m_k (Step c.2). This latter minimization is performed starting from the point generated at Step c.1 and applying an Armijo-type line search along the steepest descent direction. The point thus obtained defines the vector that will be used, at the next inner iteration (in Step c.1), to perform the exact minimization over the corresponding one-dimensional subspace.

Note that the point $p_{k,j+1}$ produced at Step c.1 is the global minimizer of m_k over a one-dimensional subspace and, by Lemma 2.2, it satisfies (2.17) and (2.18). Moreover the descent properties of the scheme imply that condition (2.2) holds. Taking into account that m_k is coercive and that an Armijo-type line search is performed, we can show that Step c terminates in a finite number of iterations with a point $p_{k,j+1}$ satisfying the accuracy requirement (2.15). All these properties allow us to prove that condition (2.16) holds. Formally, we can state the next result.

Proposition 2.3. *Assume $f \in C^2(\mathbb{R}^n)$, $g(x_k) \neq 0$. Then, Algorithm 2.2 terminates in a finite number of iterations producing a step p_k satisfying conditions (2.15), (2.17) and (2.18).*

Proof. Let J be the set of iterations j executed at Step c of Algorithm 2.2. We first show that J is finite, that is the test at Step c.2 is satisfied in a finite number of iterations. By contradiction, let us suppose that the test at Step c.2 is never satisfied so that the set J of iterations j of Algorithm 2.2 is infinite. Since m_k is coercive, the properties of the backtracking procedure ensure that

$$\lim_{j\to\infty} \|\nabla m_k(p_{k,j+1})\| = 0. \tag{2.21}$$

Considering the gradient of the model function $m_k(p)$,

$$\nabla m_k(p) = g(x_k) + B_k p + \sigma_k \|p\| p,$$

and $g(x_k) \neq 0$, from (2.21) we get that $\|p_{k,j+1}\| \geq \eta > 0$ for all j. Hence, we can write $\min\{\theta, \|p_{k,j+1}\|\}\|g(x_k)\| \geq \min\{\theta, \eta\}\|g(x_k)\|$ which is a constant. Thus, using again (2.21), it follows that, for j sufficiently large

$$\|\nabla m_k(p_{k,j+1})\| \leq \min\{\theta, \eta\}\|g(x_k)\| \leq \min\{\theta, \|p_{k,j+1}\|\}\|g(x_k)\|,$$

that is, $p_{k,j+1}$ satisfies condition (2.15). Then, Algorithm 2.2 would stop at Step c.2 thus contradicting the assumption that J is infinite.
Now, let $\bar{j}$ be the iteration such that $p_{k,\bar{j}+1}$ satisfies (2.15). Then, by the instruction of the algorithm, we also have that p_k satisfies (2.15).
Furthermore, since $p_{k,j+1}$, for all $j \geq 0$ is the global minimizer of m_k over the subspace generated by $d_{k,j}$, Lemma 2.2 implies that $p_{k,j+1}$, and in particular p_k, satisfies (2.17) and (2.18), which concludes the proof. □

Now we can prove the main result of this Chapter, namely that, at every successful iteration k, condition (2.16) holds.

Proposition 2.4. *Let*

(i) $f \in C^2(\mathbb{R}^n)$;

(ii) $\|g(x) - g(y)\| \leq \mathcal{K}\|x - y\|$, *for all* $x, y \in X$ *an open convex set containing the iterates and* $\mathcal{K} \geq 1$;

(iii) $\|H(x) - H(x_k)\| \leq L\|x - x_k\|$, *for all* $x \in [x_k, x_k + p_k]$ *and all* $k \geq 0$;

(iv) $\|(H(x_k) - B_k)p_k\| \leq C\|p_k\|^2$, *for all* $k \geq 0$ *and some positive constant* C;

(v) $\sigma_k \geq \sigma_{min}$ *for all* $k \geq 0$ *and some positive* σ_{min}.

Then, Algorithm 2.2 is such that condition (2.16) *holds at every successful iteration k of ARC satisfying*

$$\min\{\|g(x_k)\|,\ \|g(x_{k+1})\|\} > \epsilon.$$

Proof. Since Algorithm 2.2 ensures that (2.17) and (2.18) hold, from [8, Lemma 4.2] it follows that

$$f(x_k) - m_k(p_k) \geq \frac{1}{6}\sigma_k \|p_k\|^3. \tag{2.22}$$

Then, by [8, Lemma 5.2] and considering that (2.15) is satisfied, the step p_k is such that, for all successful iterations k and some positive ν,

$$\|p_k\| \geq \nu\sqrt{\|g(x_{k+1})\|}.$$

The above lower bound along with (2.22) and $\min\{\|g(x_k)\|,\ \|g(x_{k+1})\|\} > \epsilon$ implies

$$f(x_k) - m_k(p_k) \geq \frac{1}{6}\nu\sigma_{min}\epsilon^{3/2},$$

which concludes the proof. □

2.3 Parallelism between ARC and Trust Region methods

In this Section we investigate main analogies and main difference of a typical ARC algorithm respect to another kind of regularization algorithm less recent than ARC but still studied: the Trust Region framework.

Suppose we are given a scalar $\Delta_k > 0$, a sufficiently smooth objective function $f : \mathbb{R}^n \to \mathbb{R}$ and a current point $x_k \in \mathbb{R}^n$. we denote with

$$\mathcal{B}_k = \{x \in \mathbb{R}^n \ : \ \|x\| \leq \Delta_k\}$$

the ball centered in the origin of radius Δ_k and with

$$m_k^{TR}(p) = g_k^T p + \frac{1}{2}p^T B_k p$$

the quadratic model associated to f, where the notation is the same as previous Chapters.
This is not the only possible choice for the model m_k: the only properties that m_k^{TR} is required to satisfy are

$$\begin{cases} m_k^{TR}(0) = f(x_k) \\ \nabla m_k^{TR}(0) = g_k \end{cases}$$

but for our purpose we can simply think to m_k as the previous presented quadratic model. In Algorithm 2.3 we sketch the basic procedure of a classical TR algorithm.

```
Algorithm 2.3: Basic TR algorithm
Given an initial point $x_0$, an initial Trust Region radius $\Delta_0$ and the scalars
$1 > \eta_2 \geq \eta_1 > 0$, $1 > \gamma_2 \geq \gamma_1 > 0$.
Set $k = 0$.

  1. Define a model $m_k^{TR}$ in $\mathcal{B}_k$.
  2. Compute a trial step $p_k \in \mathcal{B}_k$ that "sufficiently" reduces $m_k^{TR}$.
  3. Compute the ratio
        $$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{f(x_k) - m_k^{TR}(p_k)}.$$
     If $\rho_k \geq \eta_1$ then define $x_{k+1} = x_k + p_k$.
     Otherwise define $x_{k+1} = x_k$.
  4. Set
        $$\Delta_{k+1} \in \begin{cases} [\Delta_k, \infty) & \text{if } \rho_k \geq \eta_2 & \text{(very successful iteration),} \\ [\gamma_2\Delta_k, \Delta_k] & \text{if } \eta_1 \leq \rho_k \leq \eta_2 & \text{(successful iteration),} \\ [\gamma_1\Delta_k, \gamma_2\Delta_k] & \text{otherwise} & \text{(unsuccessful iteration).} \end{cases}$$
     Set $k = k + 1$ and go back to Step 1.
```

Under suitable assumptions, very similar to the assumptions used for ARC convergence theory, the same convergence rates as ARC can be proved for TR. For more details on TR convergence, we remand the interested reader to [10].

It's immediate to see that the sketchs of both Algorithms 2.1 and 2.3 are very similar: a trial step is calculated solving a regularized subproblem and the model used is adaptively modified according to the ratio between the reduction observed on the real objective function and the reduction observed on the model. The rule used to adapt the parameters σ and Δ is the same if we think to $\sigma \sim \frac{1}{\Delta}$.

Both models have the property

$$\begin{cases} m_k(0) = f(x_k) \\ \nabla m_k(0) = g_k \end{cases}$$

but the two subproblems have a great difference: the subproblem that must be solved in ARC framework is unconstrained, while the subproblem of TR methods is a constrained optimization problem. So, in principle, the methods used for solving the subproblem may be different. Indeed this consideration is not valid for linear algebra methods: in fact the approximate minimizers of both subproblems can be found using similar Lanczos methods over evolving Krylov subspaces. A detailed description of such a method for TR framework can be found in [10].

Another difference between the two methods is of course the degree used for the regularization: the model used in Algorithm 2.3 is a polynomial of degree 2, while in Algorithm 2.1 is a polynomial of degree 3. But even in this case the difference is only ostensible. In fact the cubic term of m_k is a penalty as bigger as greater is the norm of the trial step found. In a certain way this penalty can be seen as a relaxation of the constraint imposed on the TR model to be solved within a ball of radius Δ. In order to better explain this analogy, let's suppose that at a fixed iteration k the minimizer p_k^* of the cubic model m_k is known to have 2-norm Δ. So we have that

$$m_k(p_k^*) = f(x_k) + {p_k^*}^T g(x_k) + \frac{1}{2}{p_k^*}^T B_k p_k^* + \frac{1}{3}\sigma_k \Delta^3.$$

Hence

$$m(p_k^*) = \min_{p\in\mathbb{R}^n} m_k(p) = \min_{\Delta\in\mathbb{R}^+} q_k(\Delta) + \frac{1}{3}\sigma_k\Delta^3,$$

where

$$q_k(\Delta) = \min_{\|p\|\leq\Delta} m_k^{TR}(p).$$

The main difference between the two different frameworks of regularization is the worst case complexity iterations bound: while, as showed in the previous Section, ARC algorithm requires at most $\mathcal{O}(\epsilon^{-\frac{3}{2}})$ to attain $\|g(x_k) < \epsilon$, TR algorithms requires $\mathcal{O}(\epsilon^{-2})$ and this bound is known to be strict.
This peculiarity of ARC algorithms respect to TR algorithms, make cubic regularization a very interesting field to be investigated.

Chapter 3

An ARC algorithm using iterative methods and early stopping

3.1 An iterative method in ARC framework

As discussed in the preceding chapter, a key issue of ARC algorithm concerns the computation of the trial step p_k as a suitable (approximate) unconstrained minimizer of the cubic model at each iteration. Therefore following [4] we focus on the approximate minimization of m_k and on the employment of a stopping criterion, the "early stopping", which advantageously combine the model and the "true" objective function f. In this section, we focus on the employment of gradient methods to reduce the computational cost of linear algebra operations required by Krylov-type algorithms.

From a theoretical point of view we may observe that:

- in order to guarantee global convergence, it is sufficient that the trial step is such that
$$m_k(p_k) \leq m_k(p_k^c),$$
where p_k^c is the Cauchy point;
- a complexity bound is guaranteed to hold, provided that condition (2.16) holds.

Hence, as regards the convergence properties, any globally convergent iterative method, employing at the first iteration an exact line search along the steepest descent direction, can be employed to solve the problem

$$\min_p \ m_k(p) = f(x_k) + g(x_k)^T p + \frac{1}{2} p^T B_k p + \frac{1}{3}\sigma_k \|p\|^3, \tag{3.1}$$

and compute the step p_k.
Concerning the stopping criteria, condition (2.14) can be used to solve (3.1) while condition (2.15) has to be imposed within Algorithm 2.2 to guarantee the worst-case complexity property. In the following we present a further stopping criterion that takes into account the objective function f during the minimization process of the model.

3.2 The early stopping criterion

As said before, in order to suitably combine the cubic model and the objective function f, we adopt a new stopping criterion, that we call *early stopping*. This kind of stopping criterion is a technique widely adopted in machine learning (see, e.g., [5]). When a machine learning model is trained, a descent algorithm is applied to minimize the error on the training data. In order to prevent the overfitting phenomenon (which happens when the model over-learns the training data and the generalization capability deteriorates), the performance of the model in terms of generalization capability (which is the true objective of the learning) are periodically evaluated during the optimization process using a different data set, the so-called validation set. The optimization is stopped when the error on the validation set starts to increase.

By drawing inspiration from the above early stopping strategy, during the minimization process of the cubic model m_k, the true objective function f is periodically evaluated (say every N iterations of the adopted iterative method) and the minimization method is stopped when f starts to increase.

As an example, with reference to the CUTEr test function CHAINWOO, Figure 3.1 shows the behavior of both $m_k(p(j))$ and $f(x_k + p(j))$ as functions of the inner iteration counter j, where x_k is the current iterate and $\{p(j)\}$ denotes the sequence generated by a descent method applied to the minimization of the model function $m_k(p)$. As it can be seen, after a few inner iterations, there is no agreement between the cubic model and the true objective function, so that the algorithm could take advantage of the early stopping condition just described.

Formally, we denote by $\{p(j)\}$ the sequence generated by an iterative method applied to minimize $m_k(p)$, where $p(0)$ is set equal to the Cauchy point p_k^c.
We propose to terminate the method whenever either criterion (2.14) is satisfied, or

$$f(x_k + p(j)) \geq f(x_k + p(j-N)), \qquad \text{with} \quad \mathrm{mod}\,(j, N) = 0, \tag{3.2}$$

for some $N \in \mathbb{N}, N \geq 1$ and to consider respectively $p(j)$ or $p(j-N)$ as the new trial step.
Specifically, in order to guarantee the bound on the global worst-case iteration complexity, a further test (related to condition (2.16)) on the reduction of the

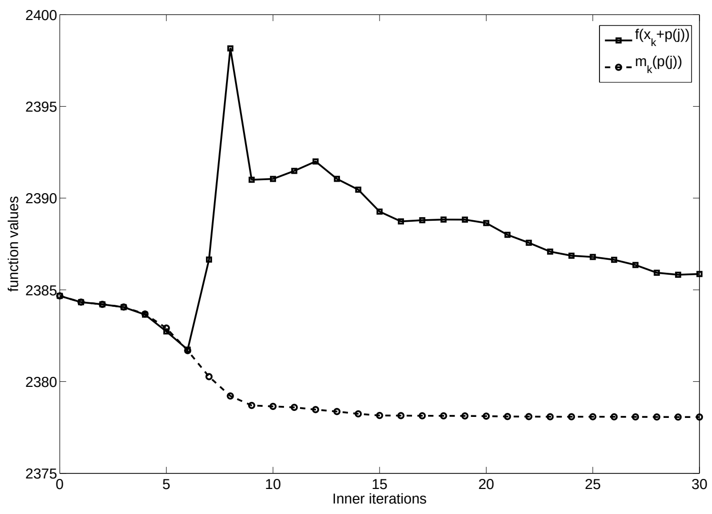

Figure 3.1: An example of a situation in which early stopping may be useful.

model must be considered. When this test is satisfied, the temptative step can be accepted as p_k, otherwise Algorithm 2.2 must be applied to compute p_k.

3.3 The minimization scheme of the cubic model

We observe that, in principle, the early stopping criterion could be used in connection with the Krylov-type algorithms described in Chapter 2. In this case the adoption of the early stopping, which requires the periodic evaluation of the true objective function, involves some critical issues that may drastically influence the computational cost. In fact the minimization in nested Krylov subspaces requires to store the elements of the basis of the subspace, which will be used to reconstruct the trial step p_k. Since in general this operation may require a large amount of memory, it is a common practice to keep in memory only the first few elements of the basis. In this case, the so-called *second-pass iteration* (see Chapter 2) is required to rebuilt the approximate minimizer $p_{k,j}$, which is used for computing the objective function value in the updated point, i.e., $f(x_k + p_{k,j})$. Such a second-pass iteration may determine a drastic increase of the computational cost.

Therefore, we focus our attention on iterative descent methods that:

(i) "directly" generate the approximate minimizers $p(j)$ of the cubic model, so that the evaluation of the true objective function can be performed without additional costs or the need of memory requirements;

(ii) are matrix-free methods (as, for instance, gradient methods, limited memory Quasi-Newton methods, conjugate gradient methods) in order to reduce the computational cost of linear algebra operations.

On these bases, in Algorithm 3.1 we report the conceptual scheme of an iterative method, using the analyzed stopping criteria, for the minimization of the model $m_k(p)$. Then, for the sake of completeness, in Algorithm 3.2 we report the k-iteration of the proposed version of ARC.

Algorithm 3.1: Sketch of an iterative method

```
Given $p(0) = p_k^c$, $\epsilon > 0$ and integers $N \geq 1$, $j_{max} \in [1, +\infty]$. Set $j = 0$.
While (2.14) and (3.2) are not satisfied and $j \leq j_{max}$
    Set $p(j+1) = p(j) + \alpha(j) d(j)$
    where $d(j)$ is a descent direction and $\alpha(j)$ is computed by means of a line search.
    Set $j = j + 1$.
End While
If (2.14) holds, then set $\tilde{p}_k = p(j)$.
If (3.2) holds, then set $\tilde{p}_k = p(j - N)$.
If $j > j_{max}$, then set $\tilde{p}_k = p(j_{max})$.
```

Algorithm 3.2: k-th iteration of the proposed version of ARC

Given x_k, the scalars $\sigma_k > 0$, $1 > \eta_2 \geq \eta_1 > 0$, $\gamma_2 \geq \gamma_1 > 1$, $\epsilon > 0$, $\alpha > 0$, and the integer $N \geq 1$.

