RAIRO-Oper. Res. 43 (2009) 309–329
DOI: 10.1051/ro/2009016

RAIRO Operations Research
www.rairo-ro.org

HOW MUCH DO APPROXIMATE DERIVATIVES HURT FILTER METHODS?

Caroline Sainvitu[1]

Abstract. In this paper, we examine the influence of approximate first and/or second derivatives on the filter-trust-region algorithm designed for solving unconstrained nonlinear optimization problems and proposed by Gould, Sainvitu and Toint in [12]. Numerical experiments carried out on small-scaled unconstrained problems from the `CUTEr` collection describe the effect of the use of approximate derivatives on the robustness and the efficiency of the filter-trust-region method.

Keywords. Unconstrained optimization, filter techniques, trust-region algorithms, approximate derivatives, numerical results.

Résumé. Dans ce papier, nous examinons l'influence des dérivées premières et secondes approximées sur l'algorithme de filtre de type région de confiance développé pour résoudre des problèmes d'optimisation non-linéaire sans contraintes et proposé par Gould, Sainvitu et Toint dans [12]. Des résultats numériques effectués sur un ensemble de problèmes de petite taille provenant de la collection `CUTEr` décrivent l'effet de l'utilisation de dérivées approximées sur la robustesse et l'efficacité de la méthode de filtre de type région de confiance.

Mots Clés. Optimisation sans contraintes, méthode de filtre, algorithme de type de région de confiance, dérivées approximées, résultats numériques.

Mathematics Subject Classification. 65K05, 90C26, 90C30, 90C53.

Received October 24, 2006. Accepted February 14, 2009.

[1] CENAERO, Eole Building, 29, Rue des Frères Wright, B-6041 Gosselies, Belgium;
`caroline.sainvitu@cenaero.be`

Article published by EDP Sciences

© EDP Sciences, ROADEF, SMAI 2009

1. Introduction

As most algorithms for nonlinear optimization, the filter-trust-region algorithm proposed by Gould *et al.* in [12] for solving unconstrained minimization problems requires knowledge of first and second derivatives of the objective function f. In the implementation used to perform numerical experiments of [12], the derivatives need to be calculated analytically and supplied by the user. So the question we are interested in here is: *is the behaviour of the filter-trust-region algorithm directly related to the use of exact derivatives?*

In some situations, the first and *a fortiori* the second derivatives may be unavailable. This may be due to the fact that the evaluation of these derivatives is very difficult, time-consuming or their calculation requires the solution of another problem. Notwithstanding, the unavailability of the derivatives does not necessarily imply that the objective function we consider is not differentiable. Actually, in the remainder of this study, we assume that the objective function f is indeed twice continuously differentiable. One way to fix the issue of unavailability of derivatives is to use finite-difference approximations to the gradient and/or the Hessian matrix. Another one, which is more widespread, is secant approximations to the second derivative matrices, like BFGS or SR1 updates.

The purpose of this paper is to answer the following question: *does the use of approximate derivatives damage the efficiency and the robustness of methods based on the filter mechanism?* This question will be analyzed by studying the performance of the algorithm over a set of test problems if we do not use exact derivatives. The first step in our investigation is to see where the first and second derivatives appear in the filter-trust-region algorithm. This will be done in the next section after the presentation of the filter algorithm. We can guess that the first derivative approximation is particularly tricky as the exact gradient is at the root of the filter method for solving unconstrained problems.

To perform this experimental study will consider two kinds of derivatives estimations: finite-difference techniques and secant approximations. Note that our aim here is only to see if the advantageous behaviour of filter methods described in many references and, in particular in [12], is directly dependent on the use of exact derivatives; this is why we will just consider problems of small size. The numerical study presented in this paper is intended to demonstrate the importance or not of the exact derivatives in the filter-based approach and not to provide a high performance solver in the case where derivatives are unavailable, which is beyond the scope of this work. Note that if one wants to treat large-scale optimization problems, it is preferable to use *limited memory BFGS update* (see [1] and [14]), this technique allows to considerably reduce the storage issue caused by the BFGS update.

We briefly describe the algorithm and the filter technique in Section 2. Section 3 will be devoted to the description of the different ways chosen to approximate the derivatives. We next present our numerical experiments in Section 4 and a brief conclusion is finally given in Section 5.

2. The problem and algorithm

We consider solving the nonlinear unconstrained optimization problem

$$\min_{x\in\mathbb{R}^n} \; f(x), \tag{1}$$

where f is a twice continuously differentiable function of the variables $x \in \mathbb{R}^n$. The idea of the algorithm is to combine a trust-region method (see [3]) with filter techniques (see [6]). The aim of the filter in this algorithm is to encourage convergence of iterates to first-order critical points by driving every component of the objective's gradient

$$\nabla_x f(x) \stackrel{\text{def}}{=} g(x) = (g_1(x), \ldots, g_n(x))^T \tag{2}$$

to zero. As in [7], this algorithm uses a multidimensional filter where each entry of it is a component of the gradient (see Fig. 1).

The philosophy of the method used to compute the trial point is to calculate a quadratic model m_k of the objective function in a *trust-region* centered at x_k,

$$\mathcal{B}_k = \{x_k + s \mid \|s\| \leq \Delta_k\}, \tag{3}$$

where $\|\cdot\|$ is the Euclidian norm on $\mathbb{R}^n$ and $\Delta_k > 0$ is the *trust-region radius*. A trial step is then computed by minimizing the model (possibly only approximately). Traditional trust-region algorithms then evaluate the objective function at the trial point $x_k^+ = x_k + s_k$ and, if the reduction achieved in the objective function is at least a fraction of that predicted by the model, the new trial point x_k^+ is accepted as the new iterate x_{k+1} and the trust-region radius is possibly enlarged. Otherwise, if the achieved reduction is too small, the trial point is rejected and the trust-region radius is reduced. In [12], Gould *et al.* have used a filter mechanism to potentially accept x_k^+ as the new iterate more often than in usual trust-region algorithms. Indeed, contrary to these methods, they do not require that the trial point is computed within the trust-region at every iteration. Some steps may not be restricted to the trust region. Recalling that the notion of filter is based on that of *dominance*, we decide that a trial point x_k^+ is *acceptable for the filter* $\mathcal{F}$ if and only if

$$\forall g_\ell \in \mathcal{F} \qquad \exists\, j \in \{1, \ldots, n\} : \quad |g_j(x_k^+)| < |g_{\ell,j}| - \gamma_g \|g_\ell\|, \tag{4}$$

where $\gamma_g \in (0, 1/\sqrt{n})$ is a small positive constant and where $g_{\ell,j} \stackrel{\text{def}}{=} g_j(x_\ell)$. So we can say that x_k^+ is not *dominated* by x_ℓ. Filter methods propose to accept a new iterate x_k^+ if it is not dominated by any other iterate in the filter. A filter with four points and its margin (dashed line), defined by the last term in equation (4), are represented in Figure 1 for a two dimensional setup, *i.e.*, $x \in \mathbb{R}^2$. All trial points belonging to the region above the dashed line are not acceptable for the filter. The margin is designed in order to avoid to accept a new point if it is arbitrarily close

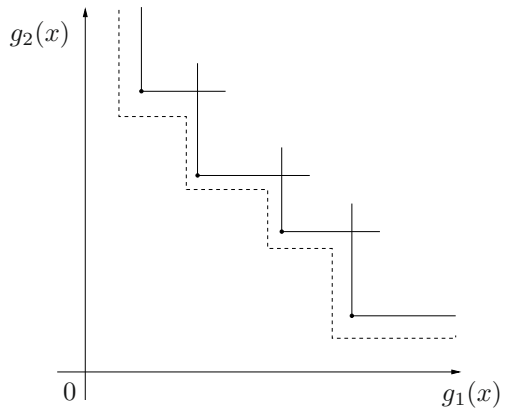

FIGURE 1. A filter for an unconstrained problem in $\mathbb{R}^2$.

to being dominated by any other point already in the filter. So the filter and the margin determine a forbidden region.

