Sensitivity of trust-region algorithms to their parameters

by N. Gould[1], D. Orban[2], A. Sartenaer[3] and Ph. L. Toint[3]

Report 04/07 August 24, 2004

[1] Computational Science and Engineering Departement,
Rutherford Appleton Laboratory,
Chilton, Oxfordshire, England.
email: n.gould@rl.ac.uk

[2] Département de Mathématiques et Génie Industriel,
Ecole Polytechnique de Montréal,
2900, Bd E. Montpetit, H3T 1J4 Montréal, Canada.
email: dominique.orban@polymtl.ca

[3] Department of Mathematics,
University of Namur,
61, rue de Bruxelles, B-5000 Namur, Belgium,
Email: annick.sartenaer@fundp.ac.be, philippe.toint@fundp.ac.be

Sensitivity of trust-region algorithms to their parameters

Nick I. M. Gould   Dominique Orban   Annick Sartenaer
Philippe L. Toint

August 24, 2004

Abstract

In this paper, we examine the sensitivity of trust-region algorithms on the parameters related to the step acceptance and update of the trust region. We show, in the context of unconstrained programming, that the numerical efficiency of these algorithms can easily be improved by choosing appropriate parameters. Recommanded ranges of values for these parameters are exhibited on the basis of extensive numerical tests.

Keywords: unconstrained programming, trust-region methods, algorithmic parameters.

1 Introduction

Trust-region methods form a popular class of iterative optimization algorithms, in which the objective function is approximated by a model and this model is minimized in a neighbourhood—the trust region—of the current iterate. Originally proposed in the context of nonlinear least-squares fitting [13, 15, 18], this class of methods was then developed [9, 19, 26] to form a robust theoretical and practical framework containing a number of locally convergent algorithms for smooth unconstrained minimization problems, and, in particular, Newton method. The reader is referred to [4, 16, 17] for additional motivation and convergence analysis.

Like many other algorithms, trust-region methods depend on the choice of a set of parameters, which specify when a trial step is deemed successful and how the trust-region size is updated as the iterations proceed. The purpose of this paper is to present an experimental study of the sensitivity, measured in terms of efficiency, of a trust-region algorithm for smooth unconstrained optimization as a function of these parameters.

We first present the trust-region algorithm in §2 and discuss the values of its parameters. We next describe the setting of our numerical experiments in §3, report on its results from the point of view of efficiency in §4. Some conclusions and perspectives are finally presented in §5.

2 The problem and algorithm

We consider the problem

$$\underset{x \in \mathrm{I\!R}^n}{\text{minimize}} \quad f(x), \tag{2.1}$$

where f is a twice continuously differentiable function from $\mathrm{I\!R}^n$ into $\mathrm{I\!R}$. We assume that the problem is well defined in that f is bounded below. The philosophy of trust-region methods is to calculate, at iterate x_k, a model m_k of the objective function in the *trust region*

$$\mathcal{B}_k = \{x_k + s \mid \|s\| \leq \Delta_k\},$$

where $\|\cdot\|$ is the Euclidean norm on $\mathrm{I\!R}^n$ and $\Delta_k > 0$ is the *trust-region radius*. A step s_k is then computed that approximately minimizes this model within the trust region. If the value of the objective function computed at the trial point $x_k + s_k$ produces a decrease in the objective function which is comparable to that predicted by the model, the trial point is accepted as the next iterate and the trust-region radius is possibly increased. Otherwise, the trial step is rejected and the radius decreased. More formally, our algorithm is defined as Algorithm 2.1.

A number of choices are possible for the model m_k. Let $\langle \cdot, \cdot \rangle$ denote the usual inner product in $\mathrm{I\!R}^n$. In what follows, we focus on a quadratic model of the form

$$m_k(x_k + s) = f(x_k) + \langle \nabla_x f(x_k), s \rangle + \tfrac{1}{2}\langle s, \nabla_{xx} f(x_k) s \rangle, \tag{2.4}$$

which is sometimes known as *Newton's model*. Note that the model and objective function coincide up to second order at x_k, i.e., $m_k(x_k) = f(x_k)$, $\nabla_x m_k(x_k) = \nabla_x f(x_k)$ and $\nabla_{xx} m_k(x_k + s) = \nabla_{xx} f(x_k)$ for all s.

We will not expand in full detail on what we mean by "sufficient model reduction" in Step 2. If we define the *Cauchy point* x_k^{C} to be a model minimizer along the intersection of the steepest descent direction and the trust region, we say that a step s_k produces sufficient decrease if and only if

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{\text{red}} \left[m_k(x_k) - m_k(x_k^{\mathrm{C}}) \right], \tag{2.5}$$

for some $\kappa_{\text{red}} \in (0, 1)$. The quantity $m_k(x_k) - m_k(x_k + s_k)$ is referred to as the *predicted decrease*. Condition (2.5) is known to ensure global convergence of the algorithm to first-order critical points of problem (2.1) under suitable assumptions [4], [16]. In the next section, we return to the choice of an algorithm ensuring (2.5) when the model (2.4) is considered.