1. Compute the Cauchy point p_k^c, and apply Algorithm 3.1 to compute $\tilde{p}_k$.

   If
   $$\frac{f(x_k) - f(x_k + \tilde{p}_k)}{f(x_k) - m_k(\tilde{p}_k)} \geq \eta_1 \tag{3.3}$$
   then, if
   $$f(x_k) - m_k(\tilde{p}_k) \geq \alpha\epsilon^{3/2}, \tag{3.4}$$
   then set $p_k = \tilde{p}_k$ and go to Step 3; otherwise, apply Algorithm 2.2 to compute p_k.

2. Compute
   $$\rho_k = \frac{f(x_k) - f(x_k + p_k)}{f(x_k) - m_k(p_k)}. \tag{3.5}$$

3. Set
   $$x_{k+1} = \begin{cases} x_k + p_k & \text{if } \rho_k \geq \eta_1, \\ x_k & \text{otherwise.} \end{cases}$$

4. Set
   $$\sigma_{k+1} \in \begin{cases} (0, \sigma_k] & \text{if } \rho_k \geq \eta_2 & \text{(very successful iteration),} \\ [\sigma_k, \gamma_1\sigma_k) & \text{if } \eta_1 \leq \rho_k \leq \eta_2 & \text{(successful iteration),} \\ [\gamma_1\sigma_k, \gamma_2\sigma_k] & \text{otherwise} & \text{(unsuccessful iteration).} \end{cases}$$

Note that the trial point $\tilde{p}_k$ must satisfy condition (3.4) for the successful iterations. This condition aims to ensure the bound on the global worst-case iteration complexity.

The global convergence of Algorithm 3.2 follows from Proposition 2.1 while its complexity bound is stated below.

Proposition 3.1. *Let $\{x_k\}$ be the sequence generated by Algorithm 3.2, and let $\{f(x_k)\}$ be bounded below. Assume that:*

(i) $f \in C^2(\mathbb{R}^n)$*;*

(ii) $\|g(x) - g(y)\| \leq \mathcal{K}\|x - y\|$*, for all $x, y \in X$ an open convex set containing the iterates and $\mathcal{K} \geq 1$;*

(iii) $\|H(x) - H(x_k)\| \leq L\|x - x_k\|$*, for all $x \in [x_k, x_k + p_k]$ and all $k \geq 0$;*

(iv) $\|(H(x_k) - B_k)p_k\| \leq C\|p_k\|^2$*, for all $k \geq 0$ and some positive constant C;*

(v) *$\sigma_k \geq \sigma_{min}$ for all $k \geq 0$ and some positive σ_{min}.*

Then Algorithm 2.1 requires at most $\mathcal{O}(\epsilon^{-3/2})$ iterations to attain

$$\|g(x_k)\| \leq \epsilon.$$

Proof. Let K_s be the index subset of successful iterations such that

$$\min\{\|g(x_k)\|,\, \|g(x_{k+1})\|\} > \epsilon.$$

For all $k \in K_s$ the step p_k satisfies (2.16) either because $\tilde{p}_k$ does or because p_k is computed by Algorithm 2.2 and by virtue of Proposition 2.4. Then, the result follows from Proposition 2.2. □

3.4 Numerical results

In this Section we report the results of the computational experiments performed in [4] in order to assess the effectiveness of ARC algorithms using iterative methods and early stopping.

3.4.1 Implementation details

Iterative cubic subproblem solvers

As iterative descent method applied to the cubic model $m_k(p)$ (see Algorithm 3.1) for computing the trial step p_k, we employ the Barzilai-Borwein Non-Monotone GRADient method (NMGRAD) defined in [19]. The proposed version of ARC algorithm, called ARC-NMGRAD, has been compared with the original ARC version (ARC-GLRT) proposed in [9], and using the Lanczos-based inexact solver implemented in GALAHAD-GLRT [12, 14].

Test problems

We considered two sets of test problems. The first set is taken from the CUTEr collection [13], the second one is taken from the Luksan's collection of dense test problems for unconstrained minimization[1]. As regards the CUTEr collection, we selected all the variable dimension nonlinear and unconstrained problems, thus coming up with a set of 52 CUTEr medium-sized ($n \in [1000, 2000]$) problems. The Luksan's collection is a set of 92 problems whose dimension n has been chosen in the range $[961, 1000]$.

[1]The collection is available at `http://www.cs.cas.cz/luksan/test.html`

ARC implementation details

We have implemented the ARC algorithms in Fortran90, with B_k set to the true Hessian $H(x_k)$. For the test problems from the Luksan's collection, the Hessian/vector product is obtained by finite differences.

The parameters defining the original ARC method (Algorithm 2.1) and the proposed ARC algorithm (Algorithm 3.2) have been chosen as described in [9]. The parameter α of condition (3.4) in Algorithm 3.2 has been set equal to 10^{-8}.

For all algorithms, the maximum number of outer iterations has been fixed equal to 50000, and a limit of 500 seconds on the computing time has been imposed. The algorithms terminate successfully when

$$\|g(x_k)\| \leq \epsilon, \qquad \text{with } \epsilon = 10^{-5}.$$

Implementation details on subproblem solution

The GALAHAD-GLRT solver of the original ARC method has been run with a memory parameter $m = 10$ and requiring 90% accuracy in solution reconstruction [15].

Concerning the early stopping parameter N of Algorithm 3.1, we conducted an experiment to understand its influence on the overall algorithm. In particular, we compared three versions of the algorithm with the early stopping parameter $N = 5, 10, \infty$, where the symbol ∞ indicates that the early stopping criterion was inhibited. In the comparison, to distinguish the versions of algorithm ARC-NMGRAD with respect to the early stopping parameter N, we denote them as ARC-NMGRAD(N).

The subproblem solvers have been invoked specifying a limit of 1000 iterations (i.e., $j_{max} = 1000$ in Algorithm 3.1), and using 10^{-4} as value of θ in the relative stopping criterion (2.14).

3.4.2 Numerical results on CUTEr problems

Algorithms have been compared by means of the performance profiles proposed in [11]. Failures are accounted for by setting the performance index $t_{p,s}$ for the pair (p, s), relative to a given problem (p) and solver (s), to a reference high value.

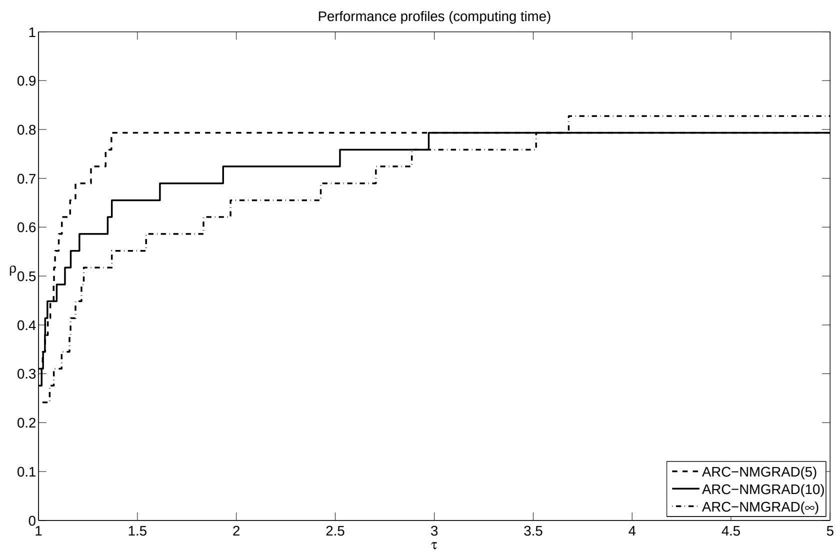

Figure 3.2: Performance profiles for ARC-NMGRAD(5), ARC-NMGRAD(10), and ARC-NMGRAD(∞) on the CUTEr problems.

First we compare the relative efficiency of ARC-NMGRAD(5), ARC-NMGRAD(10) and ARC-NMGRAD(∞) to show the effectiveness of the introduced early stopping criterion. The results are plotted in Figure 3.2. The benefits deriving from the adoption of the early stopping are evident. Indeed, both the versions using the early stopping criterion clearly outperform the version with $N = \infty$ in terms both of efficiency and robustness. Furthermore, it can be noted that ARC-NMGRAD(5) and ARC-NMGRAD(10) are substantially comparable and in the following we compare ARC-GLRT against ARC-NMGRAD(5).

For the sake of completeness the complete results of the comparison between ARC-GLRT and ARC-NMGRAD(5) are reported in Tables 3.1 and 3.2 respectively. The symbols n_i, n_f, n_g, f^*, and cpu denote the number of iterations, the number of function evaluations, the number of gradient evaluations, the final objective function value and the cpu time (in seconds), respectively. The performance profiles relative to this comparison are plotted in Figure 3.3. We note that the number of wins of ARC-GLRT is higher than that

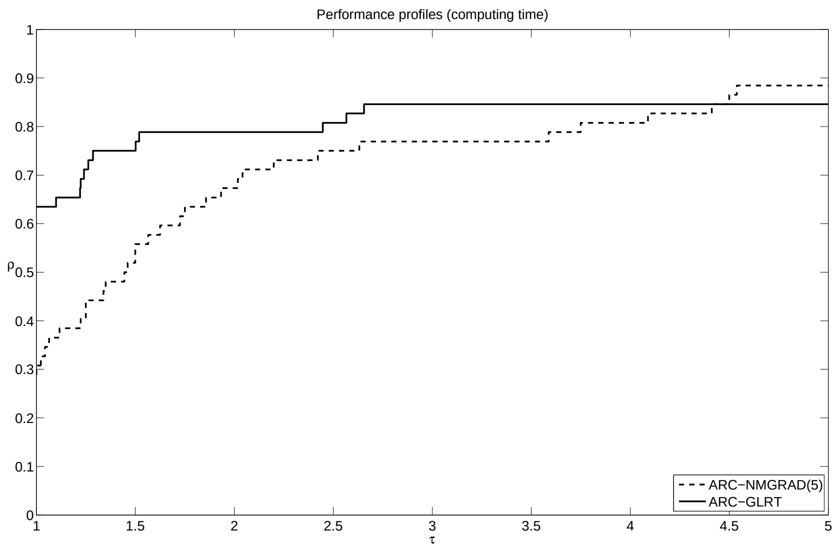

Figure 3.3: Performance profiles for ARC-NMGRAD(5), and ARC-GLRT on the CUTEr problems.

of ARC-NMGRAD(5) but on almost 70% of the tests ARC-NMGRAD(5) is within a factor 2 of the CPU time required by ARC-GLRT. Remarkably, ARC-NMGRAD(5) compares favorably with ARC-GLRT in terms of robustness. On the whole, ARC-NMGRAD(5) can be considered competitive with ARC-GLRT. To further support this conclusion, we have performed a comparison between ARC-GLRT and ARC-NMGRAD(5) on a set of runs representing the slower ones. In particular, by eliminating the test problems where both algorithms require a cpu time lower than 1 second, we obtained the performance profiles in Figure 3.5 which show the effectiveness of ARC-NMGRAD(5).

Concerning the actual failures reported by the two algorithms, we observe that:

- ARC-GLRT solves 45 over 52 problems; six of these failures are due to the CPU time limit, and one is due to the limit on the number of outer iterations

- ARC-NMGRAD(5) solves 47 over 52 problems; four of these failures are due to the CPU time limit, and one is due to the limit on the number of outer iterations.

Finally, to get more insight into the performance of ARC-NMGRAD and ARC-GLRT, we further analyze the effects of using both NMGRAD and the early stopping criterion and in Figure 3.4 we compare ARC-NMGRAD(∞) and

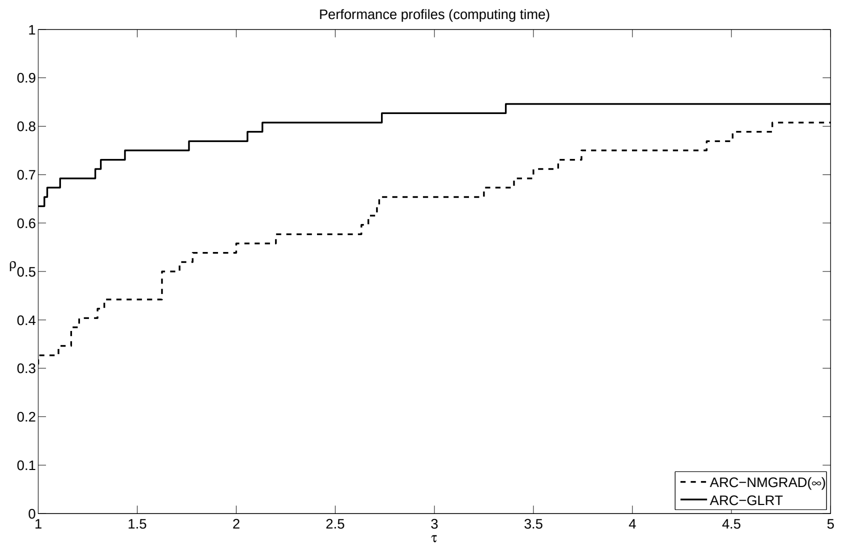

Figure 3.4: Performance profiles for ARC-NMGRAD(∞), and ARC-GLRT on the CUTEr problems.

ARC-GLRT.
From the performance profiles plotted in Figure 3.3 and in Figure 3.4 we can observe that the adoption of the early stopping criterion is crucial to make ARC-NMGRAD competitive with ARC-GLRT.

Table 3.1: Results obtained by ARC-GLRT on the CUTEr problems

Name	n_i	n_f	n_g	f^*	cpu(secs.)
ARWHEAD	8	9	9	0.000000e+00	8.001000e−03
BDQRTIC	19	20	20	3.983818e+03	8.000501e−02
BROWNBS	Not solved within 500 secs. CPU time				
BROYDN7D	111	112	105	3.649243e+02	7.000430e−01
BRYBND	30	31	25	3.905249e−14	2.240140e−01
CHAINWOO	855	856	788	5.660338e+02	8.088505e+00
CRAGGLVY	23	24	24	3.364231e+02	1.240080e−01
CURLY10	162	163	153	-1.003163e+05	1.721109+01
CURLY20	303	304	231	-1.003163e+05	3.0173885+01
CURLY30	229	230	152	-1.003163e+05	3.2834049+02
DIXMAANA	17	18	18	1.000000e+00	3.200200e−02
DIXMAANB	16	17	17	1.000000e+00	3.200200e−02
DIXMAANC	16	17	17	1.000000e+00	4.800300e−02
DIXMAAND	21	22	21	1.000000e+00	6.800400e−02
DIXMAANE	51	52	52	1.000000e+00	3.960250e−01
DIXMAANF	34	35	35	1.000000e+00	3.520220e−01
DIXMAANG	32	33	33	1.000000e+00	3.560220e−01
DIXMAANH	35	36	35	1.000000e+00	4.360270e−01

continued on next page

Table 3.1 – continued from previous page

Name	n_i	n_f	n_g	f^*	cpu(secs.)
DIXMAANI	99	100	100	1.000000e+00	4.988312e+00
DIXMAANJ	44	45	45	1.000000e+00	3.388211e+00
DIXMAANK	39	40	37	1.000000e+00	2.124132e+00
DIXMAANL	47	48	44	1.000000e+00	4.092256e+00
DQRTIC	42	43	43	3.264570e−07	3.600300e−02
EDENSCH	23	24	24	1.200328e+04	9.200601e−02
ENGVAL1	13	14	14	1.108195e+03	2.400100e−02
EXTROSNB	5800	5801	1340	1.276625e−08	4.435717e+02
FLETCBV2	12	13	13	-5.014290e−01	1.316082e+00
FLETCBV3	Not solved within 50000 outer iterations				
FLETCHBV	Not solved within 500 secs. CPU time				
FLETCHCR	1678	1679	1514	9.091554e−15	4.772298e+00
FMINSRF2	246	247	240	1.000000e+00	3.500218e+00
FREUROTH	38	39	31	1.214697e+05	7.600500e−02
GENHUMPS	Not solved within 500 secs. CPU time				
GENROSE	1012	1013	556	1.000000e+00	4.204263e+00
LIARWHD	17	18	18	2.147984e−18	2.000100e−02
MOREBV	4	5	5	1.683811e−09	3.720230e−01
NONCVXU2	935	936	935	2.317705e+03	1.751309e+01
NONCVXUN	Not solved within 500 secs. CPU time				
NONMSQRT	Not solved within 500 secs. CPU time				
NONDIA	5	6	6	2.944539e−17	8.000001e−03
NONDQUAR	101	102	84	1.007064e−06	8.720540e−01
OSCIPATH	6	7	7	9.999667e−01	4.000001e−03
POWELLSG	26	27	27	4.010935e−08	1.600100e−02
QUARTC	42	43	43	3.264570e−07	2.800100e−02
SINQUAD	36	37	30	-2.942505e+05	6.800400e−02
SPARSINE	Not solved within 500 secs. CPU time				
SPARSQUR	19	20	20	1.767216e−08	6.000400e−02
SPMSRTLS	23	24	23	5.173202e−12	2.400150e−01
SROSENBR	9	10	10	1.278262e−19	4.000999e−03
TOINTGSS	20	21	21	1.001002e+01	4.800300e−02
TQUARTIC	38	39	39	2.557968e−13	3.200200e−02
WOODS	58	59	59	1.570289e−23	5.200300e−02

