Comput Optim Appl
DOI 10.1007/s10589-011-9446-7

Updating the regularization parameter in the adaptive cubic regularization algorithm

N.I.M. Gould · M. Porcelli · P.L. Toint

Received: 24 February 2011

Abstract The adaptive cubic regularization method (Cartis et al. in Math. Program. Ser. A 127(2):245–295, 2011; Math. Program. Ser. A. 130(2):295–319, 2011) has been recently proposed for solving unconstrained minimization problems. At each iteration of this method, the objective function is replaced by a cubic approximation which comprises an adaptive regularization parameter whose role is related to the local Lipschitz constant of the objective's Hessian. We present new updating strategies for this parameter based on interpolation techniques, which improve the overall numerical performance of the algorithm. Numerical experiments on large nonlinear least-squares problems are provided.

Keywords Unconstrained optimization · Cubic regularization · Numerical performance

Electronic supplementary material The online version of this article (doi:10.1007/s10589-011-9446-7) contains supplementary material, which is available to authorized users.

N.I.M. Gould
Computational Science and Engineering Department, Rutherford Appleton Laboratory, Chilton, Oxfordshire, OX11 0QX, England, UK
e-mail: nick.gould@sftc.ac.uk

M. Porcelli (✉) · P.L. Toint
Namur Center for Complex Systems (NAXYS), FUNDP-University of Namur, 61, rue de Bruxelles, 5000 Namur, Belgium
e-mail: margherita.porcelli@fundp.ac.be

P.L. Toint
e-mail: philippe.toint@fundp.ac.be

Published online: 21 December 2011

1 Introduction

We consider the unconstrained minimization problem

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

where f is a twice continuously differentiable function of the variables $x \in \mathbb{R}^n$. A simplistic method for solving this problem is to compute an improving step s_k by minimizing a quadratic Taylor-series model of the objective function around the current iterate x_k. Unfortunately, it is well-known that an iteration based on this simple idea may not always be well-defined (when the Taylor model is nonconvex), nor converge globally. These drawbacks may be overcome by restricting the model minimization to a trust region containing x_k [8]. Clearly, trust-region strategies may be considered as regularization techniques because they control the difference between two consecutive iterates by explicitly imposing a restriction on the stepsize.

The main motivation for this paper is a series of recent papers where alternative regularization strategies are introduced [2, 3, 7, 18, 21, 25]. These procedures are based on the minimization of quadratic or cubic models for the objective function in a neighbourhood implicitly defined by a regularization term that penalizes the step length. In particular, the adaptive cubic regularization (ARC) algorithm is proposed in [3] for solving problem (1.1). At each iteration, the objective function is locally replaced by a cubic approximation, in which third- and higher-order Taylor-series terms are replaced by a cubic regularization term, and an adaptive estimation of the local Lipschitz constant of the objective function's Hessian is employed. The method has been shown to have excellent global and local convergence properties and numerical experiments indicate that the new procedure may be competitive with the trust region approach when solving small-scale problems [3]. Additionally, and of theoretical interest, ARC possesses a better worst-case evaluation-complexity bound than its trust-region competitor [5, 6].

The purpose of this paper is twofold. Firstly, we propose alternative updating rules for the regularization parameter of the ARC algorithm which are based on interpolation techniques. In particular, in the trust-region case, the restriction on the stepsize is explicitly imposed by the trust-region constraint. By contrast, in the cubic regularization case the control on the stepsize is nonlinear and is defined implicitly. This suggests a need to design an efficient updating rule for the regularization parameter that is able to control the stepsize in a flexible way.

Secondly, we shall apply these ideas and report on extensive numerical experiments on the solution of large nonlinear least-squares problems, that is problems of the form

$$\min_{x\in\mathbb{R}^n} f(x) = \frac{1}{2}\|h(x)\|_2^2, \tag{1.2}$$

where $h : \mathbb{R}^n \to \mathbb{R}^m$ is a given continuously differentiable mapping. By limiting our discussion to this problem, we may specialize the models employed in both the ARC and trust-region algorithms to those that are suited to solving nonlinear least-squares problems, specifically using regularized Gauss–Newton-based models, and consequently to take advantage of the ideas and implementations details proposed in [7] for

the solution of large regularized linear least-squares problems. Since we are primarily interested in large problems for which matrix factorization often has prohibitive computational cost, we shall focus on iterative algorithms for the subproblems, particularly on those implemented as part of version 2.4 of the GALAHAD optimization library [17]. Such procedures are based on the minimization of the local model of the objective function over a sequence of (nested) subspaces associated with the Lanczos procedure. As a result, they are especially suited to the large-scale setting and allow us to test the methods on large problems from the CUTEr test collection [16]. In particular, the new updating rules for the regularization parameter of the ARC algorithm are experimentally validated and a comparison with the trust-region algorithm is performed on problem (1.2).

The paper is organized as follows. In Sect. 2 we review the standard trust-region algorithm and the ARC algorithm for the solution of problem (1.1). New updating rules for the regularization parameter in the ARC algorithm are introduced in Sect. 3. Section 4 is dedicated to numerical experiments and, finally, in Sect. 5 we draw some conclusions.

Throughout the paper we use the following notation. The Euclidean (ℓ_2) norm is denoted by $\|\cdot\|$, and I represents the identity matrix. Given a sequence of vectors $\{x_k\}$, for any generic function h we let $h_k = h(x_k)$. Let $g(x) = \nabla f(x)$ where f is the objective function in (1.1) and let $J(x)$ denote the Jacobian matrix of the residual function $h(x)$ in (1.2). Finally, $\epsilon_m \approx 10^{-16}$ denotes the relative machine (double) precision.

2 The algorithms

In this section, we describe the kth iteration of two globally convergent algorithms for the solution of problem (1.1): the standard trust-region algorithm (e.g. [8]) and the ARC algorithm ([3]).

In the trust-region framework, a quadratic model of $f(x)$ around x_k is constructed by defining the model of the objective function to be

$$q_k(s) = f_k + g_k^T s + \frac{1}{2} s^T H_k s, \tag{2.3}$$

where H_k is a symmetric approximation to the local Hessian $\nabla_{xx} f_k$. Then, a trial step s_k is computed by solving (possibly only approximately) the subproblem

$$\min_{s \in \mathbb{R}^n} \{q_k(s) : \ \|s\| \leq \Delta_k\}, \tag{2.4}$$

where $\Delta_k > 0$ is the so-called trust-region radius.

By contrast, assuming that the objective's Hessian $\nabla_{xx} f$ is globally Lipschitz continuous on $\mathbb{R}^n$ with Lipschitz constant L, the cubic model used in the ARC algorithm is based on the bound

$$\begin{aligned} f(x_k + s) &= f_k + s^T g_k + \frac{1}{2} s^T \nabla_{xx} f_k s \\ &\quad + \int_0^1 (1 - \tau) s^T [\nabla_{xx} f(x_k + \tau s) - \nabla_{xx} f_k] s \, d\tau \\ &\leq f_k + s^T g_k + \frac{1}{2} s^T \nabla_{xx} f_k s + \frac{1}{6} L \|s\|^3 \stackrel{\text{def}}{=} l_k(s), \end{aligned} \tag{2.5}$$

which holds for all $s \in \mathbb{R}^n$. Thus, so long as $l_k(s_k) < l_k(0) = f_k$, the new iterate $x_{k+1} = x_k + s_k$ improves $f(x)$. In [3], a dynamic positive parameter σ_k replaces the Lipschitz constant $L/2$ and a symmetric approximation H_k to the local Hessian $\nabla_{xx} f_k$ is allowed. At each iteration, the cubic model

$$c_k(s) = f_k + s^T g_k + \frac{1}{2} s^T H_k s + \frac{1}{3}\sigma_k \|s\|^3, \tag{2.6}$$

is employed as an approximation to the objective f and the subproblem

$$\min_{s\in\mathbb{R}^n} c_k(s), \tag{2.7}$$

is solved. The parameter σ_k plays a crucial role in the description of the ARC algorithm as it measures the discrepancy between the objective function and its second order Taylor expansion and of the difference between the exact and the approximate Hessian [3].

It is important to note that the restriction on stepsize is explicitly imposed by the trust-region constraint in the trust-region case, while stepsize control is defined implicitly, indeed nonlinearly, in the cubic case. In fact, a step s_k derived by reducing (2.7) is always bounded [3, Lemma 2.2] by

$$\|s_k\| \leq 3 \max\left[\frac{\|H_k\|}{\sigma_k}, \sqrt{\frac{\|g_k\|}{\sigma_k}}\right].$$

Such a bound suggests that the regularization parameter σ_k for the ARC algorithm may loosely be interpreted as the reciprocal of the trust-region radius Δ_k. This observation in turn suggests choosing updating rule for the parameter σ_k by analogy with the trust-region case. In a standard trust-region scheme, the trust-region radius may be enlarged if there is a sufficient decrease in $f(x)$, computed by some measure of the relative objective changes, and it is reduced otherwise. In the regularization case, the parameter σ_k is decreased if there is a sufficient agreement between the objective function and the model, but increased or left unchanged otherwise.