The algorithm depends on the constants η_1, η_2, α_1 and α_2, whose values are only restricted to satisfy (2.2). The values

$$\eta_1 = 0.25, \quad \eta_2 = 0.75 \tag{2.6}$$

(see, for instance, [1, 2, 14, 20, 21, 22]) and

$$\alpha_1 = 0.5 \quad \text{and} \quad \alpha_2 = 2 \tag{2.7}$$

Algorithm 2.1: The Basic Trust-Region Algorithm

Step 0. [Initialization] An initial point x_0 and an initial trust-region radius Δ_0 are given, as well as the parameters η_1, η_2, α_1 and α_2 that satisfy

$$0 \le \eta_1 < \eta_2 < 1 \text{ and } 0 < \alpha_1 < 1 < \alpha_2. \tag{2.2}$$

Compute $f(x_0)$ and set $k = 0$.

Step 1. [Model definition] Define a model $m_k(x_k + s)$ of $f(x_k + s)$ in $\mathcal{B}_k$.

Step 2. [Step calculation] Compute a step s_k that "sufficiently reduces the model" m_k and such that $x_k + s_k \in \mathcal{B}_k$.

Step 3. [Acceptance of the trial point] Compute $f(x_k + s_k)$ and

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}. \tag{2.3}$$

If $\rho_k \ge \eta_1$, then set $x_{k+1} = x_k + s_k$; otherwise, set $x_{k+1} = x_k$.

Step 4. [Trust-region radius update] Set

$$\Delta_{k+1} = \begin{cases} \alpha_1 \|s_k\| & \text{if} \quad \rho_k < \eta_1 \\ \Delta_k & \text{if} \quad \eta_1 \le \rho_k < \eta_2 \\ \max[\alpha_2 \|s_k\|, \Delta_k] & \text{if} \quad \rho_k \ge \eta_2. \end{cases}$$

Increment k by one, and loop back to Step 1.

(see, for instance, [2, 5, 7, 20, 22]) have been used in practical implementations. Is the behaviour of the algorithm relatively insensitive to variations in these values? The purpose of this note is to examine this question by analyzing the performance of the algorithm for a set of alternative parameter values over a reasonable set of test problems.

The choice of η_1 also has a theoretical implication: choosing $\eta_1 > 0$ is known to be necessary in order to guarantee that all limit points of the sequence of iterates satisfy the first-order optimality conditions [16, 24, 27]. It is thus also of interest to investigate the effect of this condition on numerical efficiency.

The authors are aware that, at least the choice of α_1 and α_2 may be made less crucial if an interpolation scheme is used to determine the value of Δ_{k+1} from that of $f(x_k), f(x_k + s_k)$ and $\nabla_x f(x_k)$ [3, 6, 14]. However, good values for these parameters remain helpful in simpler implementations of the trust-region method.

3 The framework for the numerical experiments

We now turn to the discussion of the framework in which our numerical experiments are conducted. Obviously, such an experimental investigation is never perfect, and several of the choices made here, although reasonable from our point of view, are not the only ones that one could consider. We are well aware of the limitations of our approach. We briefly discuss them and propose directions for additional research in §5.

A first decision involves selecting a suitable performance measure for the algorithm. We have chosen to measure algorithm efficiency by the number of iterations to obtain convergence, which is declared as soon as

$$\|\nabla_x f(x_k)\| \leq 10^{-5}. \tag{3.8}$$

(We are aware that relative/weighted tests might be more useful for badly-scaled/non-linear problems.) We had intended to declare failure when (3.8) was not met in the first 1000 iterations of the algorithm, but this situation never happened in our tests. Note that since we are only considering unconstrained minimization, this measure in terms of iteration counts is equivalent to considering the number of function evaluations and is justified in the frequent case where objective function evaluations are computationally costly and dominate the internal work of the algorithm. If evaluating the objective function is relatively cheap compared to the algorithm's internal work, which might be the case when the dimension of the problem is large and the linear algebra therefore more expensive, then overall CPU time is the obvious choice for measuring performance.

The second important choice is that of the procedure to compute, at each iteration, a trial step s_k that approximately minimizes the model within the trust region. We have chosen to use the truncated conjugate-gradient iteration, or Steihaug-Toint algorithm [23, 25], as implemented in the GLTR module of the GALAHAD library [10, 12]. In this method, the iterates generated by the conjugate-gradient algorithm are used until either they leave the trust region or negative curvature is discovered—in either of these cases the last conjugate-gradient step is truncated on the trust-region boundary. Thus, the Steihaug-Toint procedure terminates at $x_k + s$ that satisfies

$$\|\nabla_x f(x_k + s)\| \leq \min\left[\frac{1}{10}, \|\nabla_x f(x_k)\|^{1/2}\right] \quad \text{or} \quad \|s\| = \Delta_k.$$

This procedure is known to guarantee (2.5).

We decided to test the algorithm on what we believe is a reasonably representative set of 24 problems from the CUTEr collection [11], whose average dimension is around 1000. All problems are unconstrained, with some of them being highly nonlinear. These problems have proved to be reasonably hard in the past. Their name and size can be found in Figure 3. The number of problems in our test set had to be kept relatively low in order to make the test for a large number of parameter values tractable.