If an iterate x_k is acceptable in the sense of (4) (*i.e.* if this point is below the dashed line in Fig. 1), we may wish to add it to the *multidimensional filter* $\mathcal{F}$, which is a list of n-tuples of the form $(g_{k,1}, \ldots, g_{k,n})$, such that none of the corresponding iterates is dominated by any other. We also remove from the filter all other points that are dominated by the new entry. We refer the reader to [12] for further detail.

The above-described mechanism is adequate for convex problems, where a zero gradient is both necessary and sufficient for second-order criticality. However, it may be unsuitable for nonconvex ones. Indeed it might prevent progress away from a saddle point, in which case an increase in the gradient components is acceptable. Therefore, in [12], we have modified the filter mechanism to ensure that the filter is reset to the empty set after each iteration giving sufficient descent on the objective function at which the model m_k was detected to be nonconvex, and set an upper bound on the acceptable objective function values to ensure that the obtained decrease is permanent (see Algorithm 2.1).

In short, the philosophy of the filter-trust-region algorithm is to let the filter play the main role in ensuring global convergence within "convex basins", falling back on the usual trust-region method only if things do not go well or if negative curvature is encountered. More formally, the algorithm is defined as follows:

Algorithm 2.1. ***Filter-Trust-Region Algorithm***

Step 0. Initialization An initial point x_0 and an initial trust-region radius $\Delta_0 > 0$ are given. The constants $\gamma_g \in (0, 1/\sqrt{n})$, η_1, η_2, γ_1, γ_2 and γ_3 are also given and satisfy

$$0 < \eta_1 \leq \eta_2 < 1 \quad \text{and} \quad 0 < \gamma_1 \leq \gamma_2 < 1 \leq \gamma_3. \tag{5}$$

Compute $f(x_0)$ and $g(x_0)$, set $k = 0$. Initialize the filter $\mathcal{F}$ to the empty set and choose $f_{\text{sup}} \geq f(x_0)$. Define two flags `RESTRICT` and `NONCONVEX`, the former to be unset.

Step 1. Determine a trial step
Compute a finite step s_k that "sufficiently reduces" the model m_k and that also satisfies $\|s_k\| \leq \Delta_k$ if `RESTRICT` is set or if m_k is nonconvex. In the latter case, set `NONCONVEX`; otherwise unset it. Compute the trial point $x_k^+ = x_k + s_k$.

Step 2. Compute $f(x_k^+)$ and define the following ratio

$$\rho_k = \frac{f(x_k) - f(x_k^+)}{m_k(x_k) - m_k(x_k^+)}. \tag{6}$$

If $f(x_k^+) > f_{\text{sup}}$, set $x_{k+1} = x_k$, set `RESTRICT` and go to Step 4.

Step 3. Test to accept the trial step

- Compute $g_k^+ = g(x_k^+)$.
- If x_k^+ is acceptable for the filter $\mathcal{F}$ and `NONCONVEX` is unset:
  Set $x_{k+1} = x_k^+$, unset `RESTRICT` and add g_k^+ to the filter $\mathcal{F}$ if either $\rho_k < \eta_1$ or $\|s_k\| > \Delta_k$.
- If x_k^+ is not acceptable for the filter $\mathcal{F}$ or `NONCONVEX` is set:
  If $\rho_k \geq \eta_1$ and $\|s_k\| \leq \Delta_k$, then
  set $x_{k+1} = x_k^+$, unset `RESTRICT` and if `NONCONVEX` is set, set $f_{\text{sup}} = f(x_{k+1})$ and reinitialize the filter $\mathcal{F}$ to the empty set;
  else set $x_{k+1} = x_k$ and set `RESTRICT`.

Step 4. Update the trust-region radius
If $\|s_k\| \leq \Delta_k$, update the trust-region radius by choosing

$$\Delta_{k+1} \in \begin{cases} [\gamma_1 \Delta_k, \gamma_2 \Delta_k] & \text{if} \quad \rho_k < \eta_1, \\ [\gamma_2 \Delta_k, \Delta_k] & \text{if} \quad \rho_k \in [\eta_1, \eta_2), \\ [\Delta_k, \gamma_3 \Delta_k] & \text{if} \quad \rho_k \geq \eta_2; \end{cases} \tag{7}$$

otherwise, set $\Delta_{k+1} = \Delta_k$. Increment k by one and go to Step 1.

As said in the introduction, our purpose is to study the influence of using approximate derivatives on this filter algorithm. We start by identifying where these derivatives appear in the above-described algorithm. They are present in different positions of Algorithm 2.1. All derivatives are obviously taken into account in the definition of the quadratic model of the objective function. We can guess that the approximation to the gradient will be a critical issue because the gradient is precisely the quantity we are trying to make zero. So, as in classical minimization

algorithms, the gradient of the objective function must be known accurately both for computing the next trial point and for the stopping criteria. The gradient has also a major role because it determines the filter. Indeed, the first derivatives appear in particular in the definition of the filter and thus in the filter test acceptance mechanism. The second derivatives notably influence the fact that the algorithm chooses to use the filter technique or not. When negative curvature is detected, we just fall back to a classical trust-region method and then the steps are restricted to the trust-region. Therefore, if the second-order information is not exact and the algorithm detects erroneous negative curvature, the convergence may be slowed down by smaller steps.

3. Approximating the derivatives

In this section, we will describe some different ways to approximate the gradient $\nabla_x f(x)$ and the Hessian $\nabla_{xx} f(x)$ of the objective function.

3.1. finite differences

A common way to approximate first and second derivatives is to use finite-difference approximations. The technique of finite differencing is inspired by Taylor's theorem. In fact, the derivatives are a measure of the sensitivity of the function to infinitesimal changes in the values of the variables.