Table 3.2: Results obtained by ARC-NMGRAD(5) on the CUTEr problems

Name	n_i	n_f	n_g	f^*	cpu(secs.)
ARWHEAD	6	14	7	0.000000e+00	1.200100e−02
BDQRTIC	10	67	11	3.983818e+03	1.080070e−01
BROWNBS	Not solved within 500 secs. CPU time				
BROYDN7D	69	478	66	3.456708e+02	5.640350e−01
BRYBND	21	501	17	3.590946e−13	9.160580e−01
CHAINWOO	611	7876	586	4.595685e+02	6.624414e+00
CRAGGLVY	16	114	17	3.364231e+02	1.320090e−01
CURLY10	141	20096	128	-1.003163e+05	1.377686e+01
CURLY20	149	21290	132	-1.003163e+05	1.986524e+01
CURLY30	151	21397	131	-1.003163e+05	2.601362e+01
DIXMAANA	16	46	17	1.000000e+00	5.600400e−02

continued on next page

Table 3.2 – continued from previous page

Name	n_i	n_f	n_g	f^*	cpu(secs.)
DIXMAANB	15	42	16	1.000000e+00	5.200300e−02
DIXMAANC	15	48	16	1.000000e+00	6.000300e−02
DIXMAAND	68	235	62	1.000000e+00	3.000180e−01
DIXMAANE	46	271	47	1.000000e+00	3.600220e−01
DIXMAANF	29	209	30	1.000000e+00	3.600230e−01
DIXMAANG	32	265	33	1.000000e+00	4.360270e−01
DIXMAANH	88	556	81	1.000000e+00	8.800550e−01
DIXMAANI	76	1424	77	1.000003e+00	1.944121e+00
DIXMAANJ	33	789	34	1.000003e+00	1.276079e+00
DIXMAANK	37	1797	38	1.000006e+00	2.844178e+00
DIXMAANL	105	3882	97	1.000009e+00	7.064441e+00
DQRTIC	42	155	43	8.665368e−08	5.200300e−02
EDENSCH	16	63	17	1.200328e+04	9.600600e−02
ENGVAL1	9	35	10	1.108195e+03	2.400200e−02
EXTROSNB	522	74577	481	1.390821e−06	6.204388e+01
FLETCBV2	7	1112	8	-5.014268e−01	8.760550e−01
FLETCBV3	Not solved within 50000 outer iterations				
FLETCHBV	Not solved within 500 secs. CPU time				
FLETCHCR	1512	16297	1511	1.675118e−13	1.155672e+01
FMINSRF2	147	3912	144	1.000012e+00	5.116320e+00
FREUROTH	22	216	20	1.214697e+05	2.000120e−01
GENHUMPS	33208	129369	20367	5.058678e−14	1.594540e+02
GENROSE	1053	7126	578	1.000000e+00	4.696294e+00
LIARWHD	17	63	18	5.043585e−23	4.400200e−02
MOREBV	2	241	3	1.527327e−09	1.520090e−01
NONCVXU2	866	19162	866	2.317235e+03	2.740971e+01
NONCVXUN	Not solved within 500 secs. CPU time				
NONMSQRT	Not solved within 500 secs. CPU time				
NONDIA	6	17	7	1.595894e−23	1.200100e−02
NONDQUAR	73	3246	61	1.833421e−06	1.780112e+00
OSCIPATH	2	7	3	9.999667e−01	4.000001e−03
POWELLSG	25	134	26	3.460945e−08	6.000300e−02
QUARTC	42	155	43	8.665368e−08	5.200300e−02
SINQUAD	36	191	30	-2.942505e+05	2.440160e−01
SPARSINE	127	13490	116	1.935705e−09	2.356147e+01
SPARSQUR	19	88	20	9.255839e−09	1.160070e−01
SPMSRTLS	19	142	18	4.585779e−13	1.960120e−01
SROSENBR	14	48	15	1.942930e−20	2.400200e−02
TOINTGSS	17	55	18	1.001002e+01	6.000300e−02
TQUARTIC	38	162	39	2.497239e−10	1.440090e−01
WOODS	56	403	57	2.769636e−18	2.360150e−01

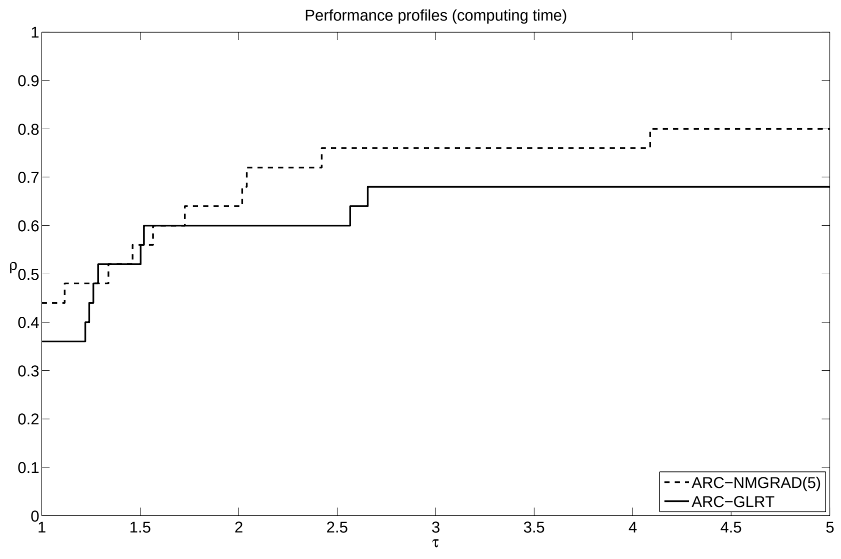

Figure 3.5: Performance profiles for ARC-NMGRAD(5), and ARC-GLRT on the CUTEr problems where at least one algorithm requires a CPU time greater than 1 second.

3.4.3 Numerical results on Luksan's problems

For this set of test problems, the Hessian/vector product $H(x_k)d$ is approximated by finite differences as follows

$$H(x_k)d \simeq \frac{g(x_k + \delta d) - g(x_k)}{\delta},$$

where

$$\delta = 2 \cdot 10^{-6} \frac{(1 + \|x_k\|)}{\max\{10^{-5}, \|d\|\}}.$$

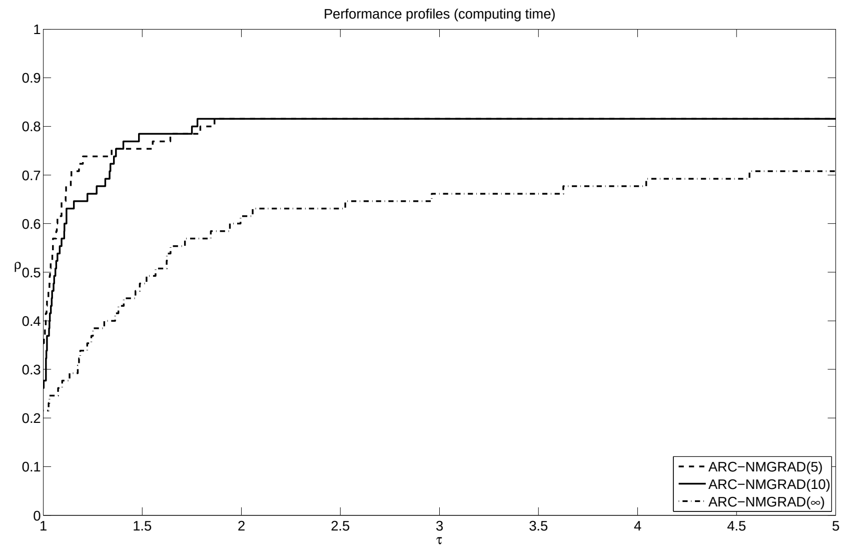

Figure 3.6: Performance profiles for ARC-NMGRAD(5), ARC-NMGRAD(10), and ARC-NMGRAD(∞) on the Luksan's problems.

First we compare the relative efficiency of ARC-NMGRAD(5), ARC-NMGRAD(10) and ARC-NMGRAD(∞) to show the effectiveness of the introduced early stopping criterion. The results are plotted in Figure 3.6 and confirm the validity of the early stopping criterion.

Then we have compared ARC-GLRT versus ARC-NMGRAD(5). The complete results relative to this comparison can be found in Tables 3.3 and 3.4 respectively. The symbols n_i, n_f, n_g, f^*, and cpu denote the number of iterations, the number of function evaluations, the number of gradient evaluations, the final objective function value and the cpu time (in seconds), respectively. The performance profiles representing this comparison are plotted in Figure 3.7. We note that for these problems, ARC-GLRT and ARC-NMGRAD(5) are almost equivalent in terms of wins. On the other hand, ARC-GLRT is slightly

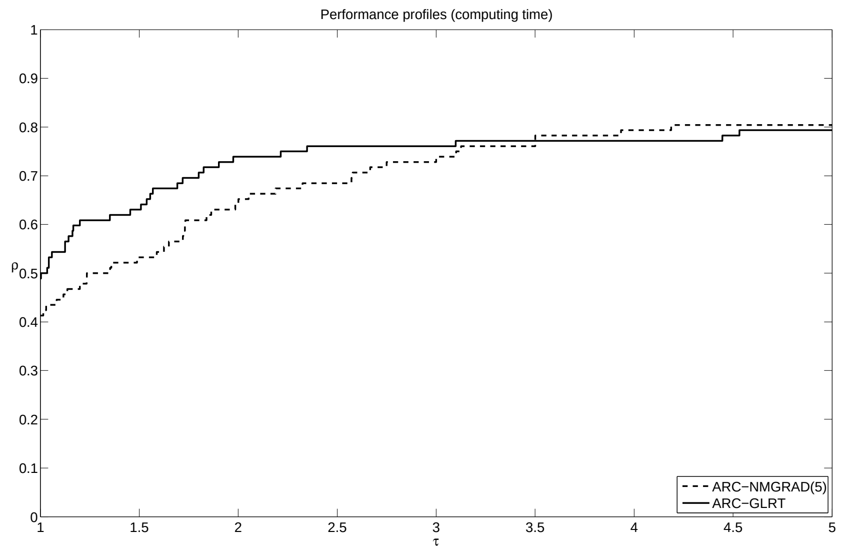

Figure 3.7: Performance profiles for ARC-NMGRAD(5), and ARC-GLRT on the Luksan's problems.

more efficient than ARC-NMGRAD(5) while this latter shows good performance in terms of robustness.

As done for CUTEr collection, we have compared ARC-GLRT and ARC-NMGRAD(5) by eliminating "easy" problems, that is, the test problems where both algorithms require a cpu time lower than 1 second. The performance profiles are reported in Figure 3.9 showing again a satisfactory behavior of ARC-NMGRAD(5).

Concerning the actual failures reported by the two algorithms, we point out that:

- ARC-GLRT solves 78 over 92 problems; twelve failures are due to the CPU time limit, and two are due to the iterations limit;
- ARC-NMGRAD(5) solve 81 over 92 problems; the eleven failures are due to the CPU time limit.

Finally, we report the performance profiles relative to the comparison between ARC-NMGRAD(∞) and ARC-GLRT. Figure 3.8 shows that, also for this set of problems, the use of the early stopping criterion in ARC-NMGRAD is crucial to be competitive with ARC-GLRT.

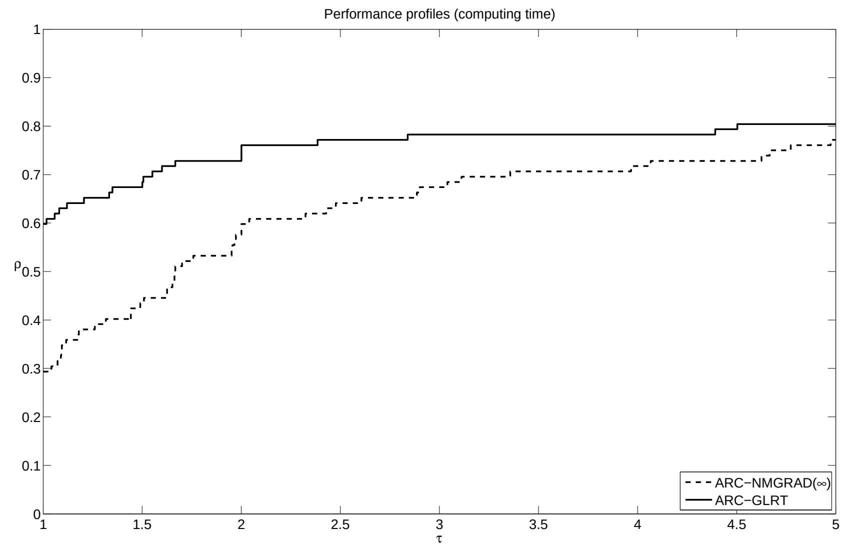

Figure 3.8: Performance profiles for ARC-NMGRAD(∞), and ARC-GLRT on the Luksan's problems.

Table 3.3: Results obtained by ARC-GLRT on the Luksan's problems

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 1	1705	1706	1525	2.666630e−14	4.252265e+00
LUKSAN 2	992	993	887	6.980949e+02	8.264516e+00
LUKSAN 3	20	21	21	1.310668e−09	4.000200e−02
LUKSAN 4	26	27	27	2.694995e+02	4.680290e−01
LUKSAN 5	14	15	15	5.745074e−12	8.800500e−02
LUKSAN 6	15	16	16	6.625240e−11	2.920180e−01
LUKSAN 7	13	14	14	3.369372e+02	1.280070e−01
LUKSAN 8	Not solved within 500 secs. CPU time				
LUKSAN 9	50	51	51	3.164361e+02	1.244077e+00
LUKSAN 10	172	173	71	-1.331500e+02	5.358735e+01
LUKSAN 11	60	61	55	1.077659e+01	1.080060e−01
LUKSAN 12	27	28	28	9.989331e+01	2.800100e−02
LUKSAN 13	13	14	14	1.666597e−24	4.000200e−02
LUKSAN 14	1	2	2	2.149546e−08	3.720230e−01
LUKSAN 15	29	30	30	1.924016e+00	4.354672e+01
LUKSAN 16	19	20	20	-4.274045e+02	1.396087e+00
LUKSAN 17	27	28	28	-3.799211e−02	3.880242e+00
LUKSAN 18	19	20	20	-2.457412e−02	1.832114e+00
LUKSAN 19	57	58	58	5.959862e+01	5.608350e+00
LUKSAN 20	86	87	81	-1.000135e+00	1.028864e+01
LUKSAN 21	42	43	43	2.138664e+00	1.401287e+01
LUKSAN 22	Not solved within 500 secs. CPU time				
LUKSAN 23	64	65	48	2.344534e+01	2.416150e+00
LUKSAN 24	Not solved within 500 secs. CPU time				