Finally, we decided on a set of parameter values to be experimented with. In a first

problem name	dimension	problem name	dimension
BIGGSB1	1000	NCB20B	1000
CURLY10	1000	NONCVXU2	1000
CURLY20	1000	NONDIA	1000
CURLY30	1000	NONDQUAR	1000
EDENSCH	2000	PENALTY1	1000
EIGENBLS	1056	POWER	1000
FREUROTH	1000	QUARTC	1000
GENROSE	1000	SINQUAD	1000
LINVERSE	999	SPARSINE	1000
MSQRTALS	1024	SPMSRTLS	1000
MSQRTBLS	1024	VAREIGVL	999
NCB20	1000	WOODS	1000

Figure 3.1: The test problems set from the `CUTEr` collection

stage, we chose to let (η_1, η_2) vary in the (coarse) uniform grid

$$G_\eta^{\text{C}} = \left\{ (\eta_1, \eta_2) \;\middle|\; \begin{array}{l} \eta_1 \in \{0, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45\}, \\ \\ \eta_2 \in \{0.5, 0.6, 0.7, 0.8, 0.9\} \end{array} \right\},$$

while (α_1, α_2) varies in the grid

$$G_\alpha = \left\{ (\alpha_1, \alpha_2) \;\middle|\; \begin{array}{l} \alpha_1 \in \{0.25, 0.33, 0.5, 0.66, 0.75\}, \\ \\ \alpha_2 \in \{1.5, 2.0, 2.5, 3.0, 3.5, 5\} \end{array} \right\}.$$

Preliminary experiments using the grids G_η^{C} and G_α seemed to indicate that the region

$$\mathcal{Z} = \{0 \le \eta_1 \le 0.1, \quad 0.7 \le \eta_2 < 1\}$$

was worth being discretized more finely as it appeared to be where the best efficiency would occur. As a result, the coarse grid G_η^{C} was replaced by the finer, but no longer uniform, grid

$$G_\eta^{\text{F}} = \left\{ (\eta_1, \eta_2) \;\middle|\; \begin{array}{l} \eta_1 \in \{0, 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4\} \\ \\ \eta_2 \in \{0.5, 0.6, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 0.99, 0.995, 0.999\} \end{array} \right\}.$$

The grid G_α remained unchanged. Altogether, this gives a set of 3,960 different values for $(\eta_1, \eta_2, \alpha_1, \alpha_2)$. We then applied the algorithm using these values on the 24 test problems, which resulted in a grand total of 95,040 test runs. All tests were performed in double precision Fortran 90 on an 1.6GHz Intel Pentium IV Linux PC with 512 MBytes of RAM.

4 Numerical results and analysis

4.1 Combined performance

Remarkably, optimizing the trust-region algorithm parameters seems to depend only marginally on whether one aims for a minimum number of iterations (and thus of function/derivatives calculations) or for a minimum CPU time. The performance measured in these two quantities seems indeed very related, as is clear from Figure 4.2, where the couples consisting of the average iteration count and the average CPU time (the averages being taken on all test problems) are plotted for each of the 3,960 tested variants. In this figure, the performance of the "standard" parameter choice, as defined by (2.6) and (2.7), is identified as the point at the intersection of the vertical and horizontal cross lines.

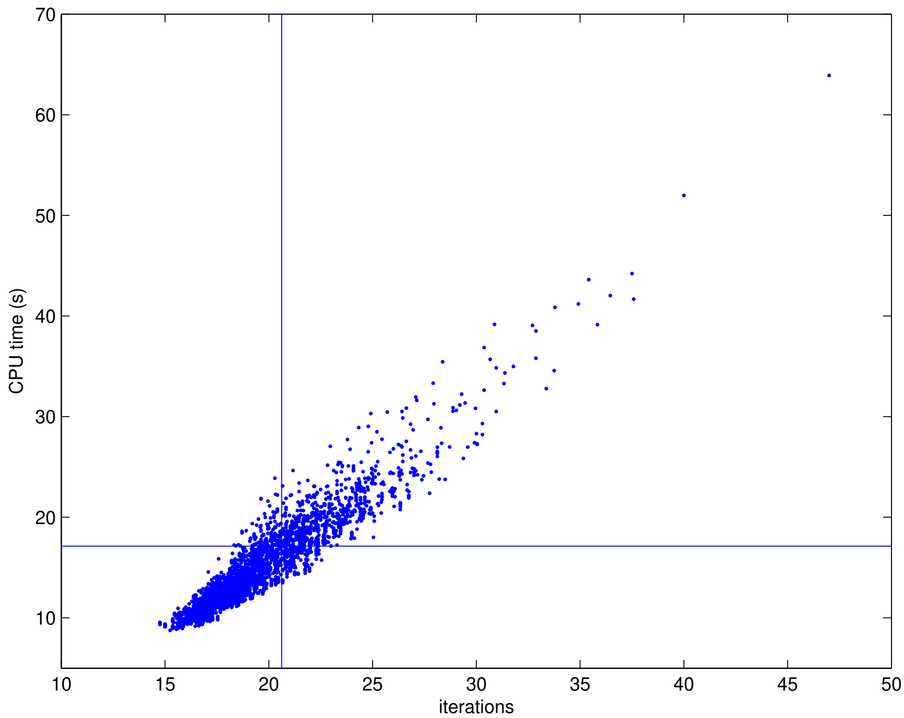

Figure 4.2: Combined performance of all 3,960 tested variants (standard parameter choice marked by crosshair)

The "comet-like" structure of the cloud of performance couples is very elongated for our test set. Moreover, the densest "core" part of the cloud appears to correspond to relatively efficient variants, the worst ones being quite far in the sparser "tail". The standard variant belongs to the core, but just barely, which shows that many significantly better parameter choices exist.