Finite-difference gradients. We have chosen two formulae for approximating the first derivative $\nabla_x f(x)$ by finite differences. We may define the *forward finite-difference approximation* to the gradient componentwise by

$$(\nabla_x f(x))_j \approx \frac{f(x + h_j e_j) - f(x)}{h_j}, \quad j = 1, \ldots, n, \tag{8}$$

where $h \in \mathbb{R}^n$ is a vector of stepsizes and e_j is the jth unit vector. A more accurate approximation to the gradient can be obtained by using the *central finite-difference approximation* given by

$$(\nabla_x f(x))_j \approx \frac{f(x + h_j e_j) - f(x - h_j e_j)}{2h_j}, \quad j = 1, \ldots, n. \tag{9}$$

An important issue in the implementation of these formulae is the choice of the stepsize h. We have tested different stepsizes that are discussed in the section devoted to the numerical experiments. The forward approximation requires n additional function evaluations while the central one requires $2n$.

Finite-difference Hessians. Finite-difference approximations are also possible for second derivative matrices. If the gradient of the objective function is analytically available, the Hessian matrix can be obtained by applying the *forward*

finite-difference formula

$$(\nabla_{xx} f(x))_{.j} \approx B_{.j} = \frac{\nabla_x f(x + h_j e_j) - \nabla_x f(x)}{h_j}, \quad j = 1, \ldots, n, \tag{10}$$

or the *central finite-difference* one

$$(\nabla_{xx} f(x))_{.j} \approx B_{.j} = \frac{\nabla_x f(x + h_j e_j) - \nabla_x f(x - h_j e_j)}{2h_j}, \quad j = 1, \ldots, n. \tag{11}$$

Remark that this column-at-a-time process does not necessarily lead to a symmetric matrix; nevertheless, we can restore the symmetry by applying $B = (B+B^T)/2$. We have also tested different stepsizes that we will discuss later. The forward formula requires n additional gradient calls while, for the central one, $2n$ additional gradient evaluations are needed.

For the case in which even first derivatives are not available, we can use the following finite-difference formula that uses only function values

$$\begin{aligned} (\nabla_{xx} f(x))_{ij} &\approx B_{ij} \\ &= \frac{f(x + h_i e_i + h_j e_j) - f(x + h_i e_i) - f(x + h_j e_j) + f(x)}{h_i h_j}, \\ &\quad 1 \le i \le j \le n. \end{aligned} \tag{12}$$

Taking symmetry into account, this approximation requires $\frac{1}{2}(n^2 + 3n)$ additional function evaluations to that of $f(x)$. Obviously, this is relatively expensive and therefore, in practice, this strategy is used only if the cost of a function evaluation is not too high. So, in the situation where the first and *a fortiori* the second derivatives of the function involved in the problem (1) are not available or are time-consuming, we often prefer to consider *derivative-free optimization* techniques (see [4]).

3.2. Secant approximations

We now focus on two popular secant update formulae for second derivative matrices, namely the BFGS and the SR1 updates. The basic idea of these techniques is to update approximations to the Hessian matrices in some computational cheap ways. A major advantage of these methods is, of course, that they do not require computation of second derivatives, neither additional function or gradient evaluations.

Broyden-Fletcher-Goldfarb-Shanno. The BFGS formula is the most widespread secant approximation. The Hessian update can be calculated by

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}, \tag{13}$$

where $s_k = x_{k+1} - x_k$ and $y_k = \nabla_x f(x_{k+1}) - \nabla_x f(x_k)$. Note that if B_k is positive definite then the update B_{k+1} will be also positive definite (when $y_k^T s_k > 0$). In some situations, the updating formula can produce bad results. For example, when $y_k^T s_k$ is negative or too close to zero, we can however simply skip the Hessian update and set $B_{k+1} = B_k$. We also skip the update if y_k is sufficiently close to $B_k s_k$.

We must also discuss the choice of the initial Hessian approximation B_0. There are no magic formulae to determine it. It is common to start secant update with B_0 set to the identity matrix. We have tested other choices :

- a multiple of the identity matrix, for example, $B_0 = |f(x_0)|\, I$;
- B_0 can be rescaled before B_1 is computed as $B_0 = \frac{y_0^T s_0}{s_0^T B_0 s_0} I$ (for more detail see [15]);
- some finite-difference approximation at x_0.

Symmetric rank-one. Contrary to the BFGS technique, which is a rank-two updating formula, the SR1 is a *symmetric-rank-one* update; it does not guarantee to produce a positive definite matrix. However, the SR1 updating formula combined with a trust-region method has proved to be quite useful (see [2]). The SR1 update of the Hessian matrix is given by

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}. \tag{14}$$

One of the drawbacks of this method is that the denominator $(y_k - B_k s_k)^T s_k$ can vanish or become very small; so, in this case, we skip the update and set $B_{k+1} = B_k$. The same choices for the initial Hessian approximation than for the BFGS update can be considered.

4. Numerical results

In this section, we analyze the influence of substituting finite-difference approximations or secant ones for the analytic derivatives on the robustness and the efficiency of the filter-trust-region algorithm described in Section 2. We want to see if the advantageous behaviour of filter methods is directly dependent on the use of exact derivatives. The numerical results are examined by means of *performance profiles* proposed by Dolan and Moré in [5]. Performance profiles give, for every $\sigma \geq 1$, the proportion $p(\sigma)$ of test problems on which each considered algorithmic variant has a performance within a factor σ of the best.

We now discuss the framework in which our numerical experiments are executed. Obviously, several of our choices are not the only ones possible or even the only ones used and therefore, our numerical investigation is not perfect. Our results are obtained by running the algorithm on 66 unconstrained small-scaled problems from the CUTEr collection (see [10]). The names of the problems with their dimensions (i.e. the number of free variables) are detailed in Table 1.

TABLE 1. The test problems and their dimension.

Problem	n	Problem	n	Problem	n
AIRCRFTB	5	EXPFIT	2	MEYER3	3
ALLINITU	4	GROWTHLS	3	OSBORNEA	5
BARD	3	GULF	3	OSBORNEB	11
BEALE	2	HAIRY	2	PALMER1C	8
BIGGS3	3	HATFLDD	3	PALMER1D	7
BIGGS5	5	HATFLDE	3	PALMER2C	8
BIGGS6	6	HEART6LS	6	PALMER3C	8
BOX2	2	HEART8LS	8	PALMER4C	8
BOX3	3	HELIX	3	PALMER5C	6
BRKMCC	2	HIELOW	3	PALMER6C	8
BROWNBS	2	HILBERTA	2	PALMER7C	8
BROWNDEN	4	HILBERTB	10	PALMER8C	8
CLIFF	2	HIMMELBB	2	ROSENBR	2
CUBE	2	HIMMELBF	4	S308	2
DENSCHNA	2	HIMMELBG	2	SINEVAL	2
DENSCHNB	2	HIMMELBH	2	SISSER	2
DENSCHNC	2	HUMPS	2	SNAIL	2
DENSCHND	3	JENSMP	2	STRATEC	10
DENSCHNE	3	KOWOSB	4	VIBRBEAM	8
DENSCHNF	2	LOGHAIRY	2	WATSON	12
DJTL	2	MARATOSB	2	YFITU	3
ENGVAL2	2	MEXHAT	2	ZANGWIL2	2

In each case, the starting point supplied with the problem was used. All tests were performed in double precision on a workstation with a 3.2 GHz Pentium IV biprocessor and 2GB of memory under SUSE Professional 9.0 Linux and the Lahey Fortran compiler (version L6.10a) with default options.