In both algorithms, the agreement between the model and the objective function is given by the standard ratio of the achieved to the predicted reduction, and the size of this ratio is used to decide whether or not to accept the trial step and to change the regularization parameter. This ratio takes the form

$$\rho_q(s_k) = \frac{f_k - f(x_k + s_k)}{q_k(0) - q_k(s_k)}, \tag{2.8}$$

in the trust-region case, and

$$\rho_c(s_k) = \frac{f_k - f(x_k + s_k)}{c_k(0) - c_k(s_k)}, \tag{2.9}$$

in the cubic regularization case, where the models q_k and c_k are defined in (2.3) and (2.6) respectively. Without ambiguity, let $\rho(s)$ represent both $\rho_c(s)$ and $\rho_q(s)$, and let η_1, η_2 be constants such that $0 < \eta_1 < \eta_2 < 1$. We say that the iteration k is *very successful* if $\rho(s_k) \geq \eta_2$, *successful* if $\rho(s_k) \in [\eta_1, \eta_2)$, *unsuccessful* otherwise. When it is useful to distinguish the case $\rho(s_k) < 0$ within the unsuccessful case, we refer to a *very unsuccessful* iteration.

The general framework of the methods described so far is presented in Algorithm 2.1. The string `METHOD` denotes the name of the method, i.e. it is either `'TRUST-REGION'` or `'ARC'`. Sections 2.1 and 3 give further insight into Steps 1 and 4.

Algorithm 2.1 (Generic trust-region/cubic regularization method)
An initial point x_0 as well as constants $0 < \eta_1 < \eta_2 < 1$ and $\gamma > 1$ are given. If `METHOD = 'TRUST-REGION'`, set the initial radius $\Delta_0 > 0$ and the constants τ_1, τ_2 such that $0 \leq \tau_1 \leq \tau_2 \leq 1$. Else set the intial regularization parameter $\sigma_0 > 0$ and the constants ν_1, ν_2 such that $1 < \nu_1 \leq \nu_2$.

For $k = 0, 1, \ldots$, until convergence,

Step 1: Trial step computation. If `METHOD = 'TRUST-REGION'`, compute s_k as an (approximate) solution of problem (2.4). Else, compute s_k as an (approximate) solution of problem (2.7).

Step 2: Step acceptance. If `METHOD = 'TRUST-REGION'`, compute $\rho(s_k) = \rho_q(s_k)$ as in (2.8). Else, compute $\rho(s_k) = \rho_c(s_k)$ as in (2.9). If $\rho(s_k) \geq \eta_1$, let $x_{k+1} = x_k + s_k$; otherwise let $x_{k+1} = x_k$.

Step 4: Regularization parameter update. If `METHOD = 'TRUST-REGION'` set

$$\Delta_{k+1} \in \begin{cases} [\Delta_k, \infty) & \text{if } \rho(s_k) \geq \eta_2 \quad \text{[very successful iteration]}, \\ [\tau_2 \Delta_k, \Delta_k] & \text{if } \rho(s_k) \in [\eta_1, \eta_2) \quad \text{[successful iteration]}, \\ [\tau_1 \Delta_k, \tau_2 \Delta_k] & \text{otherwise} \quad \text{[unsuccessful iteration]}. \end{cases} \tag{2.10}$$

Else set

$$\sigma_{k+1} \in \begin{cases} (0, \sigma_k] & \text{if } \rho(s_k) \geq \eta_2 \quad \text{[very successful iteration]}, \\ [\sigma_k, \nu_1 \sigma_k] & \text{if } \rho(s_k) \in [\eta_1, \eta_2) \quad \text{[successful iteration]}, \\ [\nu_1 \sigma_k, \nu_2 \sigma_k] & \text{otherwise} \quad \text{[unsuccessful iteration]}. \end{cases} \tag{2.11}$$

2.1 Computing a trial step

Step 1 of Algorithm 2.1 leaves substantial implementation freedom, which may be used according to context. The focus of this paper is on the case where matrix factorizations of the Hessian matrix are not feasible, implying that iterative methods for computing a trial step are needed. We consider the class of subspace minimization methods, i.e. methods that find an approximate solution by solving a sequence of minimization problems with the additional constraint that s is contained in a subspace. This class may be divided into two subclasses depending on the construction of the sequence of subspaces. The first consists of expanding subspaces methods. The Conjugate Gradient (CG) method belongs to this subclass as it may be viewed as a subspace minimization method for finding an unconstrained minimizer of a strictly convex quadratic function, where, at each successive iteration, the quadratic function is minimized by restricting the variable to a sequence of nested Krylov subspaces. In [3, 15], methods based on this approach have been proposed for solving the regularized cubic problem (2.7) and the trust-region problem (2.4), respectively. The

second subclass comprises low-dimensional subspace methods, i.e. methods that always generate subspaces of low-dimension. Such methods have been proposed in literature only for solving problem (2.4) and differ in the choice of the subspaces [11, 12, 19, 20]. In order to apply the same subspace approach to both the trust-region and the cubic case, we consider the former subclass of methods to perform Step 1.

Consider the nonlinear least-squares problem (1.2). At the current iterate x_k, the exact Hessian of the objective function f has the form

$$\nabla_{xx} f_k = J_k^T J_k + S_k,$$

where S_k contains the second-order information on the residual. If S_k is small, it is reasonable to consider the first order approximation $H_k = J_k^T J_k$. This is the case, for instance, in a neighborhood of a zero residual solution of problem (1.2), [10]. Using the approximation $H_k = J_k^T J_k$, the quadratic model in (2.3) takes the form

$$q_k(s) = \frac{1}{2}\|J_k s + h_k\|^2, \tag{2.12}$$

which is the Gauss–Newton model for f, and the cubic model in (2.6) becomes

$$c_k(s) = \frac{1}{2}\|J_k s + h_k\|^2 + \frac{\sigma_k}{3}\|s\|^3, \tag{2.13}$$

yielding a Gauss–Newton model regularized by a cubic term.

Procedures have been proposed in [7] to solve the subproblems (2.4) and (2.7) in the special case where the models are given in (2.12) and (2.13) respectively. The core component of these procedures is the Golub and Kahan bi-diagonalization process [13] that generates orthonormal basis of a sequence of expanding subspaces $\{\mathcal{V}_j\}_{j\geq 1}$. Let $V_j \in \mathbb{R}^{n\times j}$ be the orthonormal matrix whose columns span $\mathcal{V}_j$. The solutions of problems (2.4) and (2.7) are found by computing the sequence of minimizers y_j of the reduced problems

$$\min_{y\in\mathbb{R}^j} \{q_k(V_j y) : \|y\| \leq \Delta_k\}, \tag{2.14}$$

and

$$\min_{y\in\mathbb{R}^j} c_k(V_j y), \tag{2.15}$$

respectively, increasing the dimension j of the subspaces until $s_j = V_j y_j$ is sufficiently accurate. At that point, the step s_k in the full space is taken as the last computed s_j [7].

It is interesting to note, that if the LSQR algorithm [22] is used to solve the unconstrained problem $\min_s q_k(s)$, a basis of the Krylov subspaces

$$\mathcal{K}_j = \left\{(J_k^T J_k)^i J_k^T h_k\right\}_{i=0}^{j-1},$$

is given by the columns of V_j. Due to the equivalence between the LSQR and CG methods, the sequence s_j generated by LSQR has the favorable property to be monotonically increasing in norm [24]. Thus, either LSQR finds a solution in the interior of the trust-region, or finds an iterate s_j s.t. $\|s_{j-1}\| \leq \Delta_k < \|s_j\|$ and in this case we may conclude that the solution of the problem (2.4) lies on the boundary of the

trust-region. When this happens two alternative strategies can be followed: either the so-called Steihaug–Toint point [8, §7.5.1] is computed or a solution on the boundary is computed to any prescribed accuracy. The Steihaug–Toint strategy interpolates the last interior iterate s_{j-1} with the newly discovered exterior one s_j to find the boundary point between them. The resulting step has the favorable property that the optimal decrease of q_k at the exact solution of the trust-region problem (2.4), is no more than twice that achieved at the Steihaug–Toint point (see [26] or [8, Theorem 7.5.9]). On the negative side however, it makes no attempt to find a constrained solution with prescribed accuracy. A more refined strategy solves a sequence of constrained reduced problems (2.14) with increasing j until s_j is sufficiently accurate [7]. Note that this strategy specializes to problem (2.14) with q_k given in (2.12) the GLTR method [15] for the general trust-region problem (2.4) in which the CG method is used as long as the iterates are in the interior of the trust-region and the expanding subspaces are defined by the Lanczos vectors.

It is important to remark, that once a sufficiently accurate solution $y_{\bar{j}}$ of the reduced problems (2.14) and (2.15) has been found, the solution $s_{\bar{j}} = V_{\bar{j}} y_{\bar{j}}$ in the full space must be recovered. If the basis of $\mathcal{V}_j$ is needed, it has to be regenerated afresh, and a second-phase is activated. In the trust-region case, the LSQR procedure allows one to recur s_j from s_{j-1} provided the iterates lie in the interior of the trust-region or at the Steihaug–Toint point without the need for the second phase. However, the second-phase is needed if we look for a more accurate solution on the trust-region boundary or if the cubic regularization algorithm is employed.