Figure 4.3 is obtained by zooming on the tip of the core, i.e., the part of the figure containing the most efficient variants. This figure shows several vertical clusters of

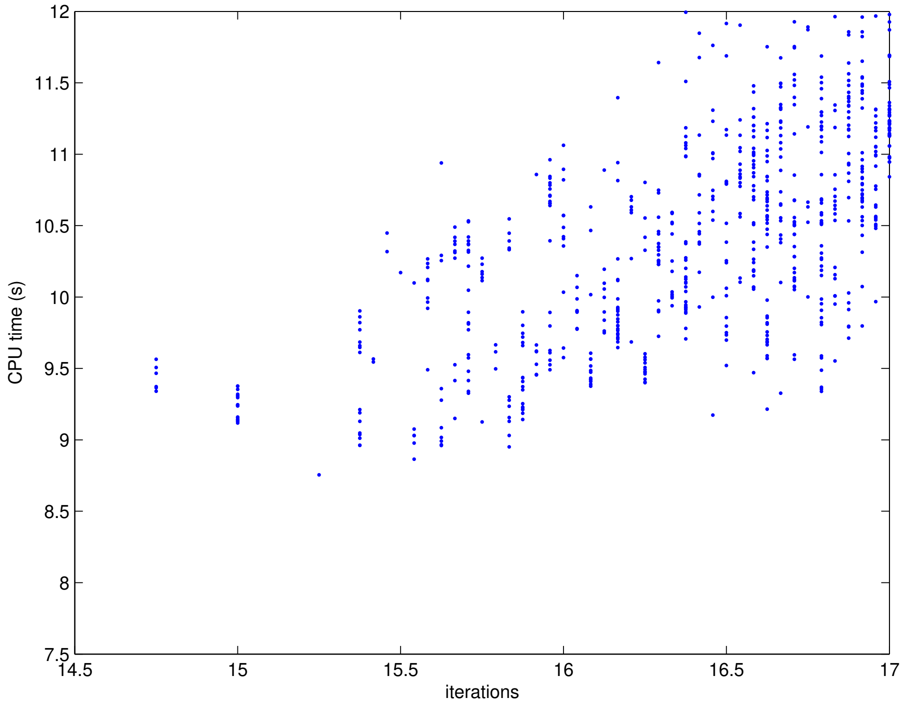

Figure 4.3: Combined performance of the more efficient tested variants

points corresponding to variants using exactly the same average number of iterations and slightly different CPU times. These clusters reflect the inaccuracy of the timing routine used (a few percent), and should therefore be interpreted as a single combined performance measure.

We now explore in more details the clusters of the efficient frontier of the above figures. This frontier is defined as the set of clusters that are not dominated by any other cluster, that is the clusters such that there is no other cluster in the figure giving better average number of iterations *and* better average CPU time.

The first cluster on the left corresponds to the best variant in terms of iterations. The associated performance is presented in Table 4.1, which shows the associated parameter values, the total number of iterations and the CPU time for this cluster, for the worst choice of the parameters, and for the standard choice. Note that the CPU time in brackets is a rounded average of the times associated with all the variants in the entire cluster.

We see that the best parameter choice results in an algorithm that requires, on average, $14.750/20.625 = 0.715$ as many iterations as the standard one. On the other hand, the worst choice produces a method that needs, on average, $47/14.75 = 3.19$ as many iterations as the best, and $47/20.625 = 2.28$ as many as the standard. The proportions in time reinforce these trends.

Another outcome of our tests is that the performance of the variants cannot be

	η_1	η_2	α_1	α_2	# its	CPU
Parameter choices for first cluster	$[0,10^{-2}]$	0.99	0.25	3.5	14.750	(9.45)
Standard choice	0.25	0.75	0.5	2	20.625	17.13
Worst choice	0.4	0.5	0.75	5	47.000	63.90

Table 4.1: Average iteration counts and CPU times (s) for the best, standard and worst parameter choices in terms of iterations

distinguished if they differ only by the choice of η_1 in the range 0 to 10^{-4}. For most choices of the other parameters, this is also the case for η_1 up to 10^{-2}. This is interesting because it seems to imply that a small strictly positive value of η_1 is as good as possible from the numerical efficiency point of view, while, at the same time, ensuring the best theoretical convergence properties. Conversely, this indicates that choosing $\eta_1 = 0$ may not have a numerically detrimental effect despite being theoretically less satisfactory. This behaviour is also apparent in Table 4.1.

Pursuing our exploration of the efficient frontier in Figure 4.3, the next dominant cluster is that corresponding to variants taking on average 15 iterations. It is in fact made up from two similar but distinct choices of parameters, as indicated in Table 4.2. Strictly speaking, only the first of those is really part of the efficient frontier, since it marginally dominates the second.