Note that the algorithm as described in Section 2 lacks a formal stopping criterion. In practice, the algorithm stops if

$$\|g_k\| \leq 10^{-6}\sqrt{n}, \tag{15}$$

where g_k stands for $\nabla_x f(x_k)$ or an approximation to this gradient, and the flag `NONCONVEX` is unset. All attempts to solve the test problems were also limited to a maximum of 1000 iterations or 1 hour of CPU time. As in [12], the variability of CPU times for small times is taken into account by repeatedly solving the same problem until a threshold of ten seconds is exceeded and then taking the average per run. We have used the values $\gamma_1 = 0.625$, $\gamma_2 = 0.25$, $\gamma_3 = 2$,$\eta_1 = 0.01$, $\eta_2 = 0.9$, $\Delta_0 = 1$ and

$$\gamma_g = \min\left[0.001, \frac{1}{2\sqrt{n}}\right], \tag{16}$$

where the constant γ_g appears in (4). We have tested two particular variants. The first, called *filter*, is the algorithm as described in Section 2, where, at each iteration, the trial point is computed by approximately minimizing the model $m_k(x_k + s)$ using the Generalized Lanczos Trust-Region algorithm of [8] (without preconditioning) as implemented in the GLTR module of the GALAHAD library (see [11]). This procedure is terminated at the first step s for which

$$\|\nabla_x m_k(x_k + s)\| \leq \min\left[0.1, \sqrt{\max(\epsilon_M, \|\nabla_x m_k(x_k)\|)}\right] \|\nabla_x m_k(x_k)\|, \quad (17)$$

where ϵ_M is the machine precision. In addition, we choose

$$f_{\text{sup}} = \min(10^6 |f(x_0)|, f(x_0) + 1000) \quad (18)$$

at Step 0 of the algorithm. Based on practical experience [13], we also impose that $\|s_k\| \leq 1000\Delta_k$ at all iterations following the first one at which a restricted step was taken.

The second algorithmic variant is the *trust-region* (TR) method, that is the same algorithm except that no trial point is ever accepted in the filter; therefore trial points are always restricted to the trust-region.

Two strategies are used for the approximation to the derivatives: finite-difference techniques and secant approximations. We also include a comparison with a perturbation of the exact Hessian matrix. Incorporating our two algorithmic variants in the different methods used to approximate the derivatives, we obtain more than 70 tested variants. For completeness, the behaviour of the two variants using the exact first and second derivatives is also shown.

4.1. Finite-difference derivatives

Firstly, we consider the effect of substituting finite-difference approximations for the analytic derivatives on the algorithm. We also analyze the practical choice of stepsizes in computing these approximations.

We first discuss the case where the exact first derivatives are available. In Section 3, we have seen two formulae to approximate the Hessian matrices, the forward (10) and the central one (11). We now have to choose the finite-difference stepsize in practice. For this selection, we have to make a compromise between large rounding errors (small stepsize) and large approximation errors (large stepsize). We have tested different choices from the literature which are given in Table 2.

We obtain the best result with the third stepsize for the forward formula and with the first stepsize for the central one; therefore we present our results with these choices. Figures 2 and 3 show the efficiency in terms of number of iterations and CPU time for the filter variant and the trust-region one, both with exact second derivatives and forward and central finite-difference approximations to them.

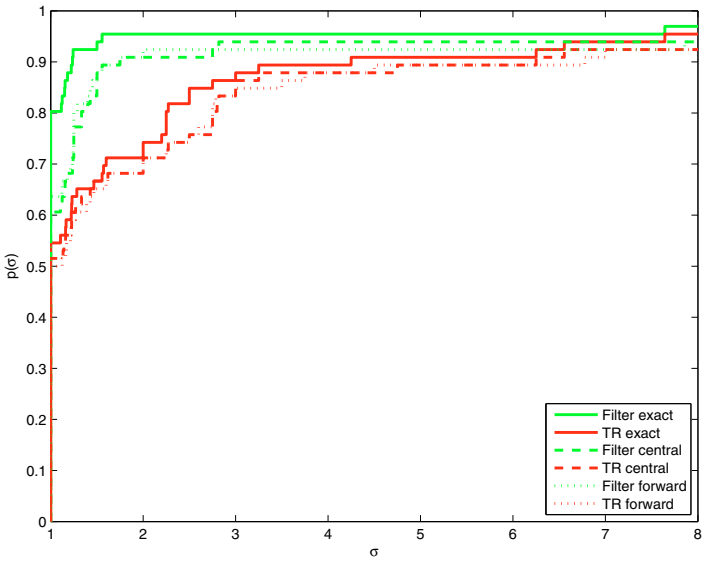

FIGURE 2. Iteration performance profile for the two variants with exact derivatives and approximate second derivatives by finite differences when the gradient is available.

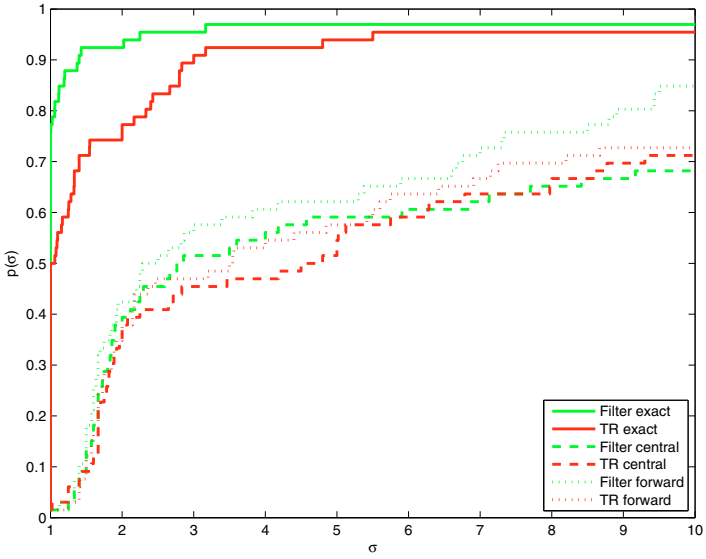

FIGURE 3. CPU time performance profile for the two variants with exact derivatives and approximate second derivatives by finite differences when the gradient is available.

TABLE 2. The different stepsizes.

	Forward	Central
Stepsize 1	$\sqrt{\epsilon_M}$	$\sqrt[3]{\epsilon_M}$
Stepsize 2	$\text{sign}\,(x_j)\sqrt{\epsilon_M}\max(\lvert x_j\rvert, 1)$	$\sqrt[3]{\epsilon_M}(1+\lvert x_j\rvert)$
Stepsize 3	$\sqrt{\epsilon_M}\max(\lvert x_j\rvert, 1)$	$\sqrt[3]{\epsilon_M}\max(\lvert x_j\rvert, 1)$
Stepsize 4	$\sqrt{\epsilon_M}(1+\lvert x_j\rvert)$	$\text{sign}\,(x_j)\sqrt[3]{\epsilon_M}\max(\lvert x_j\rvert, 1)$
Stepsize 5	-	$10^{-4}(1+\lvert x_j\rvert)$

TABLE 3. The different stepsizes.