continued on next page

Table 3.3 – continued from previous page

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 25	36	37	37	4.183369e−22	2.400100e−02
LUKSAN 26	31	32	32	6.586576e−09	3.200200e−02
LUKSAN 27	52	53	40	4.843106e−03	3.600200e−02
LUKSAN 28	37	38	38	3.094958e−22	1.600100e−02
LUKSAN 29	3	4	4	1.947689e−13	2.240130e−01
LUKSAN 30	8	9	9	2.743256e−11	2.048128e+00
LUKSAN 31	13	14	14	7.904777e−13	2.400100e−02
LUKSAN 32	14	15	15	3.262056e−13	2.000100e−02
LUKSAN 33	14	15	15	2.087061e−12	1.080060e−01
LUKSAN 34	36	37	29	6.070537e+04	6.000300e−02
LUKSAN 35	46	47	41	1.095373e−07	1.184874e+01
LUKSAN 36	Not solved within 500 secs. CPU time				
LUKSAN 37	23	24	18	1.911663e+02	2.640165e+00
LUKSAN 38	3814	3815	3437	4.562540e−20	1.348084e+01
LUKSAN 39	Not solved within 50000 outer iterations				
LUKSAN 40	77	78	23	1.31234018+5 2.920180e−01	
LUKSAN 41	19	20	16	1.085179e+02	2.168136e+01
LUKSAN 42	70	71	17	1.817631e+01	3.396212e+00
LUKSAN 43	23	24	20	2.511097e+00	3.421014e+01
LUKSAN 44	Not solved within 500 secs. CPU time				
LUKSAN 45	99	100	99	1.088574e−13	9.040560e−01
LUKSAN 46	50	51	51	6.278438e−13	3.200200e−01
LUKSAN 47	16127	16128	16128	1.075591e−10	2.109332e+02
LUKSAN 48	46	47	43	6.507736e+02	1.720100e−01
LUKSAN 49	27	28	24	4.486970e+03	6.280390e−01
LUKSAN 50	38	39	30	5.054050e−14	3.172998e+01
LUKSAN 51	47	48	41	1.417404e−11	2.560150e−01
LUKSAN 52	12	13	13	7.086282e+00	1.760110e−01
LUKSAN 53	1485	1486	1282	3.000935e+05	2.694888e+02
LUKSAN 54	18	19	19	3.052255e−08	6.800400e−02
LUKSAN 55	788	789	561	4.998163e−01	6.320395e+00
LUKSAN 56	525	526	423	3.158296e−01	5.776361e+00
LUKSAN 57	113	114	114	1.224606e+04	5.600300e−02
LUKSAN 58	39	40	40	4.098448e−07	8.800500e−02
LUKSAN 59	14	15	15	1.806224e−12	3.200200e−02
LUKSAN 60	2454	2455	2455	9.360219e−05	6.696418e+00
LUKSAN 61	446	447	435	7.331583e−18	4.200260e−01
LUKSAN 62	Not solved within 500 secs. CPU time				
LUKSAN 63	Not solved within 500 secs. CPU time				
LUKSAN 64	114	115	115	3.453677e−07	8.489330e+01
LUKSAN 65	570	571	570	1.251357e−04	4.014011e+02
LUKSAN 66	298	299	298	1.372173e−07	1.746429e+02
LUKSAN 67	103	104	104	2.080020e−11	4.110657e+01
LUKSAN 68	153	154	154	3.691512e−12	8.396524e+00
LUKSAN 69	Not solved within 500 secs. CPU time				
LUKSAN 70	63	64	64	7.861817e−10	1.605300e+01
LUKSAN 71	38	39	39	7.386779e−12	2.020126e+00
LUKSAN 72	223	224	224	7.563986e−09	1.875477e+02
LUKSAN 73	Not solved within 500 secs. CPU time				
LUKSAN 74	34	35	35	1.275229e−11	1.974923e+01
LUKSAN 75	Not solved within 500 secs. CPU time				
LUKSAN 76	21	22	22	5.366343e−18	1.200000e−02

continued on next page

Table 3.3 – continued from previous page

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 77	10	11	11	2.642602e−11	2.396149e+00
LUKSAN 78	0	1	1	4.980050e−10	0.000000e+00
LUKSAN 79	41	42	42	1.694190e−11	1.120070e−01
LUKSAN 80	12	13	13	4.423241e−13	8.800500e−02
LUKSAN 81	11	12	12	2.874604e−12	5.600300e−02
LUKSAN 82	12	13	13	4.031545e−20	3.200200e−02
LUKSAN 83	79	80	80	4.302980e−07	6.925233e+01
LUKSAN 84	12	13	13	2.731366e−13	3.600200e−02
LUKSAN 85	Not solved within 500 secs. CPU time				
LUKSAN 86	11	12	12	3.899674e−11	5.528346e+00
LUKSAN 87	85	86	86	4.403826e−07	3.496218e+01
LUKSAN 88	Not solved within 500 secs. CPU time				
LUKSAN 89	Not solved within 50000 outer iterations				
LUKSAN 90	0	1	1	1.664531e−08	0.000000e+00
LUKSAN 91	107	108	108	5.510381e−11	1.407688e+01
LUKSAN 92	218	219	219	3.127943e−11	8.328520e+00

Table 3.4: Results obtained by ARC-NMGRAD(5) on the Luksan's problems

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 1	1520	16465	1517	3.986624e+00	9.884617e+00
LUKSAN 2	607	7177	587	2.480440e+02	4.184261e+00
LUKSAN 3	16	274	17	4.999968e−09	2.520150e−01
LUKSAN 4	19	349	20	2.694995e+02	1.024064e+00
LUKSAN 5	12	44	13	1.399421e−11	1.000060e−01
LUKSAN 6	13	83	14	3.379594e−12	2.800170e−01
LUKSAN 7	7	27	8	3.369372e+02	8.800500e−02
LUKSAN 8	79	462	7	7.617750e+05	1.140471e+01
LUKSAN 9	47	142	48	3.164361e+02	2.140133e+00
LUKSAN 10	140	1992	62	-1.336300e+02	3.117395e+01
LUKSAN 11	41	433	40	1.077659e+01	1.044065e+00
LUKSAN 12	27	69	28	9.989331e+01	7.200400e−02
LUKSAN 13	13	37	14	5.232574e−23	1.240070e−01
LUKSAN 14	1	175	2	2.146466e−08	1.680100e−01
LUKSAN 15	Not solved within 500 secs. CPU time				
LUKSAN 16	7	150	8	-4.274045e+02	9.080560e−01
LUKSAN 17	17	2297	18	-3.799211e−02	5.248327e+00
LUKSAN 18	16	2606	17	-2.457412e−02	1.756109e+00
LUKSAN 19	29	3110	30	5.959862e+01	5.772360e+00
LUKSAN 20	47	2745	41	-1.000135e+00	6.824426e+00
LUKSAN 21	34	5113	35	2.138664e+00	1.205675e+01
LUKSAN 22	511	8240	85	1.000000e+00	6.544408e+00
LUKSAN 23	106	807	43	2.344534e+01	2.616163e+00
LUKSAN 24	Not solved within 500 secs. CPU time				
LUKSAN 25	37	133	38	6.984929e−21	7.200400e−02
LUKSAN 26	28	151	29	3.938530e−08	1.000060e−01
LUKSAN 27	50	120	40	4.843088e−03	7.200400e−02
LUKSAN 28	36	73	37	3.317687e−18	4.400200e−02
LUKSAN 29	2	5	3	1.948019e−13	3.880240e−01

continued on next page

Table 3.4 – continued from previous page

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 30	5	16	6	4.227790e−17	2.528158e+00
LUKSAN 31	6	25	7	7.931709e−19	2.000100e−02
LUKSAN 32	7	24	8	3.559841e−20	2.400100e−02
LUKSAN 33	7	38	8	3.233905e−13	6.000300e−02
LUKSAN 34	21	228	20	6.073486e+04	2.360140e−01
LUKSAN 35	57	5461	52	2.447920e−06	4.961510e+01
LUKSAN 36	218	628	202	2.216459e+03	7.880490e−01
LUKSAN 37	13	111	9	1.911663e+02	1.388087e+00
LUKSAN 38	3488	31681	3359	1.614838e−13	2.766573e+01
LUKSAN 39	78	658	66	2.223755e+04	1.388086e+00
LUKSAN 40	18	148	17	1.312340e+05	2.160130e−01
LUKSAN 41	4	31	5	1.085179e+02	3.316207e+00
LUKSAN 42	13	45	8	1.817631e+01	1.096068e+00
LUKSAN 43	12	579	13	2.511097e+00	2.181336e+01
LUKSAN 44	45	1023	36	4.218847e−03	8.080500e−01
LUKSAN 45	2856	347989	2680	4.432440e−17	3.192640e+02
LUKSAN 46	43	1233	44	1.648348e−15	1.768110e+00
LUKSAN 47	10123	103701	10124	3.107663e−09	1.156152e+02
LUKSAN 48	14	138	15	6.476961e+02	1.920120e−01
LUKSAN 49	17	225	17	4.486970e+03	1.036064e+00
LUKSAN 50	81	321	60	8.017310e−17	5.038715e+01
LUKSAN 51	34	658	29	4.908125e−15	6.600410e−01
LUKSAN 52	13	47	11	1.761899e−17	3.280200e−01
LUKSAN 53	938	28929	884	3.093525e+03	2.735691e+02
LUKSAN 54	17	97	18	2.092815e−08	8.400500e−02
LUKSAN 55	660	10475	549	4.998163e−01	1.162873e+01
LUKSAN 56	393	2415	324	3.158296e−01	2.460153e+00
LUKSAN 57	100	351	101	1.224606e+04	1.960120e−01
LUKSAN 58	38	160	39	4.863680e−07	3.080190e−01
LUKSAN 59	8	41	9	2.882457e−12	2.800100e−02
LUKSAN 60	Not solved within 500 secs. CPU time				
LUKSAN 61	275	8529	257	1.939807e−10	6.456403e+00
LUKSAN 62	Not solved within 500 secs. CPU time				
LUKSAN 63	Not solved within 500 secs. CPU time				
LUKSAN 64	51	5679	52	1.382230e−05	1.873317e+01
LUKSAN 65	Not solved within 500 secs. CPU time				
LUKSAN 66	152	27412	153	6.347841e−06	3.928245e+01
LUKSAN 67	117	20726	118	6.458919e−06	7.115645e+01
LUKSAN 68	85	4662	86	1.095757e−07	8.368523e+00
LUKSAN 69	Not solved within 500 secs. CPU time				
LUKSAN 70	65	9143	66	5.346279e−07	1.516495e+01
LUKSAN 71	29	1024	30	7.329030e−12	1.952121e+00
LUKSAN 72	Not solved within 500 secs. CPU time				
LUKSAN 73	Not solved within 500 secs. CPU time				
LUKSAN 74	31	94	32	1.784223e−14	3.400613e+01
LUKSAN 75	2841	66630	1267	1.485122e−05	4.209863e+01
LUKSAN 76	21	59	22	2.718483e−17	3.200100e−02
LUKSAN 77	9	29	10	2.312421e−13	4.756297e+00
LUKSAN 78	0	1	1	4.980050e−10	0.000000e+00
LUKSAN 79	35	161	36	3.619068e−12	9.600600e−02
LUKSAN 80	5	25	6	1.005674e−16	5.200300e−02
LUKSAN 81	6	26	7	1.730203e−15	3.600200e−02

continued on next page

Table 3.4 – continued from previous page

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 82	12	33	13	8.413166e−19	5.200300e−02
LUKSAN 83	21	2863	22	1.128760e−05	7.816488e+00
LUKSAN 84	7	23	8	3.308213e−13	3.200100e−02
LUKSAN 85	Not solved within 500 secs. CPU time				
LUKSAN 86	9	27	10	1.049445e−16	8.228515e+00
LUKSAN 87	24	3433	25	1.218843e−05	3.084192e+00
LUKSAN 88	Not solved within 500 secs. CPU time				
LUKSAN 89	132	16687	124	1.063317e−05	4.737856e+02
LUKSAN 90	0	1	1	1.664531e−08	0.000000e+00
LUKSAN 91	59	4043	60	9.602579e−10	1.912919e+01
LUKSAN 92	134	4303	135	1.375732e−08	7.404462e+00

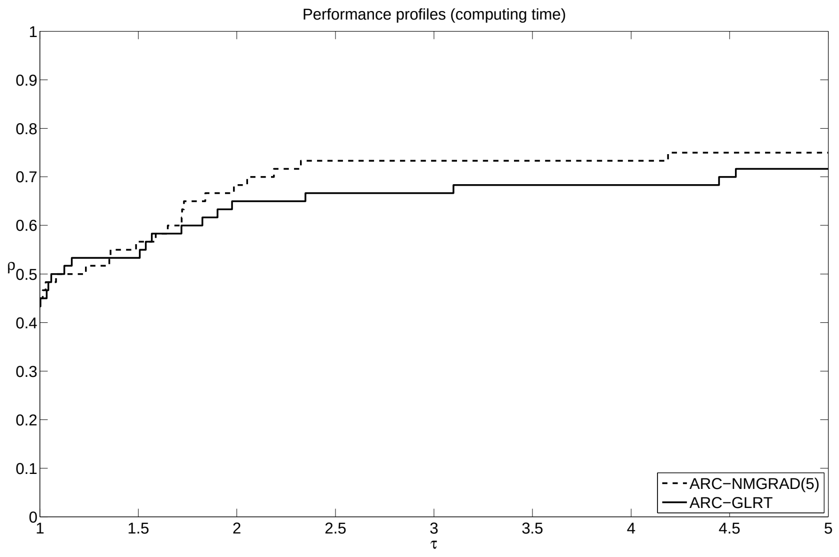

Figure 3.9: Performance profiles for ARC-NMGRAD(5), and ARC-GLRT on the Luksan's problems where at least one algorithm requires a CPU time greater than 1 second.

3.4.4 On the influence of Algorithm 2.2 on overall performance

In all the runs reported above, condition (3.4) of Algorithm 3.2 was tested with $\alpha = 10^{-8}$ and resulted to be satisfied; hence Algorithm 2.2 was never invoked. Therefore, in order to test the behavior of Algorithm 2.2, we have increased the value of α up to 10^{-4}, and we have analyzed the differences in the solution of CUTEr problems.

In Table 3.5 we show the 15 CUTEr problems where Algorithm 2.2 was invoked and report the results obtained with both $\alpha = 10^{-4}$ and $\alpha = 10^{-8}$. The results reported in Table 3.5 show that in 14 over 15 tests Algorithm 2.2 does not significantly influence the performance of the overall algorithm, while only in one case (SPARSINE) it drastically deteriorates the performance. We have verified that the strong increase of the CPU time in the solution of SPARSINE is due to the fact that Algorithm 3.1 performs the maximum number (50000) of iterations without satisfying the stopping criterion.

	n_i		n_f		n_g		cpu(secs.)	
Name	10^{-8}	10^{-4}	10^{-8}	10^{-4}	10^{-8}	10^{-4}	10^{-8}	10^{-4}
ARWHEAD	6	6	14	15	7	7	0.01	0.01
BDQRTIC	10	10	67	68	11	11	0.10	0.18
CHAINWO	611	611	7876	7877	586	586	6.61	6.66
CURLY20	149	142	21290	19898	132	125	19.86	19.48
CURLY30	151	147	21397	20601	131	125	26.01	25.24
DIXMAANA	16	16	46	47	17	17	0.05	0.05
DIXMAAND	68	68	235	236	62	62	0.30	0.30
ENGVAL1	9	9	35	36	10	10	0.02	0.03
LIARWHD	17	17	63	64	18	18	0.04	0.04
NONDIA	6	6	17	18	7	7	0.01	0.12
OSCIPATH	2	2	7	8	3	3	0.001	0.01
SINQUAD	36	36	191	192	30	30	0.24	0.25
SPARSINE	127	124	13490	12939	116	113	23.56	230.15
SROSENBR	14	14	48	49	15	15	0.02	0.02
WOODS	56	56	403	404	57	57	0.23	0.24

Table 3.5: Comparison of the results obtained by ARC-NMGRAD(5) with $\alpha = 10^{-8}$ and $\alpha = 10^{-4}$

In general, we may expect that Algorithm 2.2 is inefficient when the Hessian matrix becomes ill conditioned. On the other hand, our numerical experience indicate that such an algorithm does not have a crucial practical role, since condition (3.4) is typically satisfied for fairly small values of α.

3.4.5 General comments on the comparison between ARC-GLRT and ARC-NMGRAD

Summarizing, the results of the computational experiments show that the employment of a gradient-based method as inexact solver is a valid alternative to ARC-GLRT, and the adoption of the early stopping criterion enhances the performance of ARC-NMGRAD and leads to an efficient variant of ARC. Indeed, it appears that the early stopping strategy is a useful tool to properly manage the “over-solving” issue arising whenever the objective function may not be adequately represented by the cubic model.

The results reported previously show that, in most runs, ARC-NMGRAD requires a lower number of iterations than those required by ARC-GLRT. However, because of the early stopping rule, the number of function evaluations required by ARC-NMGRAD is much higher than that required by ARC-GLRT. This may be a serious issue whenever the function evaluation is the dominant cost. In this case, the parameter N in the early stopping should be taken large enough to reduce the computational burden while preserving the potential benefits of the early stopping criterion.

Chapter 4

A non-monotone ARC algorithm

It's trivial to verify that the sequences $\{f(x_k)\}$ generated by the Algorithms 2.1 and 3.2 are monotonically decreasing. This is due to the trial step acceptance procedure in Step 3. This property ensures that each successful iteration generates a point that is the best ever found. The main idea behind a nonmonotone method is to abandon this algorithmic restriction; this relaxation allows the sequence of iterates to better follow the objective function contours, especially in difficult nonlinear problems [17, 18]. In our knowledge, any nonmonotone method has been developed within ARC framework.

4.1 Non-monotone method sketch

Given a positive integer $M \in \mathbb{N}$, for each iteration k we define with $\mathcal{S}_{k,M}$ the set of indexes of last M successful iteration before the k^{th}, with the convention $\mathcal{S}_{0,M} = \{0\}$ and that if for a certain k the number of previous successful iteration is less than M, then $\mathcal{S}_{k,M}$ contains all successful iterations before the k^{th}.