The second-phase may be accelerated if needed by storing the first t (say) basis vectors of the subspace $\mathcal{V}_{\bar{j}}$ as calculated in the first-phase so that the bidiagonalization process may be restarted at iteration $j = t$. Clearly, the efficiency of these implementations strongly depends on the magnitude of the chosen t.

3 Updating rules for the regularization parameters

Because of its central role, the definition of a procedure to update the regularization parameters at Step 4 of Algorithm 2.1 may have a crucial influence on its overall performance. In this section, we first review two established updating strategies for the trust-region radius Δ_k and then propose new strategies for the parameter σ_k for the ARC algorithm.

Clearly, the rule (2.10) in Algorithm 2.1 leaves considerable flexibility. A simple and reasonable choice is to select

$$\Delta_{k+1} = \begin{cases} \max\{\gamma_2 \|s_k\|, \Delta_k\} & \text{if } \rho_q(s_k) \geq \eta_2 & \text{[very successful iteration],} \\ \Delta_k & \text{if } \rho_q(s_k) \in [\eta_1, \eta_2) & \text{[successful iteration], and} \\ \gamma_1 \|s_k\| & \text{otherwise} & \text{[unsuccessful iteration],} \end{cases} \tag{3.16}$$

where γ_1 and γ_2 are constants such that $0 < \gamma_1 < 1 \leq \gamma_2$, but further refinements are possible using interpolation techniques in the unsuccessful case. If $\rho_q(s_k)$ is negative, the agreement between the model and the objective function is extremely poor and some drastic action might be warranted. In this case, we presume for simplicity that

s_{k+1} will be aligned with s_k and we compute a trust-region radius small enough to ensure that the new step gives at least a successful iteration [8, Chap. 17]. To compute such a radius, we consider a step of the form αs_k with $\alpha > 0$ and we set $\Delta_{k+1} = \alpha_\eta^{bad}\Delta_k$ where α_η^{bad} solves $\rho_q(\alpha s_k) = \eta$, which is equivalent to the scalar nonlinear equation

$$f_k - f(x_k + \alpha s_k) = \eta(q_k(0) - q_k(\alpha s_k)), \tag{3.17}$$

with $\eta \in [\eta_1, 1)$ and η_1 as given in Algorithm 2.1. To avoid the expense of computing the extra function value $f(x_k + \alpha s_k)$ and to simplify the solution of (3.17), the scalar function $\hat{f}(\alpha) = f(x_k + \alpha s_k)$, $\alpha > 0$ is replaced by a quadratic interpolating polynomial for $\hat{f}$. The polynomial $t_f(\alpha)$ such that t_f and t_f' agree with $\hat{f}$ and $\hat{f}'$ at 0, and $t_f(1) = \hat{f}(1) = f(x_k + s_k)$, is given by

$$t_f(\alpha) = f_k + g_k^T s_k \alpha + \big(f(x_k + s_k) - f_k - g_k^T s_k\big)\alpha^2.$$

Substituting this value for $f(x_k + \alpha s_k)$ into (3.17) and solving for α, yields the value of α_η^{bad} given by

$$\alpha_\eta^{bad} = \frac{(1-\eta)g_k^T s_k}{(1-\eta)(f_k + g_k^T s_k) + \eta\, q_k(s_k) - f(x_k + s_k)}. \tag{3.18}$$

We may therefore modify (3.16) to use this information and obtain the more sophisticated rule

$$\Delta_{k+1} = \begin{cases} \max\{\gamma_2\|s_k\|, \Delta_k\} & \text{if } \rho_q(s_k) \geq \eta_2 \\ & \text{[very successful iteration],} \\ \Delta_k & \text{if } \rho_q(s_k) \in [\eta_1, \eta_2) \\ & \text{[successful iteration],} \\ \gamma_1\|s_k\| & \text{if } \rho_q(s_k) \in [0, \eta_1) \\ & \text{[unsuccessful iteration], and} \\ \min\{\gamma_1\|s_k\|, \max\{\gamma_3, \alpha_\eta^{bad}\}\Delta_k\} & \text{otherwise} \\ & \text{[very unsuccessful iteration],} \end{cases} \tag{3.19}$$

where α_η^{bad} is given by (3.18) and the constants $\gamma_1, \gamma_2, \gamma_3$ are such that $0 < \gamma_3 < \gamma_1 < 1 \leq \gamma_2$ [8].

Let us now consider the ARC framework with this in mind. The updating rule proposed in [3] aims to try to reduce the model rapidly to match the Newton model once convergence sets in, while maintaining some regularization before the asymptotic behaviour. The rule used in the reported experiments was

$$\sigma_{k+1} = \begin{cases} \max\{\min\{\sigma_k, \|g_k\|\}, \epsilon_m\} & \text{if } \rho_c(s_k) \geq \eta_2 \\ & \text{[very successful iteration],} \\ \sigma_k & \text{if } \rho_c(s_k) \in [\eta_1, \eta_2) \\ & \text{[successful iteration], and} \\ \gamma\sigma_k & \text{otherwise} \\ & \text{[unsuccessful iteration],} \end{cases} \tag{3.20}$$

with $\gamma \geq 1$. Clearly, the relationship between the step length and the regularization parameter in (3.20) is not as simple as in the updating rules (3.16) for the trust-region case and the control of the first by the second is performed implicitly.

To relate the step size and the parameter σ_k in a more direct way, we now present an alternative strategy for updating σ_k in the spirit of the interpolation procedures used with the trust-region scheme. Specifically, we try to ensure, in the very unsuccessful case, that the next iterate gives at least a successful iteration. In the very successful case we may also exploit the overestimation property (2.5) measuring at each iteration the gap between the current objective function value $f(x_k + s_k)$ and the current model value $c_k(s_k)$ and reduce σ_k in order to decrease this gap (cf. [18, 25]). In particular, given the current x_k, σ_k and s_k, we presume, as above, that s_{k+1} is of the form $\alpha s_k, \alpha > 0$ and compute the value σ_{k+1} to ensure suitable conditions on αs_k.

As in the trust-region case, we avoid the need to compute the value of $f(x_k + \alpha s_k)$ by using instead a suitable interpolating approximation. The interpolating cubic function $p_f(\alpha),\ \alpha \geq 0$ we use here is built by requiring that $p_f(0) = f_k,\ p'_f(0) = g_k^T s_k,\ p''(0) = s_k^T H_k s_k$ and $p_f(1) = f(x_k + s_k)$, and hence takes the form

$$p_f(\alpha) = f_k + g_k^T s_k \alpha + \frac{1}{2} s_k^T H_k s_k \alpha^2 + p_{f3} \alpha^3, \tag{3.21}$$

where

$$p_{f3} = f(x_k + s_k) - q_k(s_k). \tag{3.22}$$

The quadratic model (2.3) along the direction s_k may be written as

$$q(\alpha) = f_k + g_k^T s_k \alpha + \frac{1}{2} s_k^T H_k s_k \alpha^2, \tag{3.23}$$

while its regularized cubic counterpart (2.6) is

$$c(\alpha, \sigma) = q(\alpha) + \frac{\sigma \|s_k\|^3}{3} \alpha^3. \tag{3.24}$$

We now define the current overestimation gap χ_k^f to be

$$\chi_k^f = c_k(s_k) - f(x_k + s_k). \tag{3.25}$$

Note that the model c_k at s_k overestimates $f(x_k + s_k)$, i.e. $\chi_k^f \geq 0$, if and only if $\rho_c(s_k) \geq 1$.

Consider the very successful ($\chi_k^f > 0$) case first, in which case the regularization parameter should be decreased. If the current gap χ_k^f is large enough, we aim at reducing it by a factor $\beta \in (0, 1)$. Assume first that $f(x_k + s_k) \geq q_k(s_k)$. Remembering that the next step should minimize the cubic model (in particular along s_k), we thus search for α and σ such that

$$c(\alpha, \sigma) - p_f(\alpha) = \beta \chi_k^f \quad \text{and} \tag{3.26}$$

$$\frac{d}{d\alpha} c(\alpha, \sigma) = 0, \tag{3.27}$$

$c(\alpha, \sigma)$ and $p_f(\alpha)$ given in (3.24) and (3.21). It follows from (3.26) that

$$\sigma = 3 \frac{\beta \chi_k^f + p_{f3} \alpha^3}{\alpha^3 \|s_k\|^3} \equiv \sigma_k + 3 \frac{\chi_k^f}{\|s_k\|^3} \left(\frac{\beta - \alpha^3}{\alpha^3} \right), \tag{3.28}$$

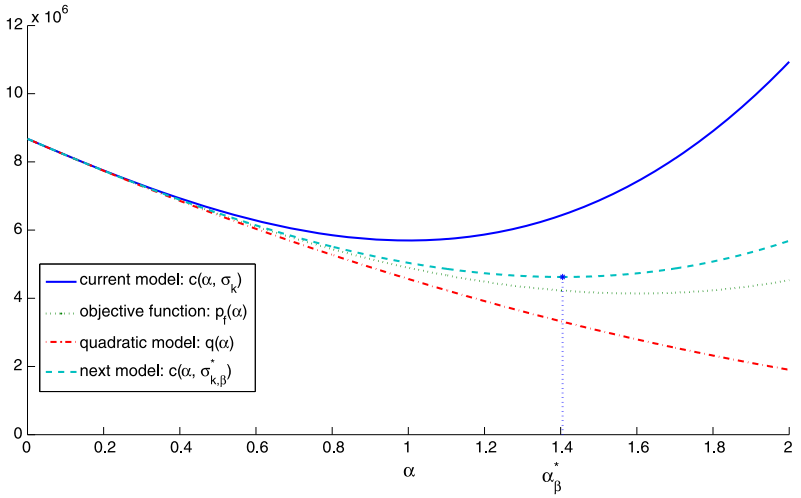

Fig. 1 Very successful iteration and $f(x_k + s_k) \geq q_k(s_k)$

and substituting (3.28) into (3.27), we find that the required α satisfies the cubic scalar equation

$$3\beta\chi_k^f + g_k^T s_k \alpha + s_k^T H_k s_k \alpha^2 + 3p_{f3}\alpha^3 = 0. \tag{3.29}$$

Thus, we determine the root α of (3.29) which exceeds $\sqrt[3]{\beta}$ by the least (if there is such a root) and recover $\sigma_{k,\beta}^*$ from (3.28). If there is no such α, or if α is too large, we simply reduce σ_k by a factor $\delta_1 \in (0, 1)$.