		Stepsize for $\nabla_x f$	Stepsize for $\nabla_{xx} f$
Forward	Stepsize 1	$\sqrt{\epsilon_M}$	$\sqrt[4]{\epsilon_M}$
	Stepsize 2	$\sqrt{\epsilon_M}$	$\text{sign}\,(x_j)\sqrt[3]{\epsilon_M}\max(\lvert x_j\rvert, 1)$
	Stepsize 3	$\sqrt{\epsilon_M}$	$\text{sign}\,(x_j)\sqrt[4]{\epsilon_M}\max(\lvert x_j\rvert, 1)$
Central	Stepsize 1	$\sqrt[3]{\epsilon_M}$	$\sqrt[4]{\epsilon_M}$
	Stepsize 2	$\sqrt[3]{\epsilon_M}$	$\text{sign}\,(x_j)\sqrt[4]{\epsilon_M}\max(\lvert x_j\rvert, 1)$
	Stepsize 3	$\text{sign}\,(x_j)\sqrt[3]{\epsilon_M}\max(\lvert x_j\rvert, 1)$	$\text{sign}\,(x_j)\sqrt[3]{\epsilon_M}\max(\lvert x_j\rvert, 1)$
	Stepsize 4	$\text{sign}\,(x_j)\sqrt[3]{\epsilon_M}\max(\lvert x_j\rvert, 1)$	$\text{sign}\,(x_j)\sqrt[4]{\epsilon_M}\max(\lvert x_j\rvert, 1)$
	Stepsize 5	$10^{-4}(1+\lvert x_j\rvert)$	$10^{-4}(1+\lvert x_j\rvert)$

Whereas the filter variants using approximate Hessian matrices are even better in terms of number of iterations than the trust-region method with exact derivatives, we obviously can observe in Figure 3 that the finite-difference versions are much more computationally expensive than the exact derivatives variants. As predicted in Section 3, the finite-difference techniques require additional gradient evaluations and are thus more costly. Moreover, some small negative curvature in the Hessian matrix may remain undetected when it is approximated by finite-differences formulae. However, we can point out the fact that the filter algorithmic variant with approximate second derivatives by finite-difference techniques is significantly better as well in terms of iteration count as in terms of CPU time than the corresponding trust-region variant. The use of approximate Hessians by finite differences within a filter-trust-region framework does not seem to have more influence than such an approximation in the more classical trust-region scheme. The two plots in Figure 4, showing the iteration and CPU time performance profiles for the two variants with the Hessian approximated by central finite differences, clearly illustrate this fact.

We now examine the case where even the gradient of the objective function is not available; so both first and second derivatives are to be approximated by finite-difference formulae. The Hessian matrices are approximated by (12) and the gradient either by the forward formula (8) or by the central one (9). The tested stepsizes are stated in Table 3.

The stepsizes that give the best performances on our test problem set are the third one when the first derivatives are approximated by forward finite differences

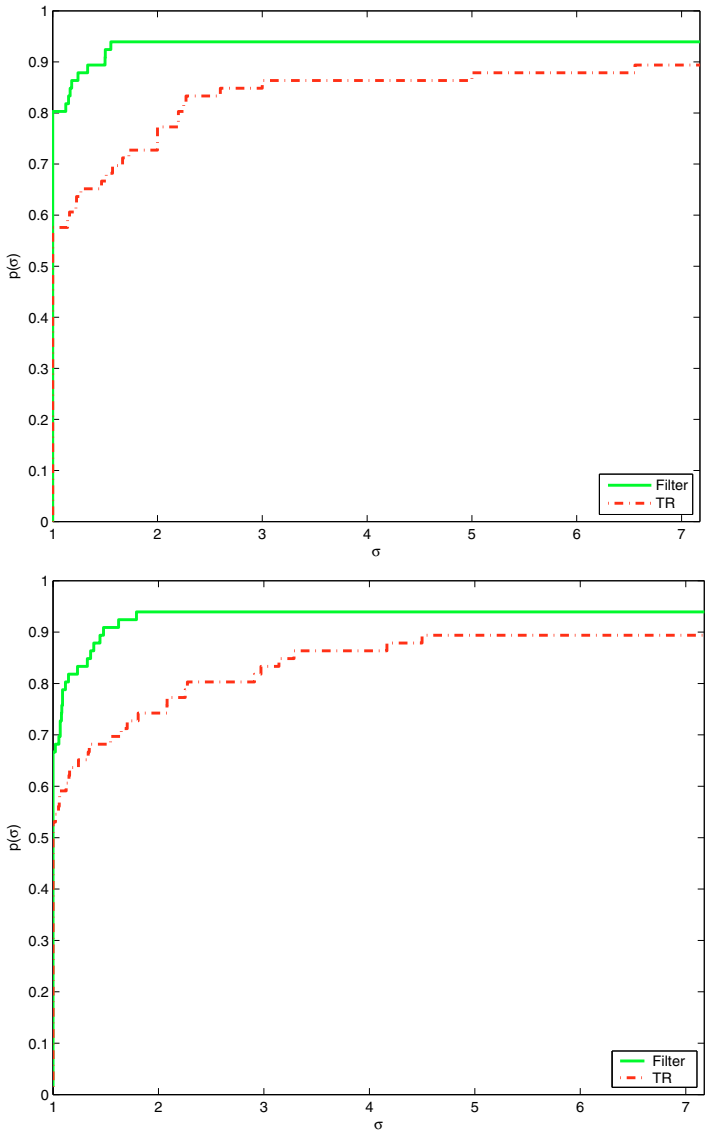

FIGURE 4. Iteration and CPU time performance profiles for the two variants with approximate second derivatives by central finite differences when the gradient is available.

and the second one when they are estimated by the central finite differences. We can see these results in Figures 5 and 6.