We propose to substitute in the acceptance criterion the value of f of the last iterate with the maximum value obtained in the last M successful iterations. Hence at the beginning of each iteration we find an index $m(k)$ such that

$$f(x_{m(k)}) = \max_{j \in \mathcal{S}_{k,M}} f(x_j) \tag{4.1}$$

and we use the ratio

$$\frac{f(x_{m(k)}) - f(x_k + p_k)}{f(x_k) - m_k(p_k)}$$

to accept or to reject the candidate point. Since $f(x_{m(k)}) \geq f(x_k)$ we can have that the new point can be accepted even if $f(x_k + p_k) \geq f(x_k)$.

Another idea that we introduce in our algorithm is to look at the trial step p_k obtained by approximately minimizing the cubic model from a different perspective: we can imagine the trial step as a sort of descent direction for the real objective function. In order to get a better reduction of f we can try to perform an extrapolation along p_k and to find a scalar $\alpha > 0$ such that $f(x_k + \alpha p_k) < f(x_k + p_k)$. In such a way we try to reduct as much as possible the value of f with a single subproblem optimization.

Keeping in mind the new role of p_k, another kind of modification that can be done is to make a nonmonotone line search over p_k in order to get a sort of sufficient reduction of the real objective function compared against $f(x_{m(k)})$. In particular we propose to perform a nonmonotone Armijo line search that satisfies the following condition for a fixed constant $\beta \in (0, 1)$

$$f(x_{m(k)}) - f(x_k + \lambda_k p_k) \geq -\lambda_k \beta g(x_k)^T p_k, \tag{4.2}$$

where $\lambda_k > 0$ is the step obtained by mean of the line search.
Without imposing some conditions on the search direction p_k we are not sure that the line search ends in a finite number of steps producing a value λ_k that satisfies (4.2). For this reason in the proposed algorithm we activate the nonmonotone line search procedure only if p_k satisfies the following conditions

$$g(x_k)^T p_k \leq -c_1 \|g(x_k)\|^2 \tag{4.3}$$

$$\|p_k\| \leq c_2 \|g(x_k)\| \tag{4.4}$$

for some constants $c_1, c_2 > 0$.

All these changes to the standard ARC algorithm have the aim to exploit as much as possible the result of the approximate minimization of the cubic problem. In this way we try to take advantage of the computation effort made to obtain the candidate step p_k, that represent the dominant cost of the whole algorithm.

Before introducing the sketch of the proposed algorithm, we need first the following definition.
Let $\sigma : [0, +\infty) \to [0, +\infty)$ be a function of one real variable. We say that σ is a *forcing function* if for every sequence $\{t_k\} \subset [0, +\infty)$ we have that

$$\lim_{k\to\infty} \sigma(t_k) = 0 \Longrightarrow \lim_{k\to\infty} t_k = 0.$$

The proposed algorithm is formally described below.

Algorithm 4.1: k-th iteration of the proposed non-monotone algorithm

Given x_k, two positive integers $L, M \in \mathbb{N}$, and the scalars $c_1 \in (0,1)$, $c_2 > 1$, $\omega \in (0,1)$, $\beta \in (0,1)$, $\sigma_k > 0$, $1 > \eta_2 \geq \eta_1 > 0$, $\gamma_2 \geq \gamma_1 > 1$, $\alpha > 1$.

1. Compute an approximate minimizer p_k of m_k such that $m_k(p_k) \leq m_k(p_k^c)$ and find the index $m(k)$ defined in (4.1).

1.a. If the trial step p_k is such that (4.3) and (4.4) hold then set $j = 0$.

While $j < L$ and $f(\alpha^{j+1}p_k) < f(\alpha^j p_k)$, set $j = j + 1$.

Choose $\lambda_k \in \{\alpha^j, \alpha^j\omega, \alpha^j\omega^2, \ldots\}$ such that (4.2) holds.

1.b. Otherwise if

$$f(x_k) - f(x_k + p_k) \leq \hat{\sigma}(\|p_k\|), \tag{4.5}$$

where $\hat{\sigma}$ is a forcing function, set $\lambda_k = 1$.

1.c. If (4.5) is not satisfied, take $p_k = -g(x_k)$ and go back to Step 1.a.

2. Compute

$$Red_k = \begin{cases} f(x_k) - f(x_k + \lambda_k p_k) & \text{if Step 1.b. has been performed} \\ f(x_{m(k)}) - f(x_k + \lambda_k p_k) & \text{otherwise} \end{cases}$$

$$Pred_k = \begin{cases} f(x_k) - m_k(p_k^c) & \text{if Step 1.c. has been performed} \\ f(x_k) - m_k(p_k) & \text{otherwise} \end{cases}$$

$$\hat{\rho}_k = \frac{Red_k}{Pred_k}$$

3. Set

$$x_{k+1} = \begin{cases} x_k + \lambda_k p_k & \text{if } \hat{\rho}_k \geq \eta_1, \\ x_k & \text{otherwise.} \end{cases}$$

4. Set

$$\sigma_{k+1} \in \begin{cases} (0, \sigma_k] & \text{if } \hat{\rho}_k \geq \eta_2 & \text{(very successful iteration),} \\ [\sigma_k, \gamma_1\sigma_k) & \text{if } \eta_1 \leq \hat{\rho}_k \leq \eta_2 & \text{(successful iteration),} \\ [\gamma_1\sigma_k, \gamma_2\sigma_k] & \text{otherwise} & \text{(unsuccessful iteration).} \end{cases}$$

It's straightforward that in this case the acceptance criterion allows that an iterate can be accepted even if it doesn't result in an improvement of the objective function.
If conditions (4.3) and (4.4) are satisfied, in Step 1.a. we first perform an ex-

trapolation on the direction of the trial step p_k. This extrapolation is limited by a maximum number L of inner iterations.
After that a non-monotone line search is performed over the direction previous computed. Condition (4.2) guarantees a "sufficient decrease". If one of conditions (4.3) and (4.4) fails we check (4.5) in Step 1.b. that requires that the trial step produces a sufficient decrease on the real objective function; if this happens we perform a standard monotone ARC iteration. If even this condition is not satisfied we force conditions (4.3) and (4.4) to be satisfied using the anti-gradient as trial step and we perform the strategy of Step 1.a. Since in this case is not guaranteed that the anti-gradient is a minimizer as good as the Cauchy point, we have to take care about it in $\hat{\rho}_k$ updating procedure performed in Step 2.

4.2 Convergence analysis

Let's first recall two useful Lemmas from [9] on the predicted reduction given by the minimizer of the subproblem.

Lemma 4.1. *[9, Lemma 2.1] Suppose that the step p_k satisfies $m_k(p_k) \leq m_k(p_k^c)$. Then for all $k \geq 0$*

$$f(x_k) - m_k(p_k) \geq \frac{\|g(x_k)\|}{6\sqrt{2}} \min\left\{\frac{\|g(x_k)\|}{1+\|B_k\|}, \frac{1}{2}\sqrt{\frac{\|g(x_k)\|}{\sigma_k}}\right\}. \tag{4.6}$$

Lemma 4.2. *[9, Lemma 2.2] Suppose that the step p_k satisfies $m_k(p_k) \leq m_k(p_k^c)$ and that $\|B_k\| \leq \kappa_B$ for all $k \geq 0$. Then*

$$\|p_k\| \leq \frac{3}{\sigma_k} \max\left\{\kappa_B, \sqrt{\sigma_k \|g(x_k)\|}\right\}. \tag{4.7}$$

We are now able to prove a result analogous to [9, Lemma 2.3].

Lemma 4.3. *Suppose $f \in C^1(\mathbb{R}^n)$ and $\|B_k\| \leq \kappa_B$ for all $k \geq 0$. If exists an infinite set $\mathcal{I}$ and $\epsilon > 0$ such that*

$$\|g(x_k)\| \geq \epsilon \ \forall\ k \in \mathcal{I} \textit{ and } \sqrt{\frac{\|g(x_k)\|}{\sigma_k}} \longrightarrow 0 \textit{ for } k \to \infty,\ k \in \mathcal{I} \tag{4.8}$$

then

$$\|p_k\| \leq 3\sqrt{\frac{\|g(x_k)\|}{\sigma_k}} \ \forall\ k \in \mathcal{I} \textit{ sufficiently large.} \tag{4.9}$$

Additionally, if the subsequence $\{x_k\}_{k\in\mathcal{I}}$ is convergent, then for each $k \in \mathcal{I}$, k sufficiently large, the iteration is very successful.

Proof. Since (4.8) implies

$$\sqrt{\sigma_k \|g(x_k)\|} = \|g(x_k)\| \sqrt{\frac{\sigma_k}{\|g(x_k)\|}} \geq \epsilon \sqrt{\frac{\sigma_k}{\|g(x_k)\|}} \longrightarrow \infty$$

for $k \in \mathcal{I}$, $k \to \infty$, the inequality (4.7) asymptotically becomes (4.9), and hence the first part of the Lemma is proved.
A simple Taylor expansion of $f(x_k + \lambda_k p_k)$ around x_k gives that for each k

$$\begin{aligned} f(x_k + \lambda_k p_k) - m_k(p_k) &= \lambda_k (g(\xi_k) - g(x_k))^T p_k - \frac{\lambda_k^2}{2} p_k^T B_k p_k - \frac{\sigma_k \lambda_k^3}{3}\|p_k\|^3 \\ &\leq \lambda_k (g(\xi_k) - g(x_k))^T p_k - \frac{\lambda_k^2}{2} p_k^T B_k p_k \end{aligned} \tag{4.10}$$

for some $\xi_k \in (x_k, x_k + \lambda_k p_k)$. By employing (4.9) we obtain

$$f(x_k+\lambda_k p_k)-m_k(p_k) \leq 3\sqrt{\frac{\|g(x_k)\|}{\sigma_k}}\left\{\lambda_k\|g(\xi_k)-g(x_k)\| + \frac{3\lambda_k^2\kappa_B}{2}\sqrt{\frac{\|g(x_k)\|}{\sigma_k}}\right\} \tag{4.11}$$

for all $k \in \mathcal{I}$ sufficiently large.
Since $f(x_{m(k)}) \geq f(x_k)$ and $m_k(p_k^c) \geq m_k(p_k)$ for each k, we have that the ratio $\hat{\rho}_k$ used in the acceptance step of Algorithm NMARC can be bounded from below by

$$\hat{\rho}_k \geq \frac{f(x_k) - f(x_k + \lambda_k p_k)}{f(x_k) - m_k(p_k)}.$$

The inequality

$$\frac{f(x_k) - f(x_k + \lambda_k p_k)}{f(x_k) - m_k(p_k)} \geq \eta_2$$

holds if and only if

$$r_k = f(x_k + \lambda_k p_k) - m_k(p_k) + (1-\eta_2)[m_k(p_k) - f(x_k)] \leq 0. \tag{4.12}$$

In order to get that we prove for each $k \in \mathcal{I}$, k sufficiently large, the iteration is very successful, we'll show that (4.12) holds if $\{x_k\}_{k\in\mathcal{I}}$ converges.
Now from (4.11) and Lemma 4.1 we obtain that

$$r_k \leq 3\sqrt{\frac{\|g(x_k)\|}{\sigma_k}}\left\{\lambda_k\|g(\xi_k)-g(x_k)\| + \frac{3\lambda_k^2\kappa_B}{2}\sqrt{\frac{\|g(x_k)\|}{\sigma_k}} - \epsilon\frac{1-\eta_2}{12\sqrt{2}}\right\}. \tag{4.13}$$

Note that λ_k is bounded from above by α^L and hence we cannot have $\lambda_k \to \infty$. Since $f \in C^1(\mathbb{R}^n)$ and $\xi_k \in (x_k, x_k + \lambda_k p_k)$ and due to the fact that (4.9) implies $p_k \longrightarrow 0$, we obtain that the right term of (4.13) asymptotically becomes

$$-3\epsilon\sqrt{\frac{\|g(x_k)\|}{\sigma_k}} \cdot \frac{1-\eta_2}{12\sqrt{2}} < 0$$

and hence for each $k \in \mathcal{I}$, k sufficiently large

$$\hat{\rho}_k \geq \frac{f(x_k) - f(x_k + \lambda_k p_k)}{f(x_k) - m_k(p_k)} \geq \eta_2$$

and then the iteration is very successful. □

The previous Lemma is crucial to prove that the Algorithm NMARC stops to produces successful iteration only if a stationary point has been found.

Lemma 4.4. *Let $f \in C^1(\mathbb{R}^n)$ and $\|B_k\| \leq \kappa_B$ for all $k \geq 0$ hold. Suppose that Algorithm NMARC produces only finitely many successful iterations. Then $x_k = x_*$ for all sufficiently large k and x_* is a critical point.*

Proof. Let k_0 be the last successful iteration. Then for each $j \in \mathbb{N}$ we have that $x_{k_0+j} = x_{k_0} := x_*$ due to the structure of Algorithm NMARC. In addition to this we have that $\sigma_k \to \infty$ as $k \to \infty$ since all iteration that follow k_0 are unsuccessful and so σ is increased by at least a fraction γ_1. If $\|g(x_*)\| = \epsilon > 0$, we have that for Lemma 4.3 each sufficiently large iteration is very successful, that contradicts the fact that $k_0 + j$ is unsuccessful for each $j \in \mathbb{N}$. This means that must be $\|g(x_*)\| = 0$. □

Before proving the convergence properties of Algorithm NMARC we need to focus on the "sufficient reduction" guaranteed at each successful iteration produced by the the algorithm.

Combining together the conditions (4.3) and (4.4) we have that

$$\begin{aligned} g(x_k)^T p_k &\le -c_1 \|g(x_k\|^2 \\ &\le -\frac{c_1}{c_2^2}\|p_k\|^2. \end{aligned}$$

Employing the previous inequality in (4.2) we obtain that at each successful iteration in which Step 1.a. is performed (and this includes also iterations that reach Step 1.c.) we have

$$\begin{aligned} f(x_{k+1}) = f(x_k + \lambda_k p_k) &\le f(x_{m(k)}) + \lambda_k \beta g(x_k)^T p_k \\ &\le f(x_{m(k)}) - \beta \frac{c_1}{c_2^2} \lambda_k \|p_k\|^2 \end{aligned}$$

and since $\lambda_k < \alpha^L$ is straightforward that

$$\begin{aligned} f(x_{k+1}) &\le f(x_{m(k)}) - \beta \frac{c_1}{c_2^2 \alpha^L} \|\lambda_k p_k\|^2 \\ &= f(x_{m(k)}) - \bar{\sigma}(\|x_{k+1} - x_k\|) \end{aligned} \tag{4.14}$$

where

$$\bar{\sigma}(t) = \beta \frac{c_1}{c_2^2 \alpha^L} t^2.$$

is a forcing function.

On the other hand we have that in Step 1.b.

$$f(x_{m(k)}) - f(x_k + \lambda_k p_k) \le \hat{\sigma}(\|\lambda_k p_k\|) \tag{4.15}$$

since $f(x_{m(k)}) \ge f(x_k)$ and $\lambda_k = 1$.

Hence due to (4.14) and (4.15) we can state that

$$f(x_{m(k)}) - f(x_k + \lambda_k p_k) \le \sigma(\|x_{k+1} - x_k\|) \tag{4.16}$$

holds for each successful iteration, where

$$\sigma(t) = \max\{\bar{\sigma}(t), \hat{\sigma}(t)\}.$$

It's trivial to verify that the function σ is itself a forcing function.

We can now state the main result of this Chapter, that guarantees the convergence of Algorithm NMARC under suitable assumptions.

Theorem 4.1. *Let $f \in C^1(\mathbb{R}^n)$ be bounded from below, $\|B_k\| \leq \kappa_B$ for all $k \geq 0$, and suppose that f is Lipschitz continuous over the compact set $\mathcal{L}_0 = \{x \in \mathbb{R}^n \ : \ f(x) \leq f(x_0)\}$. Then*

(i) $\{f(x_{m(k)})\}$ and $\{f(x_k)\}$ converge to the same limit;

(ii) $\{x_k\}$ contains a subsequence that converges to a stationary point.