In Figs. 1, 2 and 3 the current cubic model ($c(\alpha, \sigma_k)$), the approximated objective function ($p_f(\alpha)$), the quadratic model ($q(\alpha)$) and the next cubic model (denoted by $c(\alpha, \sigma_{k,\beta}^*)$ in the very successful case and by $c(\alpha, \sigma_{k,\eta}^*)$ in the very unsuccessful case) are plotted. Figure 1 represents an example where the k-th iterate is very successful and $f(x_k + s_k) \geq q_k(s_k)$. In this example, $\beta = 0.5$ and (3.29) has two positive roots. The largest one (α_β^* in the figure) is larger than $\sqrt[3]{\beta} \approx 0.7937$ and gives $\sigma_{k,\beta}^*$ such that $\sigma_{k,\beta}^* < \sigma_k$.

Consider now the case where $f(x_k + s_k) < q_k(s_k)$. If we attempt to solve the system (3.26)–(3.27), i.e. try to reduce the quantity $c_k(s_k) - f(x_k + s_k)$ by a factor β, we might reduce this gap too much, leading to undesirable value of the new σ. Figure 2(a) illustrates the typical situation: in this example ($\beta = 0.5$), (3.29) only has one positive solution ($\alpha^{\chi_k^f} \approx 0.745$ in the figure), but it is smaller than $\sqrt[3]{\beta}$ so that the corresponding $\sigma_{k,\beta}^*$ computed by (3.28) is larger than the current σ_k. To avoid this undesirable situation, we instead attempt to reduce the following gap

$$\chi_k^q = c_k(s_k) - q_k(s_k), \tag{3.30}$$

and search for α and σ such that

$$c(\alpha, \sigma) - q(\alpha) = \beta\,\chi_k^q \quad \text{and} \tag{3.31}$$

$$\frac{d}{d\alpha} c(\alpha, \sigma) = 0, \tag{3.32}$$

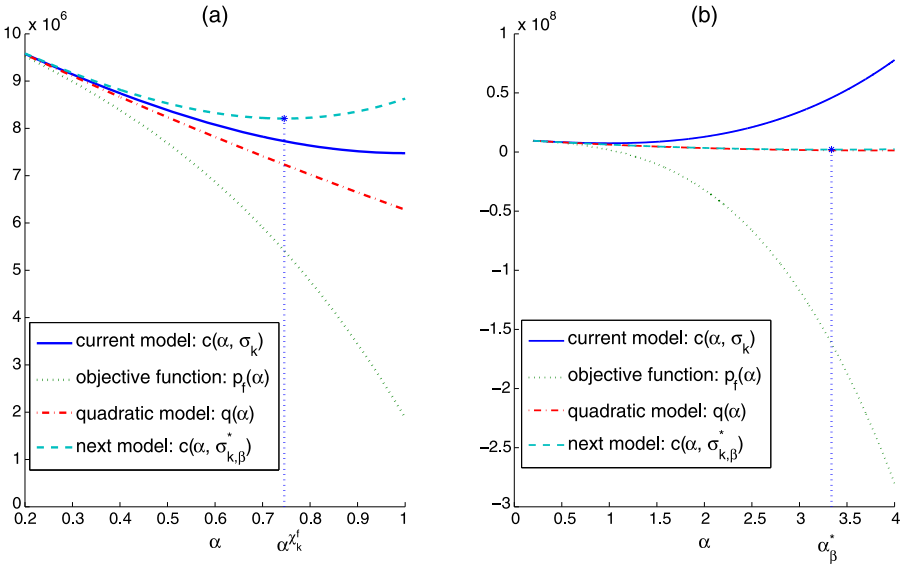

Fig. 2 Very successful iteration and $f(x_k + s_k) < q_k(s_k)$

with $c(\alpha, \sigma)$ and $q(\alpha)$ given in (3.24). Computing σ from (3.31), we then find that

$$\sigma = 3\frac{\beta \chi_k^q}{\alpha^3 \|s_k\|^3} \equiv \frac{\beta}{\alpha^3}\sigma_k, \tag{3.33}$$

and substituting (3.33) in (3.32) yields that α solves the quadratic scalar equation

$$3\beta\chi_k^q + g_k^T s_k \alpha + s_k^T H_k s_k \alpha^2 = 0. \tag{3.34}$$

As in the previous case, we compute the root of (3.34) which exceeds $\sqrt[3]{\beta}$ by the least (if such a root exists) and compute the corresponding value $\sigma_{k,\beta}^*$ using (3.33). Once again, if there is no such α, or if α is too large, we simply reduce σ_k by a factor $\delta_1 \in (0, 1)$. Figure 2(b) illustrates the same example as in Fig. 2(a) but now solving the system (3.31)–(3.32): (3.34) has 2 positive roots and one (α_β^* in the figure) is larger than $\sqrt[3]{\beta}$, so that the corresponding $\sigma_{k,\beta}^*$ is smaller than σ_k.

Let us now turn to the very unsuccessful case, $\rho_c(s_k) < 0$, where we wish to increase the regularization parameter. We proceed as in the trust-region framework simply requiring that αs_k produces at least a successful iterate. We thus search for α and σ such that

$$f_k - p_f(\alpha) = \eta(f_k - c(\alpha, \sigma)), \quad \text{and} \tag{3.35}$$

$$\frac{d}{d\alpha} c(\alpha, \sigma) = 0, \tag{3.36}$$

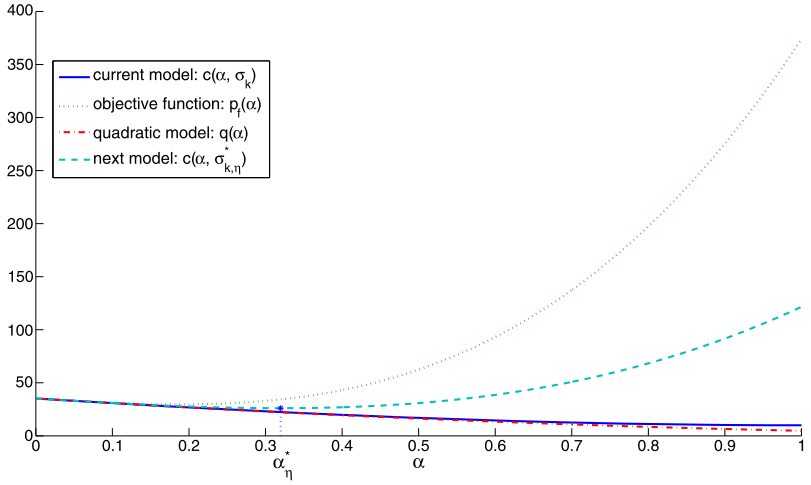

Fig. 3 Very unsuccessful iteration

for some $\eta \in [\eta_1, 1)$. Computing σ from (3.36) we obtain that

$$\sigma = \frac{-g_k^T s_k - s_k^T H_k s_k \alpha}{\alpha^2 \|s_k\|^3}, \tag{3.37}$$

and substituting this expression in (3.35), we find that α must be a root of the quadratic scalar equation

$$2(3-2\eta) g_k^T s_k + (3-\eta) s_k^T H_k s_k \alpha + 6 p_{f3} \alpha^2 = 0, \tag{3.38}$$

where p_{f3} is positive since $\rho_c(s_k) < 0$. The discriminant of the above equation is given by

$$(3-\eta)^2 (s_k^T H_k s_k)^2 - 48(3-2\eta) g_k^T s_k p_{f3},$$

and as $\eta < 3/2$, it is always positive. In this case, the above equation has two roots of opposite sign. If α_η^* is the positive one, we then compute $\sigma_{k,\eta}^*$ from (3.37) with $\alpha = \alpha_\eta^*$. Figure 3 shows an example of this case.