As expected in Section 3, the variants with both derivatives approximated by finite differences using only function values are computationally very expensive and

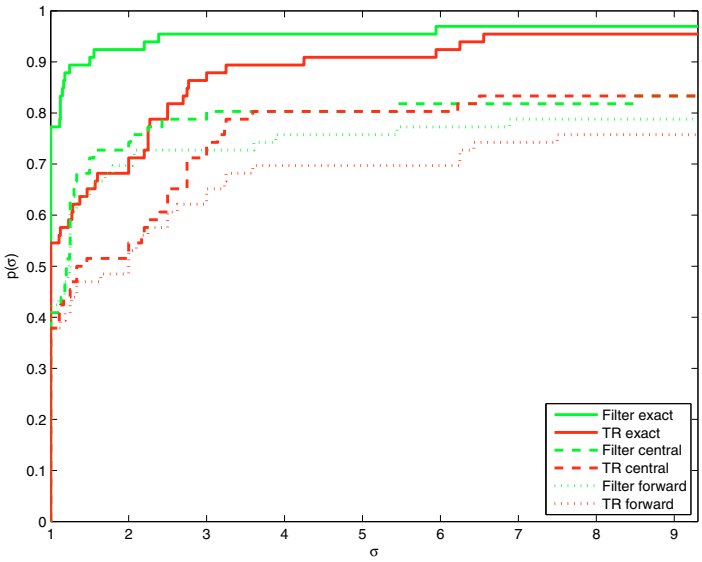

FIGURE 5. Iteration performance profile for the two variants with exact derivatives and approximate first and second derivatives by finite differences.

are less efficient and robust than those where the gradient is available. This CPU time penalty is, for a part, due to the fact that more iterations are now needed but also to the fact that lots of function evaluations are required to approximate the gradient and the Hessian. As the first derivative is exactly the criticality measure we are trying to drive to zero, this proves its importance in the filter algorithm as in other algorithms for unconstrained minimization. However, we can point out the fact that, even with these approximations to both first and second derivatives, each filter variant is significantly better than its corresponding trust-region variant.

4.2. SECANT APPROXIMATIONS

We now present our numerical experiments with the secant approximations to the Hessian matrix: the BFGS and the SR1 update formulae. Note that in these results the exact gradient is used. We have already discussed in Section 3 different choices for the initial matrix, B_0. However, for the BFGS update, it is not convenient to set B_0 to a finite-difference approximation to $\nabla^2_{xx} f(x_0)$. The reason being that secant methods based on this updating formula must be initialized with and must maintain a positive definite Hessian approximation. But with an initial approximation matrix computed by finite differences, there is no guarantee that B_0 will be positive definite. So, if one wants to use this initial approximation, the approximate matrix must be perturbed into a positive definite one. However, it is interesting to note that, in practice, the filter-trust-region algorithm using BFGS updates behaves very well with an initial finite-difference approximation without

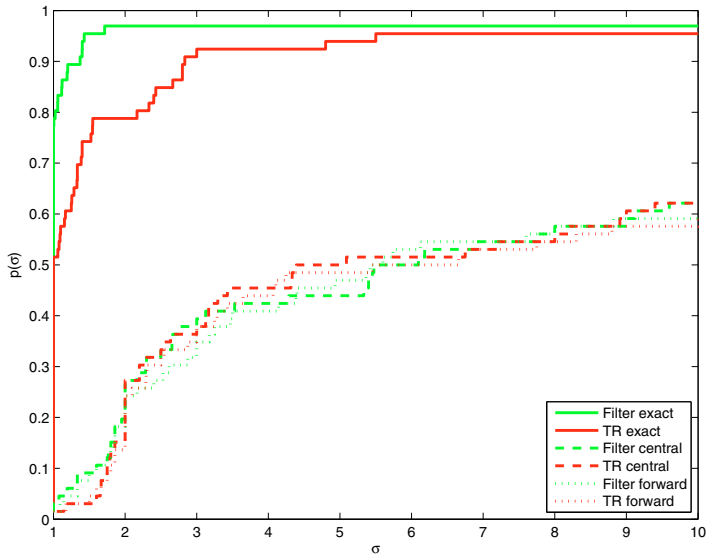

FIGURE 6. CPU time performance profile for the two variants with exact derivatives and approximate first and second derivatives by finite differences.

correction. But applying BFGS techniques with an indefinite starting matrix is not standard in the literature, although the use of a trust region allows for this. Therefore the results presented below are obtained by setting $B_0 = I$. The left plot in Figure 7 shows the results in terms of number of iterations and the right one displays the CPU time. We have excluded problems where variants do not report the same final objective function value[1].

Both BFGS versions are obviously less efficient and robust than the exact ones. In terms of iteration count and CPU time, the BFGS filter and BFGS pure trust-region variants are comparable, indicating that using a BFGS approximation to the Hessian matrix in the filter-trust-region algorithm implies a larger increase in the number of iterations (and thus in the computation time) than using the same approximation within a classical trust-region scheme. However, it can be remarked that the BFGS variants are more competitive in terms of CPU time efficiency with the exact ones than the finite-difference variants were. It is important to mention that, since the BFGS update only produces positive definite matrices, the flag `NONCONVEX` in Algorithm 2.1 is never set because negative curvature is never detected while solving the trust-region subproblem. This simplifies our test acceptance mechanism in Step 3 of the algorithm, and, furthermore, the filter is never reinitialized to the empty set. We think that, as the algorithm directly relies on detecting negative curvature, it is more appropriate to use an updating formula

[1]Problems `BIGGS6`, `GROWTHLS`, `GULF` and `JENSMP` have been excluded.

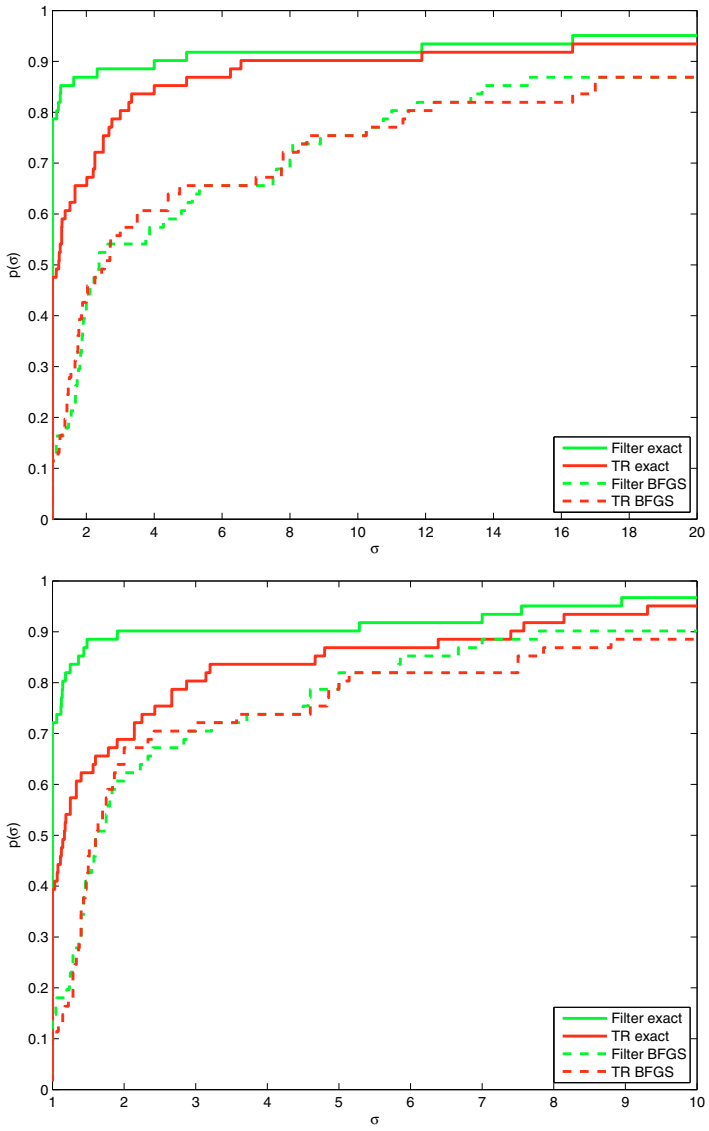

FIGURE 7. Iteration and CPU time performance profile for the two variants with exact derivatives and BFGS approximate second derivatives.

which allows to generate indefinite Hessian approximations, like the symmetric-rank-one update. This should indicate that the ability of using nonconvex models for the computation of the trial step might be important in the filter-trust-region algorithm.