Proof. The result is guaranteed by Lemma 4.4 if the number of successful iterations is finite.
Suppose now that the set of successful iteration $\mathcal{S}$ is infinite. Let $k \in \mathcal{S}$. For easiness of notation we enumerate the iterations that belong to $\mathcal{S}$ with $\{1, 2, \ldots\}$. From the definition of $m(k)$ we have that for k large enough

$$\begin{aligned} f(x_{m(k+1)}) &= \max_{j \in \mathcal{S}_{k+1,M}} f(x_j) \\ &\leq \max_{j \in \mathcal{S}_{k,M} \cup \{k+1\}} f(x_j) \\ &= \max \left\{ \max_{j \in \mathcal{S}_{k,M}} f(x_j), f(x_{k+1}) \right\} = f(x_{m(k)}). \end{aligned}$$

This means that the sequence $\{f(x_{m(k)})\}$ is monotonically non increasing and, since $f(x_{m(0)}) = f(x_0)$ we have that each point of the sequence $\{x_k\}$ is contained in the level set $\mathcal{L}_0$ that is compact. Hence $\{f(x_{m(k)})\}$, that is bounded from below, converges to a value f_*.
We prove by induction on j that

$$\lim_{k \in \mathcal{S}, k \to \infty} \|x_{m(k)-j+1} - x_{m(k)-j}\| = 0, \tag{4.17}$$

$$\lim_{k \in \mathcal{S}, k \to \infty} f(x_{m(k)-j}) = \lim_{k \in \mathcal{S}, k \to \infty} f(x_{m(k)}) \tag{4.18}$$

where k is supposed to be large enough to have $m(k) \geq k - M > 1$. The condition (4.16) ensures that

$$f(x_{m(k)}) \leq f(x_{m(m(k)-1)}) - \sigma(\|x_{m(k)} - x_{m(m(k)-1)}\|). \tag{4.19}$$

The limit for $k \to \infty$ gives that $\sigma(\|x_{m(k)} - x_{m(m(k)-1)}\|) \to 0$, and hence due to the definition of forcing function we obtain (4.17) for $j = 1$. Consequently (4.18) for $j = 1$ is straightforward since $f \in C^1(\mathbb{R}^n)$. Let (4.18) be valid for a fixed $j > 1$. The condition (4.16) gives that

$$f(x_{m(k)-j}) \leq f(x_{m(m(k)-j-1)}) - \sigma(\|x_{m(k)-j} - x_{m(k)-j-1}\|).$$

The limit for $k \to \infty$ and the inductive hypothesis give that

$$\lim_{k \to \infty} \|x_{m(k)-j} - x_{m(k)-j-1}\| = 0$$

and hence, since $f \in C^1(\mathbb{R}^n)$ and employing (4.18),

$$\lim_{k \in \mathcal{S}, k \to \infty} f(x_{m(k)-j-1}) = \lim_{k \in \mathcal{S}, k \to \infty} f(x_{m(k)}).$$

This completes the induction.
Hence (4.17) and (4.18) must be valid even if we consider $M(k) = m(k+M+1)$ instead of $m(k)$. In addiction to this, for k sufficiently large we can write

$$\begin{aligned} x_{M(k)} &= x_k + (x_{k+1} - x_k) + \ldots + (x_{M(k)} - x_{M(k)-1}) \\ &= x_k + \sum_{j=1}^{M(k)-k} (x_{M(k)-j+1} - x_{M(k)-j}). \end{aligned} \tag{4.20}$$

Now we obtain from (4.18) and (4.20) that

$$\lim_{k\to\infty} \|x_k - x_{M(k)}\| = 0.$$

Since f is Lipschitz continuous

$$\lim_{k\to\infty} f(x_k) = \lim_{k\to\infty} f(x_{M(k)}) = \lim_{k\to\infty} f(x_{m(k+M+1)}) = f_*$$

and hence, due to (4.16), we have that part (i) of the Theorem is proved.
The conditions (4.2) and (4.3) give that for every successful iteration in which Step 1.b. is not performed

$$f(x_{m(k)}) - f(x_{k+1}) \geq \lambda_k \beta c_1 \|g(x_k)\|. \tag{4.21}$$

Therefore if the subsequence of iterates in which Step 1.b. is not performed is infinite we have that that subsequence converges to a stationary point.
If starting from a certain $k_0 > 0$ the algorithm performs only Step 1.b. this means that the monotone version of ARC is applied starting from iterate k_0 and hence part (ii) of the Theorem is guaranteed by [9, Theorem 2.5]. □

If the gradient is of f is uniformly continuous on the sequence of iterates $\{x_k\}$, then we are able to prove the following strong convergence result.

Theorem 4.2. *Let $f \in C^1(\mathbb{R}^n)$ be bounded from below and $\|B_k\| \leq \kappa_B$ for all $k \geq 0$, suppose that f is Lipschitz continuous over the compact set $\mathcal{L}_0 = \{x \in \mathbb{R}^n : f(x) \leq f(x_0)\}$, and suppose that the gradient g is uniformly continuous on the sequence of iterates $\{x_k\}$. Then*

$$\lim_{k\to\infty} \|g(x_k)\| = 0. \tag{4.22}$$

Proof. Let first note that only successful iterations have effect on the sequence of gradients $\{g(x_k)\}$ since on all unsuccessful iterations the gradient remains constant.
If Algorithm NMARC produces only finitely many successful iteration, the result is guaranteed by Lemma 4.4.
Suppose hence that the set of successful iterations $\mathcal{S}$ is infinite. If the subset $\hat{\mathcal{S}} \subseteq \mathcal{S}$ of successful iterations obtained after that Step 1.b. has been performed is finite, it means that after a certain iteration k_0 the successful iterations necessarily follow Step 1.a. (we recall that Step 1.c. itself leads to Step 1.a.). In

this case (4.22) follows directly from inequality (4.21).
On the other hand, if $\mathcal{S}\setminus\hat{\mathcal{S}}$ is finite, it means that after a certain iteration k_0 the only successful iterations follows Step 1.b. and hence the standard monotone ARC algorithm is applied. Therefore we obtain that (4.22) is valid from [9, Corollary 2.6].
It remains to show that the sequence of gradient norms evaluated on the iterates produced by the Algorithm NMARC converges to 0 even if both $\hat{\mathcal{S}}$ and $\mathcal{S}\setminus\hat{\mathcal{S}}$ are infinite subsets. Suppose by contradiction that (4.22) is not valid and hence that exist an infinite subsequence $\{t_i\}\subset\mathcal{S}$ and a scalar $\epsilon>0$ such that

$$\|g_{t_i}\|\geq 2\epsilon\ \forall i>0. \tag{4.23}$$

We show that $\{t_i\}$ cannot contain an infinite subsequence $\{\hat{t}_i\}$ of successful iterates where Step 1.a. is applied. The conditions (4.2) and (4.3) give that

$$f(x_{m(\hat{t}_i)})-f(x_{\hat{t}_i})\geq\lambda_{\hat{t}_i}\beta c_1\|g(x_{\hat{t}_i})\|. \tag{4.24}$$

for each $i>0$. Due to part (i) of Theorem 4.1, $\{f(x_{m(k)})\}$ and $\{f(x_k)\}$ converge to the same value and hence the limit for $i\to\infty$ of (4.24) give

$$\|g(x_{\hat{t}_i})\|\longrightarrow 0,$$

that contradicts (4.23). For this reason we can suppose that $\{t_i\}\subset\hat{\mathcal{S}}$.
For all $i>0$ let $l_i>t_i$ be the first successful iteration in which $\|g(x_{l_i})\|<\epsilon$ and let $\mathcal{K}=\{k\in\mathcal{S}:t_i\leq k<l_i\}$. Hence for all $i>0$ and for all $t_i\leq k<l_i$ we have that

$$\|g(x_k)\|\geq\epsilon. \tag{4.25}$$

Suppose by contradiction that $\mathcal{K}$ contains an infinite subsequence $\mathcal{I}\subseteq\mathcal{K}$ of iterates in which Step 1.a. is applied. With the same argument as above we can prove that

$$\|g(x_k)\|\longrightarrow 0 \text{ for } k\in\mathcal{I},$$

that contradicts (4.25).
Hence without loss of generality we can assume that the whole set $\mathcal{K}$ is contained in the set $\hat{\mathcal{S}}$ of successful iterates obtained after that Step 1.b. has been performed. Due to (4.25) and by Lemma 4.1 we can say that for all $k\in\mathcal{K}$

$$f(x_k)-f(x_{k+1})\geq\frac{\eta_1\epsilon}{6\sqrt{2}}\cdot\min\left\{\frac{\epsilon}{1+\kappa_B},\frac{1}{2}\sqrt{\frac{\|g(x_k)\|}{\sigma_k}}\right\}.$$

Since $\{f(x_k)\}$ converges by Theorem 4.1, we obtain that the previous inequality asymptotically becomes

$$f(x_k)-f(x_{k+1})\geq\frac{\eta_1\epsilon}{12\sqrt{2}}\sqrt{\frac{\|g(x_k)\|}{\sigma_k}}$$

for $k \in \mathcal{K}$ sufficiently large and

$$\sqrt{\frac{\|g(x_k)\|}{\sigma_k}} \longrightarrow 0$$

as $k \to \infty$, $k \in \mathcal{K}$. So the hypothesis of Lemma 4.3 are satisfied by the set $\mathcal{K}$ and hence

$$f(x_k) - f(x_{k+1}) \geq \frac{\eta_1 \epsilon}{36\sqrt{2}} \|p_k\| \tag{4.26}$$

for all $t_i \leq k < l_i$, $k \in \mathcal{S}$, i sufficiently large. The sum of (4.26) over k and the triangular inequality give

$$\frac{36\sqrt{2}}{\eta_1 \epsilon}(f(x_{t_i}) - f(x_{l_i})) \geq \sum_{k=t_i, k \in \mathcal{S}}^{l_i - 1} \|p_k\| = \sum_{k=t_i}^{l_i-1} \|x_{k+1} - x_k\| \geq \|x_{t_i} - x_{l_i}\|,$$

for all i sufficiently large. Since $\{f(x_k)\}$ converges, $\{f(x_{t_i}) - f(x_{l_i})\}$ converges to 0 and hence also $\|x_{t_i} - x_{l_i}\| \to 0$. The fact that g is uniformly continuous on the sequence of iterates implies that $\|g(x_{t_i}) - g(x_{l_i})\| \to 0$ but this is a contradiction since $\|g(x_{t_i}) - g(x_{l_i})\| \geq \|g(x_{t_i})\| - \|g(x_{l_i})\| \geq \epsilon$ for all $i > 0$. $\square$

4.3 Numerical results

Test problems

We made numerical experiments using the Luksan's collection, available online at `http://www.cs.cas.cz/luksan/test.html`. The Hessian/vector product for each one of these problems is obtained by finite differences.

4.3.1 Implementation details

We tested Algorithm 4.1 against the standard non-monotone Algorithm 2.1 on an Intel Core i7 CPU 870 @ 2.93GHz. The code has been written in Fortran90, with B_k set to the true Hessian $H(x_k)$. The parameters defining the original ARC method (Algorithm 2.1) have been chosen as described in [9]. The additional parameters of Algorithm 4.1 have been chosen as follows:

$$\begin{aligned} L &= 5 \\ M &= 5 \\ c_1 &= 10^{-4} \\ c_2 &= 10^{2} \\ \omega &= 0.75 \\ \beta &= 0.5 \\ \alpha &= 2 \end{aligned}$$

In addition we chose $\hat{\sigma} = \bar{\sigma}$. A maximum cpu time of 500 seconds has been imposed. We consider failures when the norm of the trial step is less than 10^{-12} or the non-monotone line search in Step 1.a. of Algorithm 4.1 fails.

Implementation details on subproblem solution

We tested the behavior of Algorithm 4.1 for both solvers described in Section 3.4.1. For NMGRAD we set the early stopping parameter $N = 5$.

4.3.2 Numerical results with solver GLRT

In Tables 4.1 and 4.2 we report the results obtained using GLRT as solver for Algorithm 2.1 (ARC-GLRT) and 4.1 (NMARC-GLRT). The symbols n_i, n_f, n_g, f^*, and cpu denote the number of iterations, the number of function evaluations, the number of gradient evaluations, the final objective function value and the cpu time (in seconds), respectively.
In Figure 4.1 we compare the obtained results by means of the performance profiles proposed in [11] in term of cpu time.

Table 4.1: Results obtained by ARC-GLRT on the Luksan's problems

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 1	1704	1705	1522	9.93e-015	2.2561409
LUKSAN 2	917	918	831	481.4295925421	4.4122758
LUKSAN 3	20	21	21	1.31e-009	2.40e-002
LUKSAN 4	26	27	27	269.4995434872	0.12800701
LUKSAN 5	14	15	15	5.75e-012	4.80e-002
LUKSAN 6	15	16	16	6.63e-011	0.148009
LUKSAN 7	13	14	14	336.9371812776	5.60e-002
LUKSAN 8	Produced step with norm less than 10^{-12}				
LUKSAN 9	50	51	51	316.4361406766	0.33602098
LUKSAN 10	208	209	75	-133.25	32.326019
LUKSAN 11	60	61	55	10.7765878896	5.20e-002
LUKSAN 12	27	28	28	99.8933068389	2.00e-002
LUKSAN 13	13	14	14	1.67e-024	2.40e-002
LUKSAN 14	1	2	2	2.15e-008	0.18401101
LUKSAN 15	30	31	31	1.9240159855	9.2085752
LUKSAN 16	18	19	19	-427.4044763748	0.70804399
LUKSAN 17	27	28	28	-3.80e-002	1.0400651
LUKSAN 18	19	20	20	-2.46e-002	0.93605798
LUKSAN 19	55	56	56	59.5986241321	1.8841181
LUKSAN 20	86	87	81	-1.0001352001	2.876179
LUKSAN 21	42	43	43	2.1386637718	4.796299
LUKSAN 22	698	699	94	1	65.028061
LUKSAN 23	63	64	47	23.4453429939	0.67604196
LUKSAN 24	Produced step with norm less than 10^{-12}				
LUKSAN 25	36	37	37	4.19e-022	1.60e-002
LUKSAN 26	31	32	32	6.27e-009	2.00e-002
LUKSAN 27	52	53	40	4.84e-003	2.00e-002
LUKSAN 28	37	38	38	2.94e-022	2.00e-002
LUKSAN 29	3	4	4	1.95e-013	0.12400699
LUKSAN 30	8	9	9	2.74e-011	2.1121318
LUKSAN 31	13	14	14	7.90e-013	2.40e-002
LUKSAN 32	14	15	15	3.26e-013	2.00e-002
LUKSAN 33	14	15	15	2.09e-012	7.60e-002
LUKSAN 34	68	69	29	60705.3688270019	7.20e-002
LUKSAN 35	42	43	37	1.76e-007	4.4682789
LUKSAN 36	312	313	293	2216.4587065611	1.3800861
LUKSAN 37	23	24	18	191.1662898661	0.59603697
LUKSAN 38	3799	3800	3435	4.86e-014	6.776423
LUKSAN 39	108	109	75	22237.5481131278	0.31201899
LUKSAN 40	55	56	23	131234.018444958	0.144008
LUKSAN 41	Produced step with norm less than 10^{-12}				
LUKSAN 42	66	67	17	18.1763145769	1.196074
LUKSAN 43	23	24	20	2.5110967742	8.7085447
LUKSAN 44	66	67	64	6.15e-010	4.2042627
LUKSAN 45	90	91	90	5.17e-013	0.472029
LUKSAN 46	47	48	48	6.23e-013	0.20801301
LUKSAN 47	16461	16462	16461	3.70e-011	121.17558
LUKSAN 48	67	68	43	650.77362946	0.14400901
LUKSAN 49	27	28	24	4486.9702387635	0.172011
LUKSAN 50	38	39	30	5.05e-014	9.388587
LUKSAN 51	46	47	41	1.44e-010	0.160009

continued on next page

Table 4.1 – continued from previous page

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 52	12	13	13	7.0862816479	8.00e-002
LUKSAN 53	1551	1552	1481	3135.5211263348	94.153885
LUKSAN 54	18	19	19	3.08e-008	4.80e-002
LUKSAN 55	757	758	550	0.4998163336	3.708231
LUKSAN 56	529	530	425	0.315829621	3.3642101
LUKSAN 57	113	114	114	12246.0634198099	3.20e-002
LUKSAN 58	39	40	40	4.19e-007	2.80e-002
LUKSAN 59	14	15	15	1.81e-012	2.80e-002
LUKSAN 60	2440	2441	2441	8.62e-005	1.5040941
LUKSAN 61	447	448	436	8.97e-014	0.23601501
LUKSAN 62	Not solved within 500 secs. CPU time				
LUKSAN 63	Not solved within 500 secs. CPU time				
LUKSAN 64	108	109	109	3.77e-007	28.33777
LUKSAN 65	529	530	530	1.25e-004	196.99231
LUKSAN 66	111	112	112	2.98e-007	35.682228
LUKSAN 67	92	93	93	3.96e-010	11.02469
LUKSAN 68	151	152	152	5.62e-011	3.828239
LUKSAN 69	Not solved within 500 secs. CPU time				
LUKSAN 70	60	61	61	1.22e-010	7.8164878
LUKSAN 71	39	40	40	9.88e-012	1.056066
LUKSAN 72	211	212	212	9.63e-009	99.366211
LUKSAN 73	Not solved within 500 secs. CPU time				
LUKSAN 74	34	35	35	1.28e-011	10.872681
LUKSAN 75	Not solved within 500 secs. CPU time				
LUKSAN 76	21	22	22	5.37e-018	8.00e-003
LUKSAN 77	10	11	11	2.64e-011	0.88005501
LUKSAN 78	0	1	1	4.98e-010	0
LUKSAN 79	41	42	42	1.69e-011	6.80e-002
LUKSAN 80	12	13	13	4.42e-013	6.00e-002
LUKSAN 81	11	12	12	2.87e-012	2.40e-002
LUKSAN 82	12	13	13	4.55e-020	1.20e-002
LUKSAN 83	64	65	65	5.76e-007	18.41715
LUKSAN 84	12	13	13	2.73e-013	2.80e-002
LUKSAN 85	Not solved within 500 secs. CPU time				
LUKSAN 86	11	12	12	3.90e-011	2.4561532
LUKSAN 87	71	72	72	5.72e-007	13.164823
LUKSAN 88	Not solved within 500 secs. CPU time				
LUKSAN 89	122	123	100	7.53e-007	47.294952
LUKSAN 90	0	1	1	1.66e-008	0
LUKSAN 91	108	109	109	7.40e-011	4.0282507
LUKSAN 92	215	216	216	3.07e-011	4.1962619