Combining these different cases together, we are now able to state the complete rule for updating the current regularization parameter σ_k: it is described as Algorithm 3.1 on the following page. This algorithm also safeguards against the case where equations (3.29) and (3.34) do not admit a solution larger than $\sqrt[3]{\beta}$, or where such a solution exists but may be very much larger than this value, resulting in a tiny corresponding $\sigma_{k,\beta}^*$. In all these cases, we simply choose a fraction of the current σ_k. On the other hand, note that, by definition, the values of $\sigma_{k,\beta}^*$ computed in (3.39) and (3.40) are positive and smaller than the current σ_k. Figure 4 shows a typical variation of σ_{k+1} computed by Algorithm 3.1 as a function of the objective function value $f(x_k + s_k)$. This curve for σ_{k+1} is a piecewise linear function where the sloping pieces correspond to values of σ_{k+1} computed by the interpolation rules (3.39)–(3.41).

Algorithm 3.1 (Regularization parameter update)
Given the current x_k, s_k, σ_k, let the constants η_1 and η_2 be fixed by Algorithm 2.1.

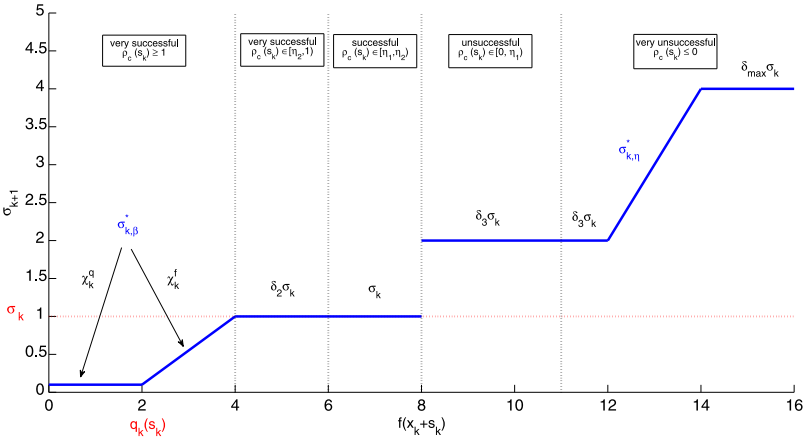

Fig. 4 Typical variation of σ_{k+1} computed by Algorithm 3.1 as a function of the objective function value $f(x_k + s_k)$

Let the positive threshold ϵ_χ and the constants $\delta_1, \delta_2, \delta_3, \delta_{max}, \beta, \eta$ be chosen such that

$$0 < \delta_1 < \delta_2 \leq 1 < \delta_3 \ll \delta_{max}, \qquad 0 < \beta < 1, \qquad 0 < \eta < 3/2, \qquad 1 < \alpha_{max}.$$

Compute $\rho_c(s_k)$ by (2.9) and

$$\chi_k = c_k(s_k) - \max\{f(x_k + s_k), q_k(s_k)\}.$$

- If $\rho_c(s_k) \geq 1$ and $\chi_k \geq \epsilon_\chi$, then
  - If $f(x_k + s_k) \geq q_k(s_k)$, solve equation (3.29) with $\chi^f_k = \chi_k$.
    Let $\mathcal{A}^* = \{\alpha \mid \alpha \text{ is a root of (3.29) and } \alpha \geq \sqrt[3]{\beta}\}$.
    * If $\mathcal{A}^* = \emptyset$, set $\sigma_{k+1} = \max\{\delta_1\sigma_k, \epsilon_m\}$.
    * If $\mathcal{A}^* \neq \emptyset$, let $\alpha^*_\beta = \text{argmin}\{(\alpha - \sqrt[3]{\beta}) \mid \alpha \in \mathcal{A}^*\}$.
      If $\alpha^*_\beta \leq \alpha_{max}$, compute

$$\sigma^*_{k,\beta} = \sigma_k + 3\frac{\chi_k}{\|s_k\|^3}\left(\frac{\beta - {\alpha^*_\beta}^3}{{\alpha^*_\beta}^3}\right), \tag{3.39}$$

      and set $\sigma_{k+1} = \max\{\sigma^*_{k,\beta}, \epsilon_m\}$;
      If $\alpha^*_\beta > \alpha_{max}$, set $\sigma_{k+1} = \max\{\delta_1\sigma_k, \epsilon_m\}$.
  - Else if $f(x_k + s_k) < q_k(s_k)$, solve equation (3.34) with $\chi^q_k = \chi_k$.
    Let $\mathcal{A}^* = \{\alpha \mid \alpha \text{ is a root of (3.34) and } \alpha \geq \sqrt[3]{\beta}\}$.
    * If $\mathcal{A}^* = \emptyset$, set $\sigma_{k+1} = \max\{\delta_1\sigma_k, \epsilon_m\}$.
    * If $\mathcal{A}^* \neq \emptyset$, let $\alpha^*_\beta = \text{argmin}\{(\alpha - \sqrt[3]{\beta}) \mid \alpha \in \mathcal{A}^*\}$.
      If $\alpha^*_\beta \leq \alpha_{max}$, compute

$$\sigma^*_{k,\beta} = \frac{\beta}{{\alpha^*_\beta}^3}\sigma_k, \tag{3.40}$$

      and set $\sigma_{k+1} = \max\{\sigma^*_{k,\beta}, \epsilon_m\}$;
      If $\alpha^*_\beta > \alpha_{max}$, set $\sigma_{k+1} = \max\{\delta_1\sigma_k, \epsilon_m\}$.

- Else if $\rho_c(s_k) \geq 1$ and $\chi_k < \epsilon_\chi$, set $\sigma_{k+1} = \max\{\delta_2\sigma_k, \epsilon_m\}$.
- Else if $\rho_c(s_k) \in [\eta_2, 1)$, set $\sigma_{k+1} = \max\{\delta_2\sigma_k, \epsilon_m\}$.
- Else if $\rho_c(s_k) \in [\eta_1, \eta_2)$, set $\sigma_{k+1} = \sigma_k$.
- Else if $\rho_c(s_k) \in [0, \eta_1)$, set $\sigma_{k+1} = \delta_3\sigma_k$.
- Else ($\rho_c(s_k) < 0$), compute the positive root α_η^* of (3.38) and compute

$$\sigma_{k,\eta}^* = \frac{-g_k^T s_k - s_k^T H_k s_k \alpha_\eta^*}{\alpha_\eta^{*2} \|s_k\|^3}. \tag{3.41}$$

Set $\sigma_{k+1} = \min\{\max\{\sigma_{k,\eta}^*, \delta_3\sigma_k\}, \delta_{max}\sigma_k\}$.

4 Numerical experiments

We now present numerical experiments on nonlinear least-squares problems (1.2), where we study the numerical behaviour of the trust-region and the ARC algorithms employing the different updating rules presented in Sect. 3 in a first stage, and, in a second stage, compare the two algorithms using the best performing rules.

To compare the overall computational effort of the algorithms we use the performance profiles proposed by Dolan and Moré [9] for a given set of test problems and a given selection of algorithms. For each problem P in our testing set and each Algorithm A, we let $\texttt{fe}_{P,A}$ denote the number of function evaluations required to solve problem P using Algorithm A and $\texttt{fe}_P$ be number of function evaluations required by the best algorithm to solve problem P, i.e. the algorithm which uses the fewest function evaluations. The function evaluation performance profile is defined for the Algorithm A as

$$\pi_A(\tau) = \frac{\text{number of problems s.t. } \texttt{fe}_{P,A} \leq \tau\, \texttt{fe}_P}{\text{number of problems}}, \quad \tau \geq 1. \tag{4.42}$$

Analogously, we define the CPU time performance profile $\phi_A(\tau)$, $\tau \geq 1$ measuring the efficiency of the algorithm A in terms of the employed CPU time.

In what follows and in order to improve readability of the performance profile graphs, we limit the plots $\pi_A(\tau)$ and $\phi_A(\tau)$ to the interval $[1, 4]$ and report the number of failures in the legend.

4.1 The problem set

Numerical results are given for problems from the CUTEr test collection [16]. The test examples we consider are constructed using the CUTEr interactive select tool in order to locate the problems with no objective function and with constraints that are systems of nonlinear equations. We exclude the problems CHEMRCTA, CHEMRCTB, DRCAVTY3, FLOSP2HH, FLOSP2HL, FLOSP2HM, FLOSP2TH, FLOSP2TL, FLOSP2TM, HYDCAR20, SEMICON2 and SEMICN2U as no algorithm succeeded in solving these problems for any tested parameter choice. For some CUTEr problems, we considered variants that differ in the dimensions (denoted with the superscript ^{2},^{3}). The resulting testing set consists of 95 problems of the form (1.2) whose names and dimensions are reported in Table 1. The problems ARGLALE, ARGBLE,