We thus consider the SR1 update. Note that the SR1 Hessian approximation is not restricted to be positive definite. Therefore we can now estimate the initial approximation B_0 by a finite-difference technique as this matrix does not need to be positive definite. Our guess is that, in the filter-trust-region algorithm, the SR1 approximation may prove more accurate than BFGS since if negative curvature is present, it may reflected in the model. The best results are obtained with a finite-difference estimation of the Hessian matrix at x_0 and these results are displayed in Figure 8.

It can be seen in these plots that the filter variant with SR1 updating formula is more efficient both in terms of iterations and CPU time than the corresponding pure trust-region variant, but the latter is a little more reliable.

4.3. Comparison

We now present a brief comparison of the different algorithmic variants. As the number of curves would be very high in a performance profile, we prefer to display our comparison by using a combined performance plot (see [9]), where each point consists of the average iteration count and the average CPU time of each tested variant. Note that we only consider problems for which all variants were successful (failure is declared if the maximum number of iterations has been reached or if the step is too short to allow further progress). Therefore, Figure 9 does not give indication about the robustness of the different variants. We only present results for the filter variant.

We have also analyzed the effect of perturbing the exact Hessian matrices. We consider two types of perturbation: either we perturb all elements of the matrix by a small constant or we only perturb the diagonal elements by this same quantity. We have tried with 10^{-4} and 10^{-6} as disturbance values. We can clearly see in Figure 9 that the perturbed variants are very close to the exact variant. There are two main "clusters". The first one contains the finite-difference variants (either with first derivatives available or not); these techniques are relatively close in terms of number of iterations to the exact variant but are much more computationally expensive. The second cluster includes the secant approximation variants; these require more iterations but are more comparable in terms of CPU time to the exact version. As we have excluded problems for which at least one variant has failed, the plot in Figure 9 does not give overall information about performance of the variants. For example, BFGS updates are not always better than SR1 ones in terms of iteration count as one may think when observing Figure 9.

We now present more accurate results for the problems `HIELOW` and `STRATEC`, which have respectively 3 and 10 variables. As previously, we only consider the filter variant. As expected by the literature, the Quasi-Newton techniques work well for these nested logit models. It can be noticed in plots of Figures 10 that using a secant approximation to the second derivatives implies an important increase of the number of iterations; indeed, this number is nearly doubled for the BFGS and SR1 variants. However, computational time per iteration is reduced for these variants, while this measure of time is much higher for the finite-difference versions.

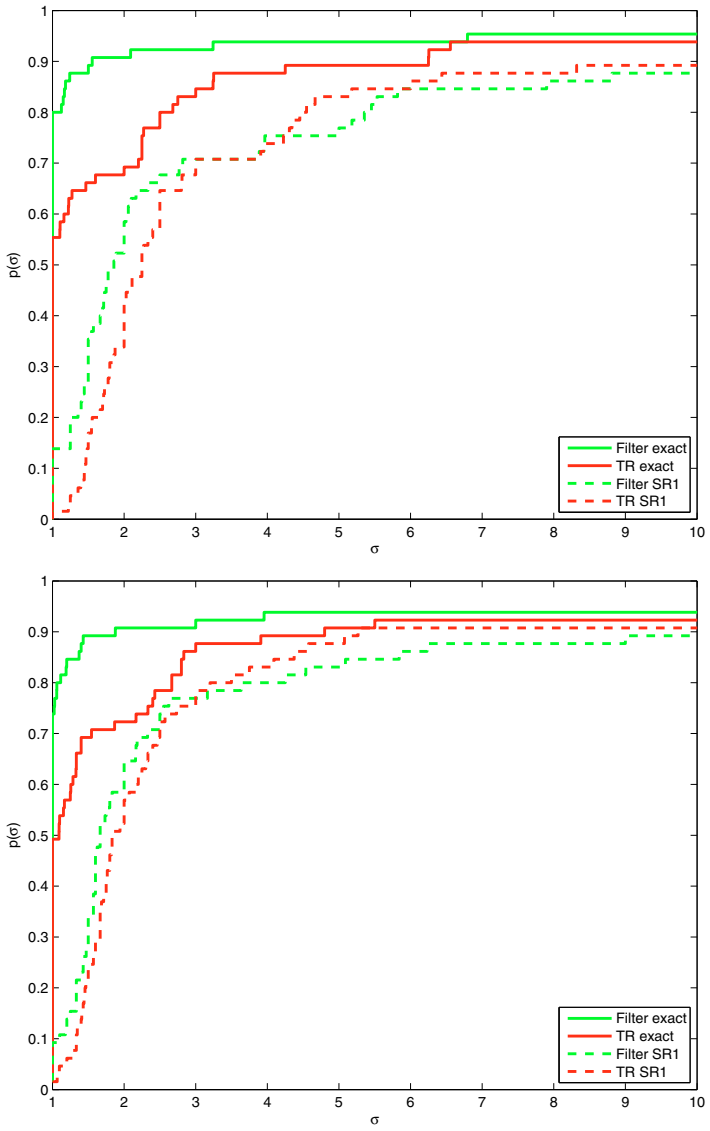

FIGURE 8. Iteration and CPU time performance profile for the two variants with exact derivatives and SR1 approximate second derivatives.

So we can guess that, for these problems, the evaluation of the gradient is very expensive. For the problem `STRATEC`, the variant which requires the most iterations is the finite-difference approximation in objective function values, this is when the

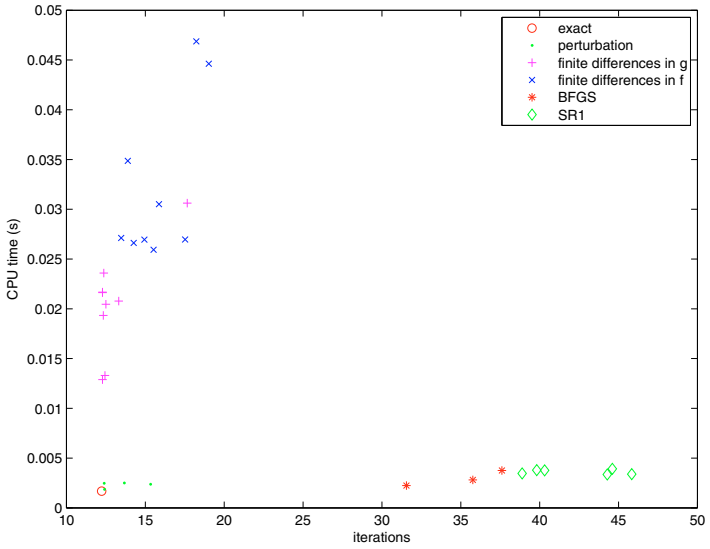

FIGURE 9. Combined performance of all tested variants (only with the filter).

gradient and the Hessian are approximated by using only function evaluations. But this algorithmic variant is better in terms of CPU time than the two other finite-difference estimations in gradient values, certainly because the evaluation of the function is cheaper than the evaluation of the gradient. Globally, we obtain a significant gain by using Hessian approximated by secant updates when the evaluations of the gradient and/or the objective function are too expensive. Even if they require more than 20 additional iterations for the problem `STRATEC`, the variants using the secant update are still competitive from a computational time point of view.