Table 4.2: Results obtained by NMARC-GLRT on the Luksan's problems

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 1	1526	3279	1521	1.90e-014	1.936121
LUKSAN 2	716	1989	706	530.8381597082	3.9122441
LUKSAN 3	16	40	17	2.25e-012	2.00e-002
LUKSAN 4	21	48	22	269.4995434872	0.120007

continued on next page

Table 4.2 – continued from previous page

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 5	12	27	13	7.75e-012	4.00e-002
LUKSAN 6	27	168	22	1.59e-011	0.25201499
LUKSAN 7	12	26	13	336.9371812776	6.00e-002
LUKSAN 8	27	87	28	761774.953705696	0.39602399
LUKSAN 9	10	36	11	316.4361406766	0.112007
LUKSAN 10	286	2070	125	-133.53	34.950184
LUKSAN 11	48	319	39	10.7765878895	4.40e-002
LUKSAN 12	10	27	11	99.8933068389	8.00e-003
LUKSAN 13	6	18	7	7.97e-014	1.60e-002
LUKSAN 14	1	3	2	2.15e-008	0.180011
LUKSAN 15	25	58	26	1.9240159855	7.7284827
LUKSAN 16	17	36	18	-427.4044763748	0.30801898
LUKSAN 17	21	51	22	-3.80e-002	0.896056
LUKSAN 18	16	37	17	-2.46e-002	0.86405396
LUKSAN 19	39	99	40	59.598624132	1.6041
LUKSAN 20	62	184	60	-1.0001352001	2.360147
LUKSAN 21	118	550	116	2.1386637718	7.8244886
LUKSAN 22	Nonmonotone line search failure				
LUKSAN 23	40	233	35	23.4453429939	0.66804099
LUKSAN 24	463	3305	399	3513623.16130011	1.056066
LUKSAN 25	15	45	16	2.56e-014	8.00e-003
LUKSAN 26	15	44	16	6.75e-008	2.00e-002
LUKSAN 27	31	92	32	4.84e-003	1.20e-002
LUKSAN 28	16	46	17	9.64e-022	1.20e-002
LUKSAN 29	3	7	4	1.95e-013	0.132008
LUKSAN 30	8	18	9	1.66e-012	2.0641291
LUKSAN 31	15	37	16	2.10e-013	2.40e-002
LUKSAN 32	13	28	14	7.22e-013	2.00e-002
LUKSAN 33	13	28	14	2.25e-012	8.00e-002
LUKSAN 34	26	118	24	60705.3688270049	3.20e-002
LUKSAN 35	44	152	43	3.88e-009	5.6043496
LUKSAN 36	54	365	39	2214.225553103	1.0320641
LUKSAN 37	31	125	28	191.1662898661	0.81204998
LUKSAN 38	3347	6957	3343	3.73e-026	7.1004429
LUKSAN 39	103	815	72	22287.9069170113	0.20001201
LUKSAN 40	32	126	30	131234.018444959	9.60e-002
LUKSAN 41	29	190	22	108.5178880501	8.2525158
LUKSAN 42	15	35	16	18.1763145769	0.36002198
LUKSAN 43	53	232	48	2.5110967742	14.164886
LUKSAN 44	55	172	53	8.20e-010	3.3642101
LUKSAN 45	Not solved within 500 secs. CPU time				
LUKSAN 46	31	80	32	2.22e-013	0.156009
LUKSAN 47	9059	26991	9054	3.91e-011	57.911617
LUKSAN 48	40	194	36	650.7654123463	0.16801001
LUKSAN 49	39	174	35	4486.9701258325	0.22801401
LUKSAN 50	Nonmonotone line search failure				
LUKSAN 51	35	119	34	3.18e-011	0.148009
LUKSAN 52	12	25	13	7.0862816479	7.60e-002
LUKSAN 53	1007	3452	972	3019.1929426356	94.621918
LUKSAN 54	11	36	12	1.99e-009	3.20e-002
LUKSAN 55	718	3835	708	4.9471582212	2.6361639
LUKSAN 56	428	2515	415	0.315829621	2.6281641

continued on next page

Table 4.2 – continued from previous page

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 57	27	83	28	12246.0634198099	1.20e-002
LUKSAN 58	17	56	18	1.60e-009	2.40e-002
LUKSAN 59	13	29	14	2.98e-013	2.40e-002
LUKSAN 60	68	209	69	9.20e-005	4.80e-002
LUKSAN 61	171	577	164	4.22e-015	0.108006
LUKSAN 62	Nonmonotone line search failure				
LUKSAN 63	Not solved within 500 secs. CPU time				
LUKSAN 64	75	197	76	3.09e-007	19.017189
LUKSAN 65	489	1047	486	1.25e-004	174.11089
LUKSAN 66	115	243	116	2.59e-007	35.602226
LUKSAN 67	77	203	78	6.78e-011	9.7326078
LUKSAN 68	96	268	97	2.71e-011	2.540159
LUKSAN 69	Not solved within 500 secs. CPU time				
LUKSAN 70	56	231	51	7.90e-011	7.8364887
LUKSAN 71	26	67	27	6.68e-012	0.71604401
LUKSAN 72	174	352	175	1.11e-008	77.884872
LUKSAN 73	Not solved within 500 secs. CPU time				
LUKSAN 74	16	50	17	5.75e-011	6.7604232
LUKSAN 75	3766	23357	3738	2.02e-009	345.86963
LUKSAN 76	7	22	8	1.25e-018	4.00e-003
LUKSAN 77	12	28	13	9.38e-012	1.0240639
LUKSAN 78	0	1	1	4.98e-010	0
LUKSAN 79	26	68	27	3.01e-012	6.40e-002
LUKSAN 80	12	25	13	4.33e-013	5.60e-002
LUKSAN 81	10	22	11	2.78e-012	3.20e-002
LUKSAN 82	6	18	7	1.82e-019	8.00e-003
LUKSAN 83	47	117	48	6.43e-007	12.048754
LUKSAN 84	10	22	11	9.00e-013	2.80e-002
LUKSAN 85	Not solved within 500 secs. CPU time				
LUKSAN 86	9	21	10	6.00e-011	2.388149
LUKSAN 87	55	136	56	5.44e-007	9.2165766
LUKSAN 88	Nonmonotone line search failure				
LUKSAN 89	82	257	79	8.90e-007	34.406147
LUKSAN 90	0	1	1	1.66e-008	0
LUKSAN 91	72	197	73	3.32e-011	2.832176
LUKSAN 92	125	359	126	5.71e-011	2.6361639

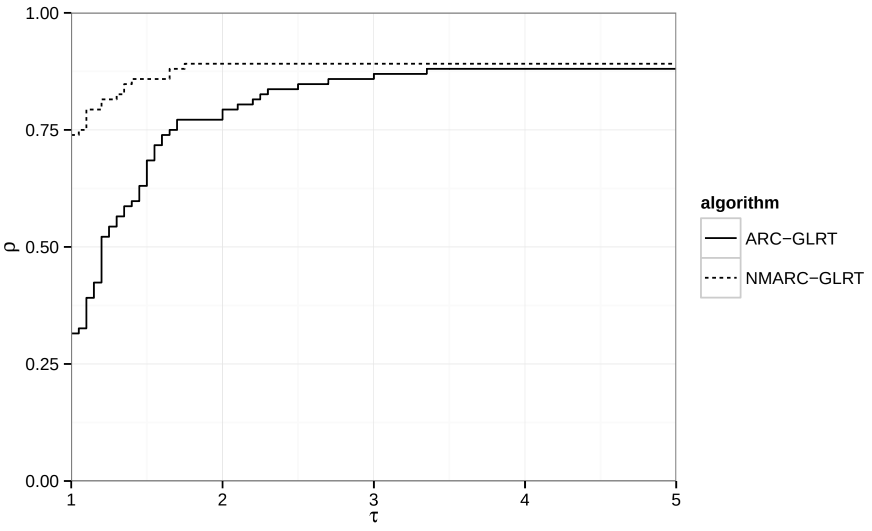

Figure 4.1: Performance profiles for ARC-GLRT and NMARC-GLRT on the Luksan's problems.

4.3.3 Numerical results with solver NMGRAD

In Tables 4.3 and 4.4 we report the results obtained using NMGRAD as solver for Algorithm 2.1 (ARC-NMGRAD) and 4.1 (NMARC-NMGRAD) and in Figure 4.1 we compare the obtained results by means of the cpu time performance profiles.

Table 4.3: Results obtained by ARC-NMGRAD on the Luksan's problems

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 1	1509	15840	1507	3.9866238543	5.5243449
LUKSAN 2	574	7091	555	343.5654989855	2.7161689
LUKSAN 3	16	255	17	5.12e-009	0.16001
LUKSAN 4	19	306	20	269.4995434872	0.26001599
LUKSAN 5	12	44	13	1.40e-011	5.20e-002
LUKSAN 6	13	83	14	3.38e-012	0.132008
LUKSAN 7	7	26	8	336.9371812776	3.60e-002
LUKSAN 8	Produced step with norm less than 10^{-12}				
LUKSAN 9	47	142	48	316.4361406766	0.55203396
LUKSAN 10	150	1861	66	-133.7099999998	13.528845
LUKSAN 11	39	412	38	10.7765878895	0.200012
LUKSAN 12	27	69	28	99.8933068388	3.20e-002
LUKSAN 13	13	37	14	5.23e-023	5.60e-002
LUKSAN 14	1	171	2	2.15e-008	0.112006
LUKSAN 15	1110	151866	1110	1.9240159871	275.02118
LUKSAN 16	7	160	8	-427.4044763749	0.24401501
LUKSAN 17	17	2274	18	-3.80e-002	1.368085
LUKSAN 18	16	2580	17	-2.46e-002	1.0480649
LUKSAN 19	30	3207	31	59.5986241328	2.100131
LUKSAN 20	46	2657	40	-1.0001351999	1.808112
LUKSAN 21	31	4566	32	2.1386637772	3.6962311
LUKSAN 22	384	8080	80	1	3.644227
LUKSAN 23	105	793	42	23.4453429938	0.71604401
LUKSAN 24	Not solved within 500 secs. CPU time				
LUKSAN 25	37	133	38	4.75e-023	5.20e-002
LUKSAN 26	28	162	29	3.86e-008	0.100006
LUKSAN 27	50	121	40	4.84e-003	5.20e-002
LUKSAN 28	36	73	37	3.45e-018	3.20e-002
LUKSAN 29	2	5	3	1.95e-013	0.21201299
LUKSAN 30	5	16	6	4.22e-017	2.5041568
LUKSAN 31	6	25	7	2.41e-018	2.40e-002
LUKSAN 32	7	24	8	5.67e-020	2.00e-002
LUKSAN 33	7	38	8	3.24e-013	4.40e-002
LUKSAN 34	21	225	20	60734.8550547264	0.148009
LUKSAN 35	48	6015	48	2.61e-006	22.793425
LUKSAN 36	218	628	202	2216.4587065611	0.46402898
LUKSAN 37	13	117	9	191.1662898661	0.34002098
LUKSAN 38	3559	31864	3360	7.17e-017	14.796924
LUKSAN 39	79	650	66	22237.5481131281	0.52803302
LUKSAN 40	18	135	17	131234.018444959	0.116007
LUKSAN 41	4	35	5	108.5178880501	1.292081
LUKSAN 42	13	45	8	18.1763145769	0.39602399
LUKSAN 43	12	540	13	2.5110967742	5.3483338

continued on next page

Table 4.3 – continued from previous page

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 44	35	818	30	5.29e-003	0.41602501
LUKSAN 45	3546	446376	3370	1.46e-017	248.93156
LUKSAN 46	43	1232	44	1.50e-014	1.6841049
LUKSAN 47	10113	102515	10114	1.02e-008	65.232079
LUKSAN 48	14	130	15	647.69613606	0.104006
LUKSAN 49	18	230	17	4486.9702387635	0.240015
LUKSAN 50	78	293	59	2.64e-021	11.340709
LUKSAN 51	48	673	34	4.18e-013	0.68404198
LUKSAN 52	10	38	11	4.72e-020	0.26001599
LUKSAN 53	933	29095	882	3098.5873877796	70.276398
LUKSAN 54	17	95	18	2.09e-008	6.00e-002
LUKSAN 55	621	8648	513	0.4998163336	5.9443712
LUKSAN 56	363	2366	315	0.315829621	1.468091
LUKSAN 57	100	349	101	12246.0634198102	0.120007
LUKSAN 58	38	145	39	5.17e-007	7.60e-002
LUKSAN 59	8	41	9	2.89e-012	2.80e-002
LUKSAN 60	Not solved within 500 secs. CPU time				
LUKSAN 61	308	12972	281	4.64e-010	8.7005434
LUKSAN 62	Not solved within 500 secs. CPU time				
LUKSAN 63	Not solved within 500 secs. CPU time				
LUKSAN 64	57	6864	58	1.27e-005	7.940496
LUKSAN 65	2239	411741	2240	2.31e-004	367.65497
LUKSAN 66	162	29237	163	6.34e-006	24.453527
LUKSAN 67	116	20722	117	6.43e-006	20.981312
LUKSAN 68	88	5127	89	6.38e-009	4.2282639
LUKSAN 69	Not solved within 500 secs. CPU time				
LUKSAN 70	69	10196	70	5.25e-007	9.6646042
LUKSAN 71	29	1104	30	1.37e-012	1.104069
LUKSAN 72	Not solved within 500 secs. CPU time				
LUKSAN 73	715	130935	716	3.72e-006	365.05881
LUKSAN 74	31	94	32	1.78e-014	17.825113
LUKSAN 75	2549	59812	1183	1.77e-005	21.597349
LUKSAN 76	21	59	22	2.72e-017	2.80e-002
LUKSAN 77	9	28	10	2.31e-013	1.628101
LUKSAN 78	0	1	1	4.98e-010	0
LUKSAN 79	35	166	36	1.85e-014	7.60e-002
LUKSAN 80	5	25	6	1.01e-016	3.60e-002
LUKSAN 81	6	26	7	1.73e-015	2.80e-002
LUKSAN 82	12	33	13	8.66e-019	2.40e-002
LUKSAN 83	25	3718	26	1.12e-005	3.436214
LUKSAN 84	7.00e+000	23	8	3.31e-013	2.40e-002
LUKSAN 85	Not solved within 500 secs. CPU time				
LUKSAN 86	9	27	10	1.05e-016	3.448215
LUKSAN 87	27	4014	28	1.07e-005	2.0081251
LUKSAN 88	Produced step with norm less than 10^{-12}				
LUKSAN 89	80	9570	72	7.50e-006	33.286079
LUKSAN 90	0	1	1	1.66e-008	0
LUKSAN 91	58	3758	59	3.08e-008	5.0563159
LUKSAN 92	135	4192	136	2.97e-009	4.0322509