	η_1	η_2	α_1	α_2	# its	CPU
Parameter choices	$[0,\ 10^{-1}]$	$\{0.995, 0.999\}$	0.25	5	15.000	(9.15)
for middle cluster	$[0,\ 10^{-4}]$	0.999	0.33	3.5	15.000	(9.35)
Standard choice	0.25	0.75	0.5	2	20.625	17.13
Worst choice	0.4	0.5	0.75	5	47.000	63.90

Table 4.2: Average iteration counts and CPU times (s) for the middle dominant cluster of variants, and for the standard and worst ones

Algorithms using these choices of parameters require, on average, 15/20.625=0.727 as many iterations as the standard choice and 15/47= 0.319 as many as the worst. Again, the ratios for CPU time confirm the trends observed for iterations.

The third and last point on the efficient frontier is given by the variant which is best in terms of CPU time (i.e., is lowest in Figure 4.3). Table 4.3 shows the (unique) set of parameter values and total CPU time for this variant, and compares them again with the standard and worst choices.

Following the same logic as above, we note that this parameter choice results in an algorithm that is, on average, $17.13/8.76 = 1.96$ as fast as the standard one. On the other hand, the worst choice produces a method that is, on average, $63.90/8.76 = 7.3$ times slower than the best, and $63.90/17.13 = 3.73$ slower than the standard. We note that this best variant in CPU time is close to those in the middle cluster, except for a surprisingly larger value of η_1. We might thus see it as the result of a particularly

	η_1	η_2	α_1	α_2	# its	CPU
Parameter choice for third cluster	0.15	0.999	0.33	5	15.25	8.76
Standard choice	0.25	0.75	0.5	2	20.625	17.13
Worst choice	0.4	0.5	0.75	5	47.000	63.90

Table 4.3: Average iteration counts and CPU times (s) for the best, standard and worst parameter choices in terms of CPU time

lucky set of time measures.

4.2 Iterations sensitivity

We now examine the sensitivity in iterations of the above conclusions for the parameter choices of the first cluster (most efficient in terms of iterations).

η_1 \ η_2	0.50	0.60	0.70	0.75	0.80	0.85	0.90	0.95	0.99	0.995	0.999
0.000000	0.75	0.79	0.80	0.78	0.75	0.76	0.82	0.83	$\boxed{1.00}$	0.89	0.94
0.000001	0.75	0.79	0.80	0.78	0.75	0.76	0.82	0.83	$\boxed{1.00}$	0.89	0.94
0.000010	0.75	0.79	0.80	0.78	0.75	0.76	0.82	0.83	$\boxed{1.00}$	0.89	0.94
0.000100	0.75	0.79	0.80	0.78	0.75	0.76	0.82	0.83	$\boxed{1.00}$	0.89	0.94
0.001000	0.75	0.79	0.80	0.78	0.75	0.76	0.82	0.83	$\boxed{1.00}$	0.89	0.94
0.010000	0.75	0.79	0.80	0.78	0.75	0.76	0.82	0.83	$\boxed{1.00}$	0.89	0.94
0.100000	0.69	0.78	0.75	0.80	0.80	0.78	0.86	0.84	0.91	0.83	0.87
0.150000	0.72	0.72	0.75	0.77	0.80	0.74	0.76	0.78	0.95	0.86	0.90
0.200000	0.74	0.70	0.74	0.75	0.73	0.77	0.81	0.85	0.92	0.90	0.91
0.250000	0.70	0.72	0.70	0.78	0.72	0.76	0.84	0.77	0.90	0.86	0.87
0.300000	0.70	0.66	0.72	0.78	0.71	0.76	0.76	0.78	0.91	0.90	0.89
0.400000	0.71	0.72	0.72	0.74	0.73	0.77	0.80	0.80	0.88	0.87	0.86

Table 4.4: Relative performance in iterations as a function of η_1 and η_2 for fixed $\alpha_1 = 0.25$ and $\alpha_2 = 3.5$

Table 4.4 presents the relative performance, compared to the best, of all considered choices of η_1 and η_2 when α_1 and α_2 are fixed to their optimal value (see Table 4.1). We note in this table that, as indicated above, the results are identical for all values of η_1 that are between 0 and 0.01. This indicates that small values of this parameter are best, but also that the precise choice of a small value of η_1 is not crucial. A second observation is that a choice of η_2 larger that 0.99 is comparatively better than that of a smaller value. Globally the performance degrades nearly monotonically when η_1 grows and/or η_2 decreases.

The relative performance of all considered choices of α_1 and α_2 (for values of η_1 and η_2 fixed to one of their optimal values) are shown in Table 4.5. Again, the neigbouhood of the best values is clear. One notices that the optimal values for α_2, around 3.5, are considerably larger than their standard (2). It therefore seems more efficient to increase the radius relatively unfrequently (η_2 close to 1) but to do so more decisively when it happens. This is supported by the observation that the choice $\alpha_2 = 5$ remains very satisfactory for $\alpha_1 = 0.25$ and $\alpha_1 = 0.33$ (see also Tables 4.2 and 4.3). The best value $\alpha_1 = 0.25$ shows that unsuccessful iterations ($\rho_k < \eta_1$), although not frequent, should not reduce the trust-region radius too aggressively.