5. CONCLUSION

We have presented a practical study of the influence of approximate derivatives on the filter-trust-region algorithm designed for unconstrained optimization. In view of the numerical experiments, we can say that, generally, the filter-trust-region algorithm does not suffer more than the classical trust-region method from the use of approximate derivatives.

Acknowledgements. The author wishes to thank Philippe L. Toint for a number of useful comments and suggestions. The author is also indebted to two anonymous referees for their constructive comments and corrections.

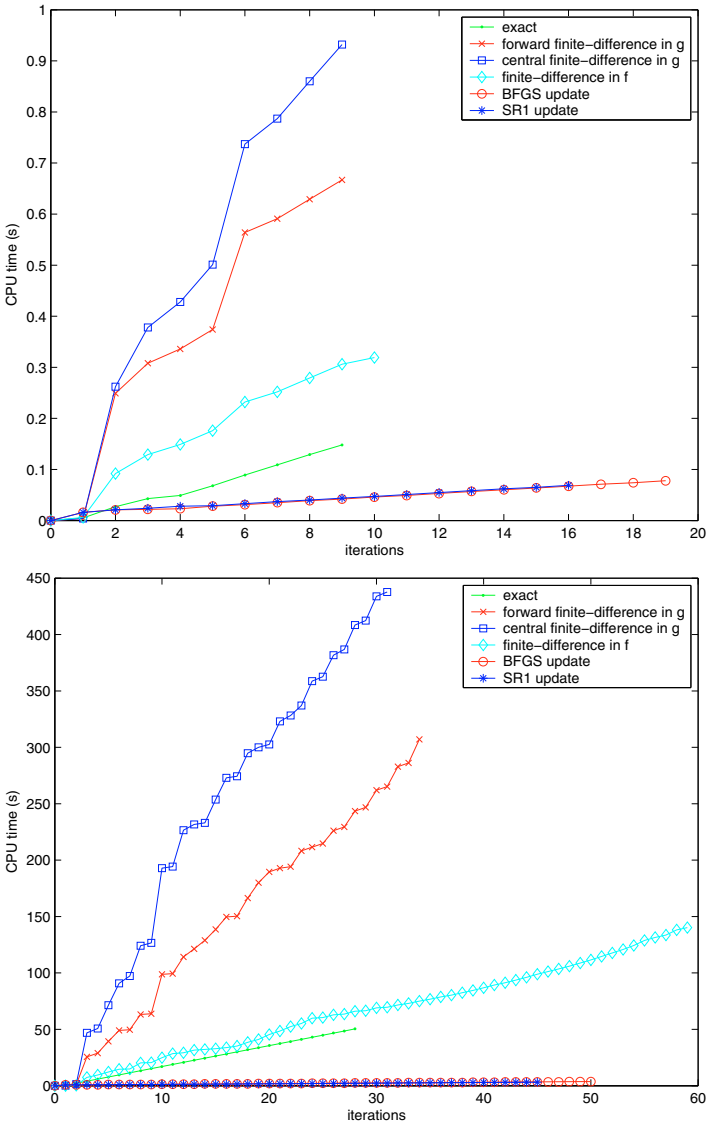

FIGURE 10. Comparison of computational time between exact and approximate filter variants for the `HIELOW` and `STRATEC` problems, respectively.

REFERENCES

[1] R.H. Byrd, P. Lu, J. Nocedal and C. Zhu, A limited memory algorithm for bound constrained optimization. *SIAM J. Sci. Comput.* **16** (1995) 1190–1208.

[2] A.R. Conn, N.I.M. Gould and P.L. Toint, Convergence of quasi-Newton matrices generated by the Symmetric Rank One update. *Math. Program.* **50** (1991) 177–196.
[3] A.R. Conn, N.I.M. Gould and P.L. Toint, *Trust-Region Methods*. MPS-SIAM Series on Optimization 1, SIAM, Philadelphia, USA (2000).
[4] A.R. Conn, K. Scheinberg and P.L. Toint, Recent progress in unconstrained nonlinear optimization without derivatives. *Math. Program. Ser. B* **79** (1997) 397–414.
[5] E.D. Dolan and J.J. Moré, Benchmarking optimization software with performance profiles. *Math. Program.* **91** (2002) 201–213.
[6] R. Fletcher and S. Leyffer, Nonlinear programming without a penalty function. *Math. Program.* **91** (2002) 239–269.
[7] N.I.M. Gould, S. Leyffer and P.L. Toint, A multidimensional filter algorithm for nonlinear equations and nonlinear least-squares. *SIAM J. Optim.* **15** (2005) 17–38.
[8] N.I.M. Gould, S. Lucidi, M. Roma and P.L. Toint, Solving the trust-region subproblem using the Lanczos method. *SIAM J. Optim.* **9** (1999) 504–525.
[9] N.I.M. Gould, D. Orban, A. Sartenaer and P.L. Toint, Sensitivity of trust-region algorithms on their parameters. *4OR, Quarterly Journal of Operations Research* **3** (2005) 227–241.
[10] N.I.M. Gould, D. Orban and P.L. Toint, CUTEr, a constrained and unconstrained testing environment, revisited *ACM Trans. Math. Software* **29** (2003) 373–394.
[11] N.I.M. Gould, D. Orban and P.L. Toint, GALAHAD — a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Trans. Math. Software* **29** (2003) 353–372.
[12] N.I.M. Gould, C. Sainvitu and P.L. Toint, A Filter-Trust-Region Method for Unconstrained Optimization. *SIAM J. Optim.* **16** (2005) 341–357.
[13] N.I.M. Gould and P.L. Toint, FILTRANE, *a Fortran* 95 *Filter-Trust-Region Package for Solving Systems of Nonlinear Equalities, Nonlinear Inequalities and Nonlinear Least-Squares Problems*. Technical report 03/15, Rutherford Appleton Laboratory, Chilton, Oxfordshire, UK (2003).
[14] D.C. Liu and J. Nocedal, On the limited memory BFGS method for large scale optimization. *Math. Program. Ser. B* **45** (1989) 503–528.
[15] D.F. Shanno and K.H. Phua, Matrix conditionning and nonlinear optimization. *Math. Program.* **14** (1978) 149–160.