Table 4.4: Results obtained by NMARC-NMGRAD on the Luksan's problems

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 1	1456	15694	1457	6.91e-013	4.8803039
LUKSAN 2	396	3995	397	121.9636793701	1.176073
LUKSAN 3	9	139	10	9.49e-009	6.80e-002
LUKSAN 4	12	246	13	269.4995434872	0.21201301
LUKSAN 5	9	44	10	1.07e-010	4.40e-002
LUKSAN 6	11	87	12	3.17e-012	0.13600801
LUKSAN 7	5	28	6	336.9371812776	4.00e-002
LUKSAN 8	5	33	6	761774.953705698	0.148009
LUKSAN 9	7	44	8	316.4361406768	0.112007
LUKSAN 10	Nonmonotone line search failure				
LUKSAN 11	33	323	34	10.7765878895	0.12800799
LUKSAN 12	10	50	11	99.8933068389	2.40e-002
LUKSAN 13	6	27	7	7.97e-014	2.80e-002
LUKSAN 14	1	172	2	2.15e-008	0.108006
LUKSAN 15	224	27517	225	1.9240159855	56.18351
LUKSAN 16	6	139	7	-427.4044763748	0.208012
LUKSAN 17	11	1823	12	-3.80e-002	1.1320701
LUKSAN 18	8	1458	9	-2.46e-002	0.60403699
LUKSAN 19	17	1998	18	59.5986241324	1.2960811
LUKSAN 20	26	2172	27	-1.0001351989	1.4200881
LUKSAN 21	28	2894	29	2.1386637719	2.2921429
LUKSAN 22	131	3410	109	1	1.28808
LUKSAN 23	32	465	30	23.4453429938	0.36802301
LUKSAN 24	Produced step with norm less than 10^{-12}				
LUKSAN 25	20	108	21	4.70e-021	2.80e-002
LUKSAN 26	13	90	14	2.48e-008	3.60e-002
LUKSAN 27	33	135	34	4.84e-003	4.00e-002
LUKSAN 28	16	248	17	1.72e-019	4.3802729
LUKSAN 29	2	7	3	1.95e-013	0.21201301
LUKSAN 30	4	17	5	5.02e-015	1.8601159
LUKSAN 31	4	24	5	2.91e-015	2.40e-002
LUKSAN 32	4	21	5	7.06e-018	1.60e-002
LUKSAN 33	7	53	8	2.54e-019	5.60e-002
LUKSAN 34	10	144	11	60734.8550547261	9.20e-002
LUKSAN 35	23	1860	24	8.35e-007	7.0244389
LUKSAN 36	28	193	29	2214.225553103	0.176011
LUKSAN 37	8	104	9	191.1662898661	0.288017
LUKSAN 38	3016	29250	3017	3.17e-017	11.944746
LUKSAN 39	40	660	40	22226.9546264401	0.52003199
LUKSAN 40	13	140	14	131234.018444959	0.104006
LUKSAN 41	4	22	5	108.5178880501	0.55603498
LUKSAN 42	6	28	7	18.1763145769	0.18401101
LUKSAN 43	9	353	10	2.5110967742	3.2082
LUKSAN 44	54	5896	55	8.75e-003	2.88818
LUKSAN 45	Not solved within 500 secs. CPU time				
LUKSAN 46	20	789	21	2.77e-012	0.69604301
LUKSAN 47	2241	45656	2242	2.32e-010	24.365522
LUKSAN 48	22	208	23	650.745506124	0.14400899
LUKSAN 49	21	258	22	4486.9702387635	0.356022
LUKSAN 50	8	83	9	26483.4791026691	4.852303
LUKSAN 51	1553	282867	1554	1.92e-004	147.82524

continued on next page

Table 4.4 – continued from previous page

Name	n_i	n_f	n_g	f^*	cpu(secs.)
LUKSAN 52	8	40	9	5.79e-016	0.108006
LUKSAN 53	552	14879	547	3184.5659338247	29.781862
LUKSAN 54	6	69	7	9.32e-011	6.40e-002
LUKSAN 55	679	8832	671	9.2277223676	4.8603029
LUKSAN 56	360	2821	356	6.3558202685	1.0640661
LUKSAN 57	19	120	20	12246.0634198099	3.60e-002
LUKSAN 58	16	111	17	3.54e-008	5.20e-002
LUKSAN 59	6	47	7	1.58e-016	2.40e-002
LUKSAN 60	72	1237	73	9.21e-005	33.10207
LUKSAN 61	95	4410	96	5.90e-012	2.0481279
LUKSAN 62	Nonmonotone line search failure				
LUKSAN 63	Not solved within 500 secs. CPU time				
LUKSAN 64	20	2081	21	9.40e-006	2.3561471
LUKSAN 65	549	101416	550	2.26e-004	89.933617
LUKSAN 66	49	8662	50	3.74e-006	7.1804481
LUKSAN 67	13	1753	14	3.33e-008	1.804112
LUKSAN 68	46	2237	47	9.54e-009	1.6601031
LUKSAN 69	Nonmonotone line search failure				
LUKSAN 70	29	3050	30	5.04e-007	2.844177
LUKSAN 71	14	641	15	3.96e-014	0.59603697
LUKSAN 72	155	28816	156	2.75e-007	34.92218
LUKSAN 73	148	26997	149	3.22e-006	74.060631
LUKSAN 74	10	57	11	4.71e-012	6.604413
LUKSAN 75	1541	35366	1542	1.27e-007	12.404775
LUKSAN 76	7	33	8	9.66e-018	1.20e-002
LUKSAN 77	6	28	7	1.86e-019	1.216076
LUKSAN 78	0	1	1	4.98e-010	0
LUKSAN 79	13	104	14	5.50e-014	4.80e-002
LUKSAN 80	5	30	6	9.45e-017	4.00e-002
LUKSAN 81	5	28	6	3.21e-015	2.40e-002
LUKSAN 82	6	27	7	4.28e-018	2.00e-002
LUKSAN 83	26	3883	27	3.71e-006	3.492218
LUKSAN 84	6	28	7	2.25e-018	2.80e-002
LUKSAN 85	Not solved within 500 secs. CPU time				
LUKSAN 86	4	19	5	3.92e-014	1.980123
LUKSAN 87	15	1804	16	1.17e-005	0.88805503
LUKSAN 88	Nonmonotone line search failure				
LUKSAN 89	85	11524	86	9.71e-006	25.517595
LUKSAN 90	0	1	1	1.66e-008	0
LUKSAN 91	36	2263	37	1.34e-009	2.904181
LUKSAN 92	67	2122	68	6.42e-009	1.728107

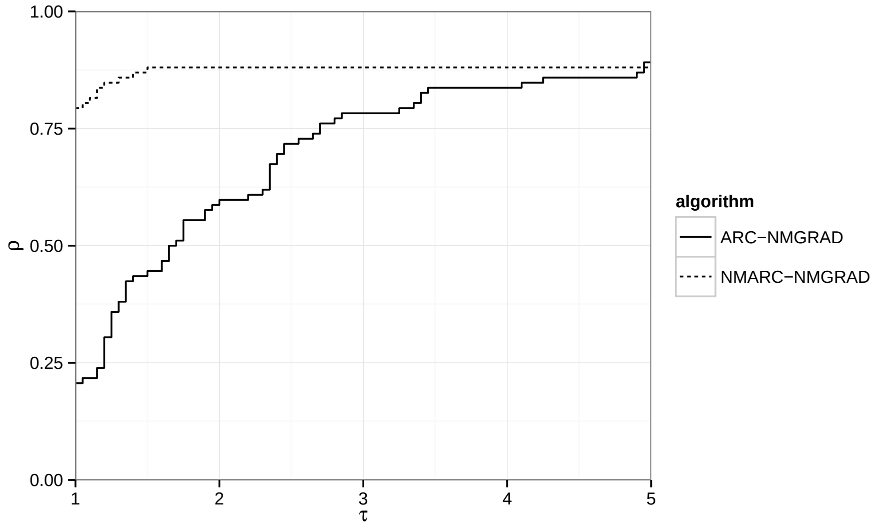

Figure 4.2: Performance profiles for ARC-NMGRAD and NMARC-NMGRAD on the Luksan's problems.

4.3.4 General comments and comparison between solvers GLRT and NMGRAD

For the sake of completeness in Figure 4.3 we report the cpu time performance profiles for NMARC-GLRT and NMARC-NMGRAD.

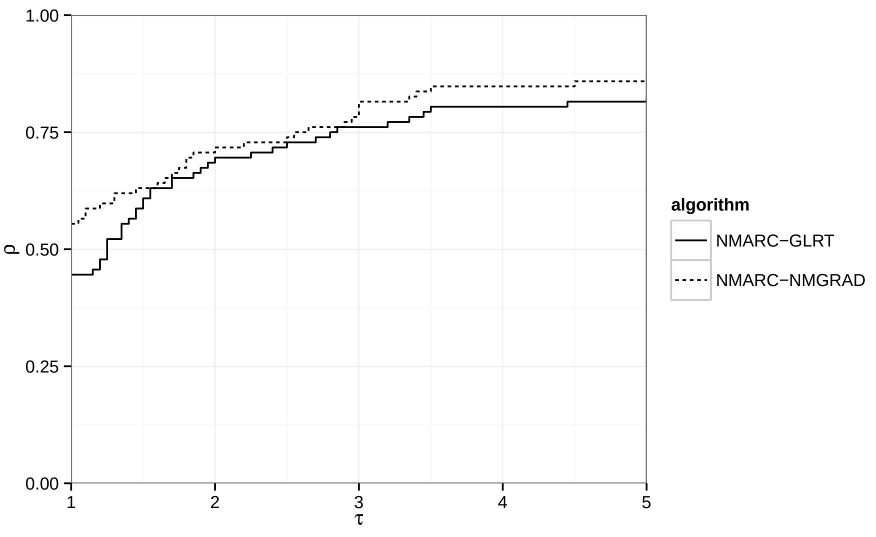

Figure 4.3: Performance profiles for NMARC-GLRT and NMARC-NMGRAD on the Luksan's problems.

Both Figures 4.1 and 4.2 show that introducing non-monotone techniques improves the efficiency of a standard ARC algorithm, independently by the solver used to obtain the approximate minimizer of the cubic subproblem.
This is due to the operations performed in Step 1.a.,that plays a key role in the efficiency of the new algorithm: in fact with the choice of the parameters presented above we observed that the other steps are performed just a couple of times. Hence we tested the effectiveness of both the operations performed in Step 1.a. of Algorithm 4.1 by isolating the extrapolation and the nonmonotone line search. For this purpose we compared the presented version of the algorithm with a version in which only the extrapolation is performed and another one that uses only nonmonotone line search in Step 1.a. Both solvers GLRT and NMGRAD have been used in these tests.

In Figures 4.4 and 4.5 we report the performance profiles relative to this comparison.

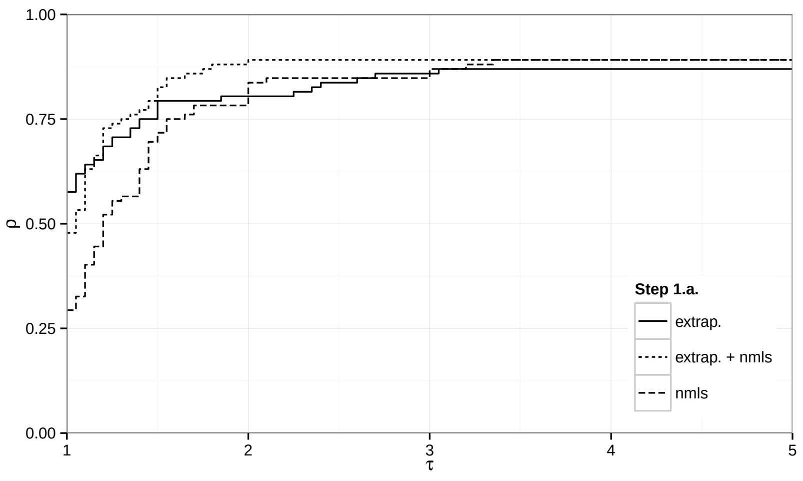

Figure 4.4: Performance profiles of the variants of Step 1.a. with solver GLRT on the Luksan's problems.

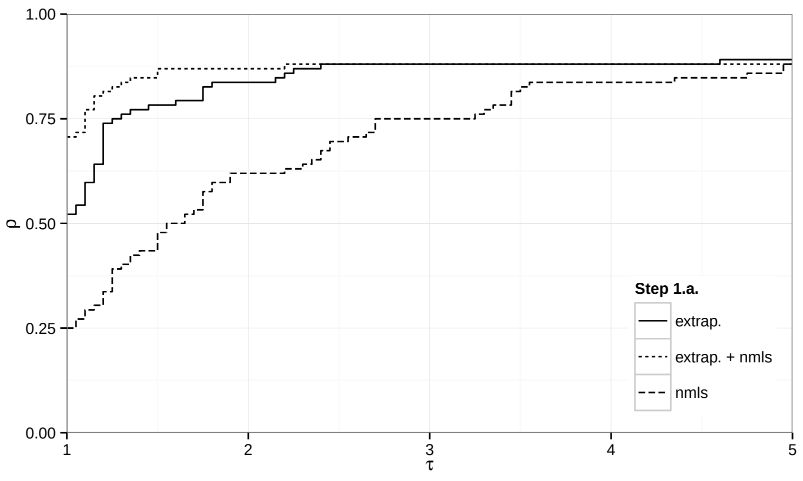

Figure 4.5: Performance profiles of the variants of Step 1.a. with solver NM-GRAD on the Luksan's problems.

Bibliography

[1] S. Bellavia, C. Cartis, N. Gould, B. Morini, and P. Toint. Convergence of a regularized euclidean residual algorithm for nonlinear least-squares. *SIAM Journal on Numerical Analysis*, 48(1):1–29, 2010.

[2] S. Bellavia and B. Morini. Strong local convergence properties of adaptive regularized methods for nonlinear least squares. *IMA Journal of Numerical Analysis*, 2014.

[3] Hande Benson and David Shanno. Interior-point methods for nonconvex nonlinear programming: cubic regularization. *Computational Optimization and Applications*, 58(2):323–346, 2014.

[4] Tommaso Bianconcini, Giampaolo Liuzzi, Benedetta Morini, and Marco Sciandrone. On the use of iterative methods in cubic regularization for unconstrained optimization. *Computational Optimization and Applications*, pages 1–23, 2014.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[6] C. Cartis, N. Gould, and P. Toint. On the complexity of steepest descent, newton's and regularized newton's methods for nonconvex unconstrained optimization problems. *SIAM Journal on Optimization*, 20(6):2833–2852, 2010.

[7] C. Cartis, N. I. M. Gould, and PH. L. Toint. An adaptive cubic regularization algorithm for nonconvex optimization with convex constraints and its function-evaluation complexity. *IMA Journal of Numerical Analysis*, 2012.

[8] Coralia Cartis, Nicholas I. M. Gould, and Philippe L. Toint. Adaptive cubic overestimation methods for unconstrained optimization. part ii: worst-case function-evaluation complexity. *Mathematical Programming*, 130:295–319, 2011.

[9] Coralia Cartis, Nicholas I. M. Gould, and Philippe L. Toint. Adaptive cubic regularization methods for unconstrained optimization. part i: motivation, convergence and numerical results. *Math. Program.*, 127:245–295, 2011.

[10] Andrew R. Conn, Nicholas I. M. Gould, and Philippe L. Toint. *Trust-region methods*. MPS-SIAM Series on Optimization. SIAM, Philadelphia (USA), 2000.

[11] Elizabeth D. Dolan and Jorge J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91:201–213, 2002.

[12] N. Gould, S. Lucidi, M. Roma, and P. Toint. Solving the trust-region subproblem using the lanczos method. *SIAM Journal on Optimization*, 9(2):504–525, 1999.

[13] Nicholas I. M. Gould and Dominique Orban. Cuter (and sifdec), a constrained and unconstrained testing environment, revisited. Technical report, ACM Transactions on Mathematical Software, 2001.

[14] Nicholas I. M. Gould, Dominique Orban, and Philippe L. Toint. Galahad, a library of thread-safe fortran 90 packages for large-scale nonlinear optimization. *ACM Trans. Math. Softw.*, 29(4):353–372, December 2003.

[15] Nicholas I. M. Gould, M. Porcelli, and Philippe L. Toint. Updating the regularization parameter in the adaptive cubic regularization algorithm. *Computational Optimization and Applications*, 53(Issue 1):1–22, September 2012.

[16] Andreas O. Griewank. The modification of newton's method for unconstrained optimization by bound- ing cubic terms. Technical Report NA/12, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, United Kingdom, 1981.

[17] 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.

[18] L. Grippo, F. Lampariello, and S. Lucidi. A truncated newton method with nonmonotone line search for unconstrained optimization. *Journal of Optimization Theory and Applications*, 60(3):401–419, 1989.

[19] Luigi Grippo and Marco Sciandrone. Nonmonotone globalization techniques for the barzilai-borwein gradient method. *Computational Optimization and Applications*, 23:143–169, 2002.

[20] Yuri Nesterov. Modified gauss-newton scheme with worst case guarantees for global performance. *Optimization Methods Software*, 22(3):469–483, June 2007.

[21] Yuri Nesterov and B. T. Polyak. Cubic regularization of newton method and its global performance. *Mathematical Programming*, 108(1):177–205, 2006.

[22] Philippe L. Toint. Nonlinear stepsize control, trust regions and regularizations for unconstrained optimization. *Optimization Methods and Software*, 28(1):82–95, 2013.

[23] Martin Weiser, Peter Deuflhard, and Bodo Erdmann. Affine conjugate adaptive newton methods for nonlinear elastomechanics. *Optimization Methods Software*, 22(3):413–431, June 2007.