α_2 / α_1	1.50	2.00	2.50	3.00	3.50	5.00
0.10	0.79	0.87	0.90	0.87	0.94	0.92
0.25	0.79	0.81	0.90	0.93	$\boxed{1.00}$	0.95
0.33	0.83	0.92	0.88	0.90	0.93	0.94
0.50	0.83	0.86	0.85	0.85	0.79	0.93
0.75	0.71	0.71	0.74	0.72	0.72	0.68

Table 4.5: Relative performance in iterations as a function of α_1 and α_2 for fixed $\eta_1 = 10^{-5}$ and $\eta_2 = 0.99$

We conclude our sensitivity analysis in terms of number of iterations by considering the performance profiles [8] for the worst, standard and best parameter choices. Such profiles are defined as follows. Assume a certain set $\mathcal{A}$ of competing algorithms $\mathcal{A}_1, \ldots, \mathcal{A}_q$ is tested on a set $\mathcal{S}$ of p test problems and algorithm $\mathcal{A}_i$ reports a certain measure of performance $\pi_{i,t}$ (iteration counts, in our case) when run on test problem t such that Algorithm $\mathcal{A}_i$ is "better" than Algorithm $\mathcal{A}_j$ on this problem t if $\pi_{i,t} \leq \pi_{j,t}$. If we define the *indicator* function

$$\kappa(\pi_1, \pi_2; \sigma) \equiv \begin{cases} 1 & \text{if } \pi_1 \leq \sigma\pi_2, \\ 0 & \text{otherwise,} \end{cases}$$

the *performance profile* of Algorithm $\mathcal{A}_i$ on the test set $\mathcal{S}$ with respect to the performance measure π is defined as the function

$$\pi_i(\sigma) \equiv \frac{\sum_{t=1}^{p} \kappa(\pi_{i,t}, \pi_{\min,t}; \sigma)}{p}, \quad \text{for every } \sigma > 0,$$

where $\pi_{\min,t} = \min_{i=1,\ldots,q} \pi_{i,t}$ is the best perfomance achieved on problem t. Hence, $\pi_i(\sigma)$ is a measure of the probability for the performance of Algorithm $\mathcal{A}_i$ to be within a factor σ of the performance of the best algorithm. For instance, $\pi_i(1)$ measures the probability for the performance of Algorithm $\mathcal{A}_i$ to be the best, and $\lim_{\sigma\to\infty} \pi_i(\sigma)$ is the probability that Algorithm $\mathcal{A}_i$ solves a problem.

The iteration performance profiles for the algorithms corresponding to the worst, standard and best parameter choices are presented in Figure 4.4. Interestingly, the standard choice is less often the best ($\sigma = 1$) than the worst and best.

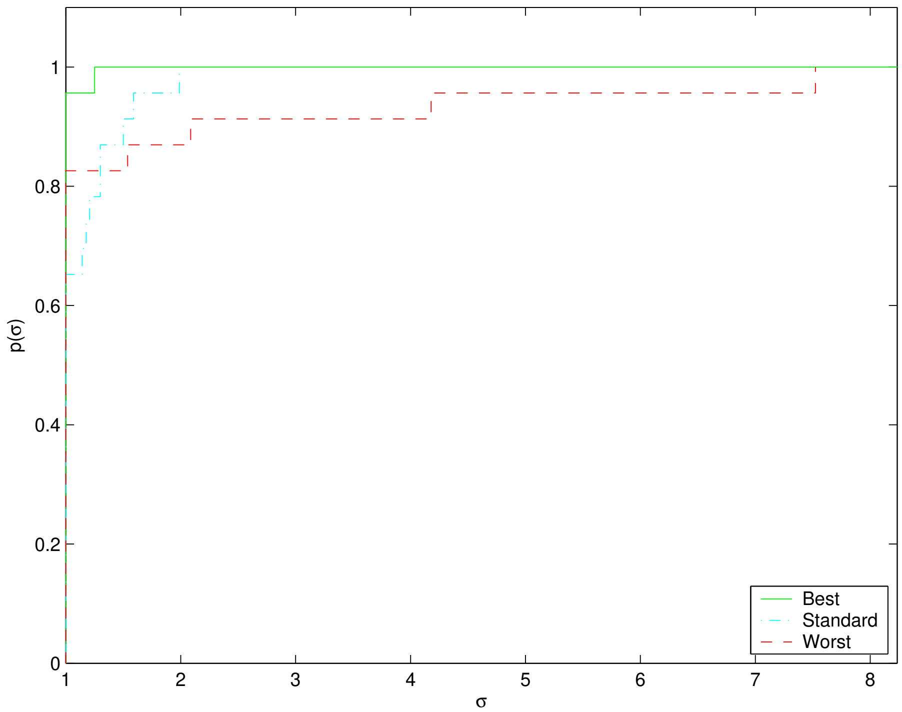

Figure 4.4: Iteration performance profiles for the best, standard and worst parameter choices

4.3 CPU time sensitivity

We now turn to sensitivity in CPU time and analyze the third cluster of the efficient frontier, which is the most efficient for this criterion.