Table 1 The problem set

Name	n	m	Name	n	m	Name	n	m
AIRCRFTA	8	5	DECONVNE	61	41	OSCIPANE	500	500
ARGAUSS	3	15	DRCAVTY1	196	100	PFIT1	2	2
ARGLALE	200	400	DRCAVTY2	4489	3969	PFIT2	2	2
ARGLBLE	200	400	EIGENA	110	110	PFIT3	2	2
ARGTRIG	200	200	EIGENA^2	2550	2550	PFIT4	2	2
ARTIF	502	500	EIGENA^3	4970	4970	POROUS1	1024	900
ARTIF^2	5002	5000	EIGENB	110	110	POROUS1^2	5184	4900
ARWDHNE	500	998	EIGENB^2	2550	2550	POROUS1^3	22500	21904
BDVALUES	102	100	EIGENC	462	462	POROUS2	1024	900
BDVALUS^2	5002	5000	EIGENC^2	2652	2652	POROUS2^2	5184	4900
BOOTH	2	2	GOTTFR	2	2	POROUS2^3	22500	21904
BRATU2D	484	400	GROWTH	3	12	POROUS2^4	62500	61504
BRATU2D^2	5184	4900	HATFLDF	3	3	POWELLBS	2	2
BRATU2DT	484	400	HATFLDG	25	25	POWELLSQ	2	2
BRATU2DT^2	5184	4900	HEART6	6	6	QR3D	610	610
BRATU3D	1000	512	HEART8	8	8	QR3D^2	2420	2420
BRATU3D^2	4913	3375	HIMMELBA	2	2	QR3DBD	457	610
BROWNALE	200	200	HIMMELBC	2	2	QR3DBD^2	1717	2420
BROWNALE^2	1000	1000	HIMMELBD	2	2	RECIPE	3	3
BROYDN3D	1000	1000	HIMMELBE	3	3	SINVALNE	2	2
BROYDN3D^2	10000	10000	HS8	2	2	SPMSQRT	10000	16664
BROYDNBD	1000	1000	HYDCAR6	29	29	TRIGGER	7	6
BROYDNBD^2	10000	10000	HYPCIR	2	2	WOODSNE	10000	7501
CBRATU2D	3200	2888	INTEGREQ	102	100	YATP1SQ	2600	2600
CBRATU3D	3456	2000	INTEGREQ^2	502	500	YATP1SQ^2	40400	40400
CHANDHEQ	100	100	METHANB8	31	31	YATP1SQ^3	63000	63000
CHANNEL	2400	2398	METHANL8	31	31	YATP2SQ	2600	2600
CHANNEL^2	9600	9598	MSQRTA	4900	4900	YATP2SQ^2	40400	40400
CHNRSBNE	50	98	MSQRTA^2	5625	5625	YATP2SQ^3	63000	63000
CLUSTER	2	2	MSQRTB	4900	4900	YFITNE	3	17
COOLHANS	9	9	MSQRTB^2	5625	5625	ZANGWIL3	3	3
CUBENE	2	2	NYSTROM5	18	20			

`GROWTH`, `HIMMELBD` and `OSCIPANE` are large residual problems, i.e. the objective function value at the computed solution is much greater than one, the remaining are small or zero residual problems. Moreover, for 9 problems $m > n$, for 28 problems $m < n$, the remaining 58 ones being square.

4.2 Implementation issues

We implemented Algorithm 2.1 in Fortran 95, using the procedures presented in Sect. 2.1 to solve the subproblem at Step 1. We consider two implementations of

the trust-region algorithm (TR-ST and TR-bST) which use the GALAHAD's package [17] LSTR and differ in the computation of the boundary trust-region solution: TR-ST computes the Steihaug–Toint point, TR-bST computes a more accurate solution as described in Sect. 2.1. The tested version of the ARC algorithm for solving problem (1.2) has been implemented using the GALAHAD's packages LSRT and it is denoted by ARC-LS.

In Algorithm 2.1, we set the specific algorithmic constants

$$\eta_1 = 0.01, \qquad \eta_2 = 0.95, \tag{4.43}$$

and the initial regularization parameters Δ_0 and σ_0 are chosen equal to one. The algorithm is terminated as soon as either

$$\|J_k^T h_k\| \leq \max\{\epsilon_{ga},\, \epsilon_{gr}\, \|J_0^T h_0\|\} \quad \text{or} \quad \|h_k\| \leq \max\{\epsilon_{fa},\, \epsilon_{fr}\, \|h_0\|\}, \tag{4.44}$$

where $\epsilon_{fa}, \epsilon_{ga}, \epsilon_{fr}, \epsilon_{gr} > 0$ are tolerances chosen as $\epsilon_{fa} = \epsilon_{ga} = 10^{-6}$, $\epsilon_{fr} = \epsilon_{gr} = 10^{-12}$. Moreover, we require that the trial step s_k computed at Step 1 of Algorithm 2.1 satisfies the inexact stopping criterion given by

$$\|J_k^T (J_k s_k + h_k) + \lambda_k s_k\| \leq \min\{\epsilon_{in},\, \|J_k^T h_k\|^{1/2}\} \|J_k^T h_k\|, \tag{4.45}$$

where $\lambda_k = \sigma_k \|s_k\|$ in ARC-LS and in the trust-region case if $\|s_k\| = \Delta_k$ then λ_k is an estimate of the Lagrange multiplier associated with the trust-region constraint while if the $\|s_k\| < \Delta_k$ then $\lambda_k = 0$; the parameter ϵ_{in} is fixed at $\epsilon_{in} = 10^{-1}$.

If the problem dimension n is lower than 50, we allow for the generation of the full space in the Krylov sequence in order to compute a very accurate solution of the subproblems (2.14) and (2.15). Furthermore, we record the first $t = 10$ basis vectors of the current Krylov subspace to re-use them to make computational savings as mentioned in Sect. 2.1 in any potential second phase. Any run exceeding 2 hours of CPU time, performing more than 5000 outer iterations or if the magnitude of computed search direction is lower than $10\epsilon_m$, is considered a failure. All other parameters in the GALAHAD's packages are set at their default values.

All our tests were performed on an Intel Xeon (TM) 3.4 GHz, 1 GB of RAM; the codes are all double precision, and compiled under g95 without optimization (default).

4.3 Numerical results

We consider first the trust-region algorithm and the trust-region radius updating rules described in Sect. 3. The sensitivity of trust-region algorithms to their parameters have been intensively studied in literature, see e.g. [14], and then we refer to [8, Chap. 17] for optimal choice of the parameters in the updating rules (3.16) and (3.19). In particular, we use the suggested values

$$\gamma_1 = 1/2, \qquad \gamma_2 = 2, \qquad \gamma_3 = 0.0625, \qquad \eta = \eta_2, \tag{4.46}$$

and we tried the further value $\eta = \eta_1$ in (3.18) with η_1, η_2 given in (4.43).

In Fig. 5, the function evaluation performance profiles show that both TR-ST and TR-bST are slightly more efficient using the updating rule (3.19) with $\eta = \eta_1$. Moreover, TR-bST is also a little more robust with this choice. The performance profile

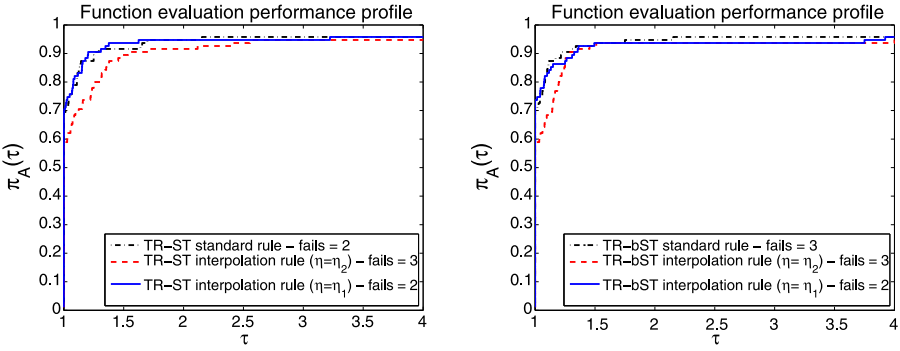

Fig. 5 The function evaluation performance profile: `TR-ST` (*left*) and `TR-bST` (*right*) with (3.16) ("standard rule") and (3.19) using $\eta = \eta_1, \eta_2$ ("interpolation rule")

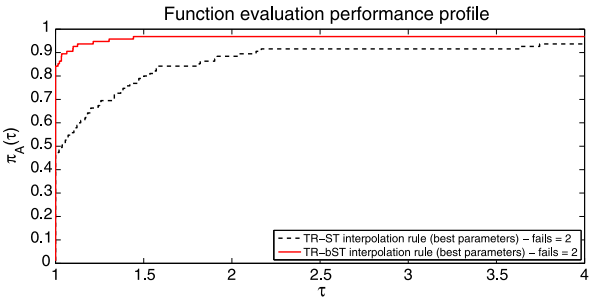

Fig. 6 The function evaluation performance profile: `TR-ST` and `TR-bST` with the interpolation rule (3.19) and the best parameter choice ($\eta = \eta_1$)

of Fig. 6 summarizes the comparison between the two trust-region implementations using the best performing rule with the best parameter choice. As one might hope, the figure suggests that the extra effort required to solve the subproblem more accurately appears to offer some overall benefit.

We now examine the sensitivity in number of function evaluations for the parameter choices of the new updating rule for σ_k for the ARC algorithm. To this purpose, we performed a small parametric study starting from the following reasonable values for the parameters in Algorithm 3.1:

$$\begin{aligned} &\beta = 1/100, \qquad \alpha_{max} = 2, \qquad \epsilon_\chi = 10^{-8}, \qquad \delta_1 = 1/10, \qquad \delta_2 = 1, \\ &\eta = \eta_1, \qquad \delta_3 = 2, \qquad \delta_{max} = 100, \end{aligned} \tag{4.47}$$

and varying one parameter at the time in some set to find the best performing value.