η_1 \ η_2	0.50	0.60	0.70	0.75	0.80	0.85	0.90	0.95	0.99	0.995	0.999
0.000000	0.47	0.44	0.46	0.47	0.51	0.63	0.72	0.86	0.97	0.97	0.89
0.000001	0.46	0.44	0.46	0.47	0.51	0.64	0.72	0.86	0.98	0.97	0.89
0.000010	0.46	0.44	0.45	0.47	0.51	0.63	0.72	0.86	0.96	0.97	0.88
0.000100	0.45	0.43	0.45	0.47	0.51	0.62	0.72	0.85	0.96	0.96	0.87
0.001000	0.45	0.43	0.45	0.46	0.50	0.61	0.70	0.85	0.96	0.97	0.87
0.010000	0.45	0.43	0.45	0.55	0.51	0.62	0.71	0.91	0.96	0.96	0.87
0.100000	0.32	0.40	0.43	0.44	0.57	0.60	0.79	0.91	0.97	0.92	0.97
0.150000	0.28	0.35	0.41	0.43	0.54	0.50	0.74	0.92	0.93	0.96	$\boxed{1.00}$
0.200000	0.31	0.34	0.42	0.44	0.48	0.52	0.82	0.73	0.85	0.86	0.86
0.250000	0.32	0.39	0.39	0.44	0.44	0.49	0.76	0.73	0.87	0.91	0.91
0.300000	0.30	0.34	0.41	0.42	0.45	0.53	0.74	0.72	0.90	0.85	0.85
0.400000	0.28	0.34	0.39	0.37	0.48	0.50	0.66	0.72	0.81	0.80	0.80

Table 4.6: Relative performance in CPU time as a function of η_1 and η_2 for fixed $\alpha_1 = 0.33$ and $\alpha_2 = 5$

Table 4.6 presents the relative performance in CPU time of all considered choices of η_1 and η_2 when α_1 and α_2 are fixed to their optimal value (see Table 4.3). Although the best performance is obtained for $\eta_1 = 0.15$ and $\eta_2 = 0.999$, we see that the choices of η_1 between 0 and 0.1 for $\eta_2 = 0.99$ and 0.995 remain highly efficient. The general conclusions in terms of CPU time are thus very consistent with those obtained when considering iteration counts: small values of η_1 are best and a choice of η_2 larger that 0.99 is comparatively better than that of a smaller value.

The relative CPU time performance of all considered choices of α_1 and α_2 (for values of η_1 and η_2 fixed to their optimum) are finally shown in Table 4.7. Again, the conclusions follow the lines stated above: values of α_1 should be moderate while values of α_2 should definitely exceed the standard choice of 2.

α_2 / α_1	1.50	2.00	2.50	3.00	3.50	5.00
0.10	0.67	0.77	0.87	0.92	0.74	0.90
0.25	0.79	0.78	0.87	0.79	0.82	0.94
0.33	0.75	0.89	0.88	0.78	0.79	$\boxed{1.00}$
0.50	0.72	0.78	0.67	0.71	0.73	0.69
0.75	0.58	0.66	0.56	0.47	0.49	0.54

Table 4.7: Relative performance in CPU time as a function of α_1 and α_2 for fixed $\eta_1 = 0.15$ and $\eta_2 = 0.999$

Finally, Figure 4.7 shows the CPU-time performance profiles for the best, standard and worst parameter choices. Again the profile for the best choice indicates a clear superiority of this variant on the other two.

4.4 Tentative recommendations

Recommending parameter values in general is always somewhat risky. The actual performance of an algorithm may indeed strongly depend on the class of problems on which it is applied, and the conclusions of this study rely on a particular test-problem set. The values suggested here are therefore best seen as appropriate for general purpose use and are not intended to replace application dependent choices. This said, it seems that a reasonable parameter choice could be

$$\eta_1 = 0.0001, \quad \eta_2 = 0.99, \quad \alpha_1 = 0.25, \quad \alpha_2 = 3.5.$$

If distinguishing between the two measures of efficiency is critical, the values suggested in Sections 4.2 and 4.3 might be considered, but direct experience remains of course the best source of inspiration.

5 Conclusions

We have performed simple, systematic, tests with a basic trust-region algorithm for unconstrained programming. The test problems are taken from the CUTEr collection

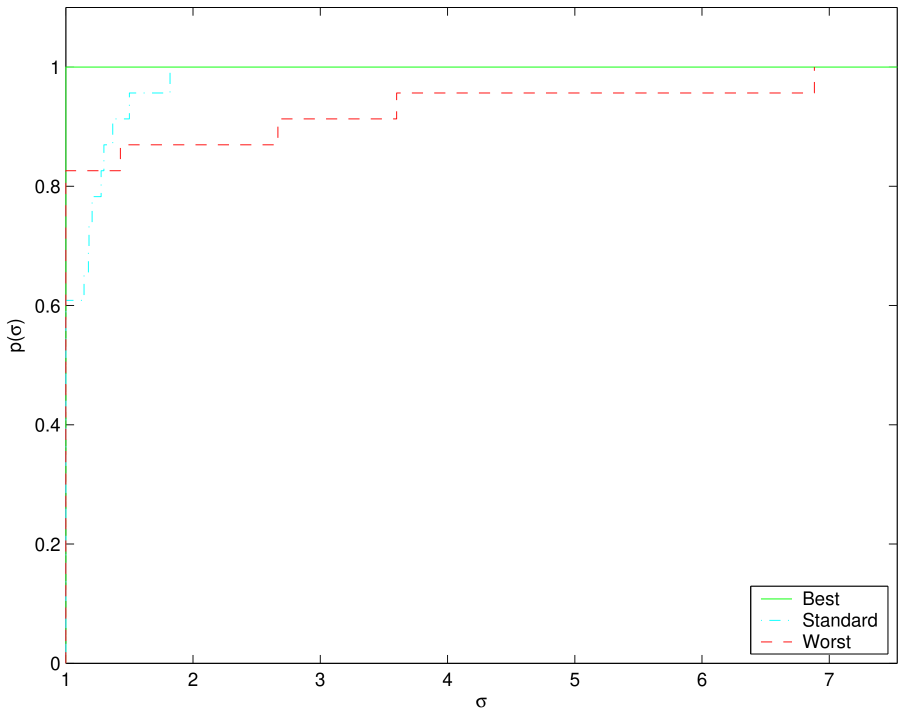

Figure 4.5: CPU time performance profiles for the best, standard and worst parameter choices