More precisely, let all the parameters be ordered as β, α_{max}, ϵ_χ, δ_1, δ_2, η, δ_3, δ_{max} and be fixed as in (4.47). Let p be a parameter to be analyzed. Moreover, let $I_p = \{p_1, \ldots, p_q\}$ be a set of trial values for p, A_{p_i} be the `ARC-LS` algorithm run with $p = p_i$ and let $\pi_{A_{p_i}}(\tau)$ be the performance measure defined in (4.42) comparing the algorithms A_{p_i}, $p_i \in I_p$. To estimate the efficiency of these algorithms, we compute the percentage of problems ($\%\mathrm{pb}_{\hat{\tau}}$) for which $\pi_{A_{p_i}}(\tau) \leq \hat{\pi}$ with $\hat{\pi} \gtrsim 1$ and to evaluate their robustness, we compute the number of failures. Taking into account these performance measures, we fix the "best" value for the parameter $p \in I_p$ and we proceed with the analysis of the subsequent parameter in the list. In Table 2, we

Table 2 Parametric study

p	I_p	#fails	%pb$_{\hat{\tau}}$, $\hat{\tau}=1$	%pb$_{\hat{\tau}}$, $\hat{\tau}=1.15$	%pb$_{\hat{\tau}}$, $\hat{\tau}=1.25$	%pb$_{\hat{\tau}}$, $\hat{\tau}=1.5$	%pb$_{\hat{\tau}}$, $\hat{\tau}=2$
β	0.001	3	58.95	90.53	92.63	93.68	95.79
	0.005	4	55.79	86.32	92.63	93.68	95.79
	0.01	3	68.42	91.58	95.79	95.79	96.84
	0.05	3	51.58	82.11	89.47	93.68	95.79
	0.1	4	51.58	81.05	89.47	95.79	95.79
α_{max}	1	4	42.11	69.47	75.79	91.58	94.74
	2	3	60.00	88.42	92.63	94.74	94.74
	3.5	3	55.79	85.26	90.53	92.63	92.63
	5	3	61.05	84.21	90.53	92.63	92.63
	10	3	63.16	86.32	90.53	91.58	92.63
	50	4	61.05	84.21	89.47	90.53	91.58
ϵ_χ	10^{-12}	3	81.05	92.63	93.68	95.79	95.79
	10^{-11}	3	83.16	93.68	95.79	95.79	96.84
	10^{-10}	2	80.00	95.79	96.84	96.84	97.89
	10^{-9}	3	76.84	92.63	93.68	94.74	96.84
	10^{-8}	3	73.68	92.63	92.63	94.74	94.74
	10^{-6}	4	65.26	78.95	82.11	86.32	90.53
δ_1	0.01	6	69.47	85.26	88.42	91.58	92.63
	0.05	3	65.26	92.63	94.74	95.79	95.79
	0.1	2	71.58	94.74	97.89	97.89	97.89
	0.25	3	62.11	87.37	93.68	95.79	95.79
	0.5	3	58.95	83.16	92.63	95.79	95.79
δ_2	0.25	3	61.05	74.74	85.26	91.58	94.74
	0.5	3	53.68	78.95	85.26	90.53	96.84
	0.75	4	57.89	86.32	91.58	94.74	95.79
	0.9	4	57.89	85.26	90.53	93.68	95.79
	1	2	58.95	89.47	95.79	97.89	97.89
η	**η_1**	2	72.63	94.74	95.79	96.84	97.89
	$(\eta_2-\eta_1)/2$	4	68.42	88.42	91.58	94.74	95.79
	η_2	5	62.11	81.05	88.42	91.58	94.74
	1.25	3	63.16	78.95	87.37	89.47	90.53
δ_3	1.50	4	69.47	89.47	93.68	94.74	94.74
	2	2	72.63	93.68	96.84	97.89	97.89
	2.5	4	66.32	89.47	91.58	94.74	95.79
	3	4	65.26	88.42	94.74	95.79	95.79
	4	3	66.32	83.16	92.63	94.74	96.84
δ_{max}	10	4	66.32	86.32	90.53	91.58	93.68
	50	4	70.53	89.47	93.68	95.79	95.79
	100	2	74.74	95.79	97.89	97.89	97.89
	500	4	71.58	91.58	93.68	93.68	94.74
	1000	4	72.63	88.42	93.68	93.68	94.74

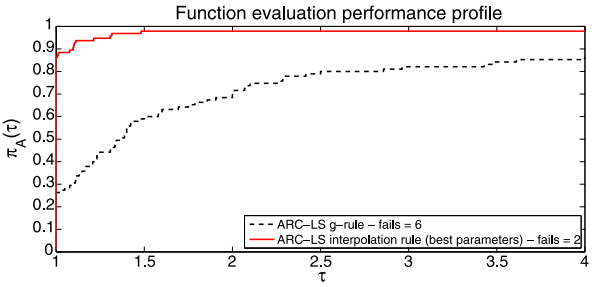

Fig. 7 The function evaluation performance profile: ARC-LS with (3.20) ("g-rule") and ARC-LS with Algorithm 3.1 and parameters (4.48) ("interpolation rule (best parameters)")

report the sets I_p for all the parameters in Algorithm 3.1, the efficiency measure (%pb$_{\hat{\tau}}$) for $\hat{\tau} = 1, 1.15, 1.25, 1.5, 2$ and the number of failures (#fails). We note that a more sophisticated choice, in which the globally optimal parameters for our test set is determined [1], is possible but has not been performed.

For each set I_p, it is quite easy to find the best performing parameter choice. It results from Table 2 that the new updating rule is not very sensitive to the parameter choice and that ARC-LS performs slightly better with the following parameter assignment:

$$\begin{aligned} &\beta = 1/100, \qquad \alpha_{max} = 2, \qquad \epsilon_\chi = 10^{-10}, \qquad \delta_1 = 1/10, \\ &\delta_2 = 1 \qquad \eta = \eta_1, \qquad \delta_3 = 2, \qquad \delta_{max} = 100. \end{aligned} \tag{4.48}$$

We remark that in the experiments, a solution α_β^* of (3.29) and (3.34) was always found and that only in a few cases this values was larger than α_{max}. Moreover, the value $\sigma_{k,\eta}^*$ computed by (3.41) was very often positive and lower than the current σ_k. Consequently, the regularization parameter was in fact updated by using the proposed interpolation techniques most of the time.

In Fig. 7, ARC-LS using Algorithm 3.1 and the parameters in (4.48) is compared with ARC-LS using the old rule (3.20) and $\gamma = 2$ employed in [3]. The new rule clearly outperforms the old one. A possible explanation of the relatively poor behaviour of ARC-LS with the old rule may be found in what follows. In the experiments, we noticed that the norm of the gradient oscillates considerably for some problems, resulting in high oscillations in the updated σ_k through the iterations. Furthermore, we observed that, using (3.20), σ_k was updated in several runs using a small $\|g_k\|$ and hence was considerably reduced; the next iterate was then unsuccessful and doubling σ_k to recover an acceptable σ_k gave rise to many unsuccessful iterations.

Finally, we compare TR-ST, TR-bST and ARC-LS using the best performing updating rules for the regularization parameters, i.e. for the trust-region radius Δ_k the rule (3.19) with the parameters in (4.46) but with $\eta = \eta_1$ and for the regularization parameter σ_k, the rule presented in Algorithm 3.1 with the parameter choice (4.48). The corresponding function evaluation performance profiles are plotted in Figs. 8 and 9. ARC-LS fails on problems ARWHDNE, DRCAVITY2, TR-bST on problems DRCAVITY2, POROUS2 and TR-ST on problems QR3D^2, POROUS2. Evidently, ARC-LS is much more efficient than TR-ST. Compared to TR-bST, it is better for

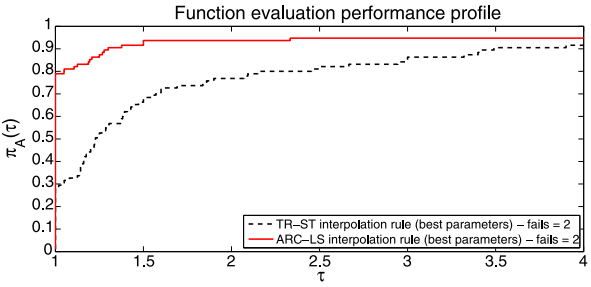

Fig. 8 The function evaluation performance profile: TR-ST rule (3.19) with $\eta = \eta_1$ ("interpolation rule (best parameters)") and ARC-LS with Algorithm 3.1 and parameters (4.48) ("interpolation rule (best parameters)")

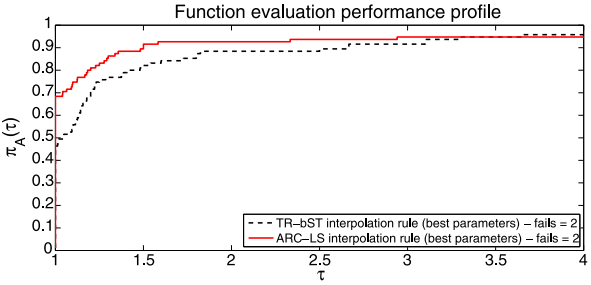

Fig. 9 The function evaluation performance profile: TR-bST rule (3.19) with $\eta = \eta_1$ ("interpolation rule (best parameters)") and ARC-LS with Algorithm 3.1 and parameters (4.48) ("interpolation rule (best parameters)")

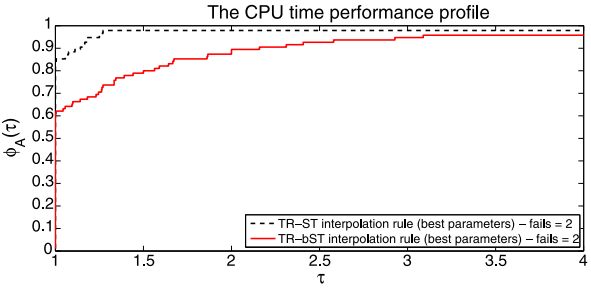

Fig. 10 The CPU time performance profile: TR-ST rule (3.19) with $\eta = \eta_1$ ("interpolation rule (best parameters)") and TR-bST rule (3.19) with $\eta = \eta_1$ ("interpolation rule (best parameters)")

the 68.42% of the runs and TR-bST is within a factor 2 of ARC-LS for the 88.10% of the runs.