and our algorithm underwent tests for nearly 4000 values of the trust-region parameters. The commonly used "standard" values for these parameters appear not to be the best choice and alternative values have been pointed out, emphasizing the gain that these could produce over the standard values.

Only simple, multiplicative, updating rules have been used in this algorithmic framework. The authors are well aware that other rules could have been tested and that a much finer grid could have been used. The values of the α parameters could also be determined dynamically using a second or third degree polynomial interpolation of the objective function, or of the ρ function. While the conclusions drawn here therefore remain tentative and dependent on a particular set of test problems, the authors believe that they may be of interest for algorithm developers.

References

[1] T. F. Coleman and Y. Li. An interior trust region approach for nonlinear minimization subject to bounds. *SIAM Journal on Optimization*, 6(2):418–445, 1996.

[2] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, 50:399–430, 1988.

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

[4] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods.* Number 01 in MPS-SIAM Series on Optimization. SIAM, Philadelphia, USA, 2000.

[5] J. E. Dennis, M. Heinkenschloss, and L. N. Vicente. Trust-region interior-point SQP algorithms for a class of nonlinear programming problems. *SIAM Journal on Control and Optimization*, 36(5):1750–1794, 1998.

[6] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1983. Reprinted as *Classics in Applied Mathematics 16*, SIAM, Philadelphia, USA, 1996.

[7] J. E. Dennis and L. N. Vicente. Trust region interior-point algorithms for minimization problems with simple bounds. In H. Fisher, B. Riedmüller, and S. Schäffler, editors, *Applied Mathematics and Parallel Computing, Festschrift for Klaus Ritter*, pages 97–107, Heidelberg, Berlin, New York, 1996. Physica-Verlag, Springer-Verlag.

[8] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[9] S. M. Goldfeldt, R. E. Quandt, and H. F. Trotter. Maximization by quadratic hill-climbing. *Econometrica*, 34:541–551, 1966.

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

[11] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr, a contrained and unconstrained testing environment, revisited. *ACM Transactions on Mathematical Software*, 29(4):373–394, 2003.

[12] N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD—a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Transactions on Mathematical Software*, 29(4):353–372, 2003.

[13] K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly Journal on Applied Mathematics*, 2:164–168, 1944.

[14] C. Lin and J. J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.

[15] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11:431–441, 1963.

[16] J. J. Moré. Recent developments in algorithms and software for trust region methods. In A. Bachem, M. Grötschel, and B. Korte, editors, *Mathematical Programming: The State of the Art*, pages 258–287, Heidelberg, Berlin, New York, 1983. Springer Verlag.

[17] J. J. Moré and D. C. Sorensen. Newton's method. In G. H. Golub, editor, *Studies in Numerical Analysis*, number 24 in MAA Studies in Mathematics, pages 29–82, Providence, Rhode-Island, USA, 1984. American Mathematical Society.

[18] D. D. Morrison. Methods for nonlinear least squares problems and convergence proofs. In J. Lorell and F. Yagi, editors, *Proceedings of the Seminar on Tracking Programs and Orbit Determination*, pages 1–9, Pasadena, USA, 1960. Jet Propulsion Laboratory.

[19] M. J. D. Powell. A new algorithm for unconstrained optimization. In J. B. Rosen, O. L. Mangasarian, and K. Ritter, editors, *Nonlinear Programming*, pages 31–65, London, 1970. Academic Press.

[20] A. Sartenaer. Armijo-type condition for the determination of a generalized Cauchy point in trust region algorithms using exact or inexact projections on convex constraints. *Belgian Journal of Operations Research, Statistics and Computer Science*, 33(4):61–75, 1993.

[21] Ch. Sebudandi and Ph. L. Toint. Nonlinear optimization for seismic travel time tomography. *Geophysical Journal International*, 115:929–940, 1993.

[22] J. S. Shahabuddin. *Structured trust-region algorithms for the minimization of nonlinear functions*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, New York, USA, 1996.

[23] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[24] S. Thomas. *Sequential estimation techniques for quasi-Newton algorithms*. PhD thesis, Cornell University, Ithaca, New York, USA, 1975.

[25] Ph. L. Toint. Towards an efficient sparsity exploiting Newton method for minimization. In I. S. Duff, editor, *Sparse Matrices and Their Uses*, pages 57–88, London, 1981. Academic Press.

[26] D. Winfield. Function minimization by interpolation in a data table. *Journal of the Institute of Mathematics and its Applications*, 12:339–347, 1973.

[27] Y. Yuan. An example of non-convergence of trust region algorithms. In Y. Yuan, editor, *Advances in Nonlinear Programming*, pages 205–218, Dordrecht, The Netherlands, 1998. Kluwer Academic Publishers.