We conclude this section adding some remarks on the performance of the algorithms in terms of CPU time. In Fig. 10 we plot the CPU time performance profile between TR-ST and TR-bST and, as expected, TR-ST is the most efficient since the second-phase is never activated. Concerning the behaviour of ARC-LS, its efficiency is directly related to the number t of stored basis vectors for the second-phase. In Fig. 11 we plot the CPU time performance profile between TR-ST and ARC-LS with increasing $t = 10, 100, 500$. It is important to remark that the extra cost of the second-phase is strictly dependent on the specific implementation used for solving the subproblems and is not an issue, e.g., if we follow a direct approach. Moreover, note that although for the CUTEr test problems, the cost of function and derivative evaluation is very low, had the evaluations been more expensive, the trend shown for function evaluations in Fig. 8 would have been repeated in Fig. 11.

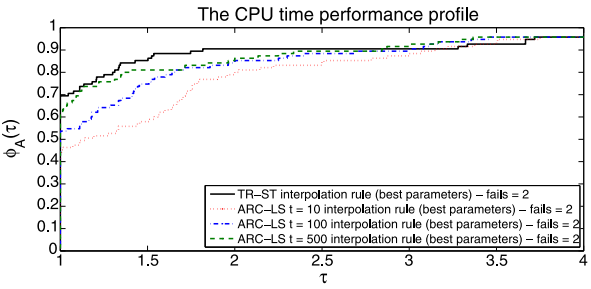

Fig. 11 The CPU time performance profile: `TR-ST` rule (3.19) with $\eta = \eta_1$ ("interpolation rule (best parameters)") and `ARC-LS` with Algorithm 3.1, parameters (4.48) ("interpolation rule (best parameters)") and number of stored basis vectors $t = 10, 100, 500$

We report in the Appendix the complete set of results of the experiments described in this section.

We also considered strategies for choosing the initial regularization parameter σ_0 along the lines of the strategy proposed in [23] for automatically computing the initial trust-region radius. In particular, we tested a strategy in which one solves a one-dimensional minimization problem (along the steepest descent direction) in the hope of estimating a better value of σ_0 for starting the minimization in the full space. However, these experiments (not reported here) produced disappointing results in that it turned out to be generally better to start minimization in the full-space from the start and not "waste" additional function evaluations for estimating σ_0. This is not entirely unexpected in our context where we assume the cost of function evaluation to dominate the inner linear algebra calculations. But it is also clear that any *a priori* user estimation of the Hessian Lipschitz constant can be usefully exploited by selecting σ_0 appropriately.

5 Conclusion

In this paper we propose a new reliable strategy to update the regularization parameter in the cubic regularization algorithm (ARC). This strategy is based on analyzing the adequacy between the objective function and its cubic model, and exploits its overestimation property. Moreover, it has the favorable feature of not requiring extra function values. We report numerical tests which show that the new rule considerably improves the numerical performance of the ARC algorithm. We also provide a numerical comparison between the ARC and trust-region frameworks on a set of large nonlinear least-squares CUTEr problems. These suggest a numerical advantage of the former on our set of test problems in a framework where the overall computational cost is dominated by the cost of a function evaluation.

Acknowledgements The work of the first author was supported by EPSRC grant EP/E053351/1. The second author wishes to thank Stefania Bellavia and Benedetta Morini for several helpful discussions and for their continued encouragement and support.

References

1. Audet, C., Orban, D.: Finding optimal algorithmic parameters using derivative-free optimization. SIAM J. Control Optim. **17**(3), 642–664 (2006)

2. Bellavia, S., Cartis, C., Gould, N.I.M., Morini, B., Toint, Ph.L.: Convergence of a regularized Euclidean residual algorithm for nonlinear least-squares. SIAM J. Numer. Anal. **48**, 1–29 (2010)
3. Cartis, C., Gould, N.I.M., Toint, Ph.L.: Adaptive cubic overestimation methods for unconstrained optimization. Part I: Motivation, convergence and numerical results. Math. Program., Ser. A **127**(2), 245–295 (2011)
4. Cartis, C., Gould, N.I.M., Toint, Ph.L.: Adaptive cubic overestimation methods for unconstrained optimization. Part II: Worst-case function- and derivative-evaluation complexity. Math. Program. Ser. A **130**(2), 295–319 (2011). doi:10.1007/s10107-009-0337-y
5. Cartis, C., Gould, N.I.M., Toint, Ph.L.: Complexity bounds for second-order optimality in unconstrained optimization. J. Complex **28**(1), 93–108 (2012). doi:10.1016/j.jco.2011.06.001
6. Cartis, C., Gould, N.I.M., Toint, Ph.L.: On the complexity of steepest descent, Newton's and regularized Newton's methods for nonconvex unconstrained optimization. SIAM J. Control Optim. **20**(6), 2833–2852 (2010)
7. Cartis, C., Gould, N.I.M., Toint, Ph.L.: Trust-region and other regularisations of linear least-squares problems. BIT **49**(1), 21–53 (2009)
8. Conn, A.R., Gould, N.I.M., Toint, Ph.L.: Trust-Region Methods. SIAM, Philadelphia (2000)
9. Dolan, E.D., Moré, J.J.: Benchmarking optimization software with performance profiles. Math. Program. **91**, 201–213 (2002)
10. Dennis, J.E., Schnabel, R.B.: Numerical Methods for Unconstrained Optimization and Nonlinear Equations. Prentice Hall, Englewood Cliffs (1983)
11. Erway, J.B., Gill, P.E.: A subspace minimization method for the trust-region step. SIAM J. Control Optim. **20**, 1439–1461 (2009)
12. Erway, J.B., Gill, P.E., Griffin, J.D.: Iterative methods for finding a trust-region step. SIAM J. Control Optim. **20**, 1110–1131 (2009)
13. Golub, G.H., Kahan, W.: Calculating the singular values and pseudo-inverse of a matrix. SIAM J. Numer. Anal. **2**(2), 205–224 (1965)
14. Gould, N.I.M., Orban, D., Sartenaer, A., Toint, Ph.L.: Sensitivity of trust-region algorithms to their parameters. 4OR **3**(3), 227–241 (2005)
15. Gould, N.I.M., Lucidi, S., Roma, M., Toint, Ph.L.: Solving the trust-region subproblem using the Lanczos method. SIAM J. Control Optim. **9**(2), 504–525 (1999)
16. Gould, N.I.M., Orban, D., Toint, Ph.L.: CUTEr, a constrained and unconstrained testing environment, revisited. ACM Trans. Math. Softw. **29**(4), 373–394 (2003)
17. Gould, N.I.M., Orban, D., Toint, Ph.L.: GALAHAD—a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. ACM Trans. Math. Softw. **29**(4), 353–372 (2003)
18. Griewank, A.: The modification of Newton's method for unconstrained optimization by bounding cubic terms. Technical Report NA/12 (1981), Department of Applied Mathematics and Theoretical Physics, University of Cambridge, United Kingdom (1981)
19. Hager, W.W., Park, S.C.: Global convergence of SSM for minimizing a quadratic over a sphere. Math. Comput. **74**, 1413–1423 (2005)
20. Hager, W.W.: Minimizing a quadratic over a sphere. SIAM J. Control Optim. **12**, 188–208 (2001)
21. Nesterov, Yu., Polyak, B.T.: Cubic regularization of Newton's method and its global performance. Math. Program. **108**(1), 177–205 (2006)
22. Paige, C.C., Saunders, M.A.: ALGORITHM 583: LSQR: an algorithm for sparse linear equations and sparse least squares. ACM Trans. Math. Softw. **8**(2), 195–209 (1982)
23. Sartenaer, A.: Automatic determination of an initial trust region in nonlinear programming. SIAM J. Sci. Comput. **18**(6), 1788–1803 (1997)
24. Steihaug, T.: The conjugate gradient method and trust regions in large scale optimization. SIAM J. Numer. Anal. **20**, 626–637 (1983)
25. Weiser, M., Deuflhard, P., Erdmann, B.: Affine conjugate adaptive Newton methods for nonlinear elastomechanics. Optim. Methods Softw. **22**(3), 413–431 (2007)
26. Yuan, Y.: On the truncated conjugate-gradient method. Math. Program., Ser. A **87**(3), 561–573 (1999)