Math. Program., Ser. A 91: 239–269 (2002)

Digital Object Identifier (DOI) 10.1007/s101070100244

Roger Fletcher · Sven Leyffer

Nonlinear programming without a penalty function

Received: October 17, 1997 / Accepted: August 17, 2000
Published online September 3, 2001 –

Abstract. In this paper the solution of nonlinear programming problems by a Sequential Quadratic Programming (SQP) trust-region algorithm is considered. The aim of the present work is to promote global convergence without the need to use a penalty function. Instead, a new concept of a "filter" is introduced which allows a step to be accepted if it reduces *either* the objective function *or* the constraint violation function. Numerical tests on a wide range of test problems are very encouraging and the new algorithm compares favourably with LANCELOT and an implementation of $S l_1$QP.

Key words. nonlinear programming – SQP – filter – penalty function

1. Introduction

This paper is concerned with the development of an algorithm for finding a local solution of a Nonlinear Programming (NLP) problem. For ease of explanation, we assume that the problem is of the form

$$(P)\quad \begin{cases} \underset{\mathbf{x}}{\text{minimize}} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{c}(\mathbf{x}) \leq \mathbf{0}, \end{cases}$$

although we evolve to a more general formulation later in the paper. In problem (P) we assume that $f : \mathbb{R}^n \to \mathbb{R}$ and $\mathbf{c} : \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable, and that expressions for first and second derivatives are available, although much of what we have to say is applicable to algorithms that approximate second derivatives in some way. However, given the ready availability of effective software for automatic differentiation, we currently regard the exact second derivatives format as being the most important for use in practical applications.

Our motivation in this work has been to dispense with the idea of a penalty function, with a view to providing an algorithm that does not require difficult decisions from the user in regard to the choice of penalty parameters. Within this constraint we look for an algorithm that is robust, is readily implemented, and exhibits rapid convergence in practice. Since we have second derivatives available we can expect to get second order convergence close to a local solution under mild conditions. One aspect of this that is not often emphasised is the need to obtain rapid and ultimately second order

R. Fletcher, S. Leyffer: University of Dundee, Department of Mathematics, Dundee, DD1 4HN, Scotland, U.K., e-mail: fletcher@maths.dundee.ac.uk, sleyffer@maths.dundee.ac.uk

Mathematics Subject Classification (2000): 90C30, 90C55, 49M30

convergence to some "best" solution in situations where the constraints of (P) are inconsistent (or locally so). Such situations occur quite often in practice, for example due to inadequate modelling. Some types of algorithm exhibit slow convergence and possibly ill-conditioning in such circumstances, thus leaving the user dissatisfied and uncertain of the outcome of the calculation.

In order to obtain second order convergence, it is necessary to have an underlying Newton iteration, and in the context of NLP the most attractive option is to take the Sequential Quadratic Programming (SQP) method as the basic iterative method. SQP methods date back to Wilson [14] and were popularized by Han [12] and Powell [13], see [8, Chapter 12.4] and Conn, Gould and Toint [4] for a selection of other references. In its basic form, SQP solves (P) via a sequence of quadratic programming (QP) approximations obtained by replacing the nonlinear constraints by a linear first order Taylor series approximation and the nonlinear objective by a second order Taylor series approximation augmented by second order information from the constraints. It can be shown under certain conditions that SQP converges quadratically near a solution.

If started far from a solution, however, the SQP method may fail to converge to a local solution of (P). In order to induce convergence, many popular methods use a *penalty function* which is a linear combination of the objective function f and some measure of the constraint violation. A related idea is an *augmented Lagrangian function* in which a weighted penalty term is added to a Lagrangian function. A step in an SQP method is then accepted when it produces a sufficient decrease in the penalty function.

There are several difficulties associated with the use of penalty functions, and in particular the choice of the penalty parameter. There is usually a threshold value below which the penalty function does not have a local minimum at the solution to (P). This threshold value is not known *a-priori*. Thus too low a choice can result in an infeasible point of (P) being obtained, or even an unbounded increase in the penalty. On the other hand, too large a choice damps out the effect of the objective function, resulting for example in slow convergence when the minimization method follows the curved boundary of the feasible region of (P). Methods have been suggested which change the penalty parameter adaptively, but these are also not without problems. Another concern is the effect of the nondifferentiability of some popular exact penalty functions. This has lead to the use of methods (e.g. the *watchdog* method [1]) in which the penalty function is allowed to increase for a limited number of iterations.

On the other hand, there is evidence that an SQP method can be made to work very well in practice without the use of a penalty function. In a recent Ph.D. thesis [15], Zoppke-Donaldson presents an SQP algorithm which does not require any penalty parameter or backtracking as in the watchdog technique. Although there is not a proof of global convergence, good numerical results are reported on a wide range of CUTE test problems and other practical problems.

This experience has encouraged us to experiment with a "minimalistic approach" to SQP. The aim is to interfere as little as possible with the SQP iteration but to do enough to give a bias towards convergence. The present paper describes one such approach and the experience gained with it. The new method is a trust-region based approach which introduces the concept of a "filter". It does not require the user to specify any penalty parameters and uses a standard QP solver. Most importantly though, the filter allows for a certain amount of non-monotonicity compared to a penalty function approach. This

property is seen to be advantageous in practice. It is worth noting that the ideas that are developed in the paper are not specific to SQP and trust-regions, and may also be applicable in other circumstances.

The ideas that are introduced in the filter approach are such that it is not obvious how to apply standard techniques for proving global convergence of penalty function methods (that is, convergence to a stationary point from an arbitrary initial estimate). At present the question of global convergence is open, although it has provoked some interesting discussions with other colleagues. In the context of our algorithm, heuristics are readily suggested that exclude obvious situations in which the algorithm might fail to converge, and we discuss this issue later in the paper. However we observe in practice that we can dispense with these heuristics and yet solve a substantial proportion of standard test problems using our algorithm. Thus any heuristics that might be suggested to enable a global convergence result to be proved should be cheap to apply and rarely invoked.

This paper is divided into 4 sections. The next section introduces the new concept of a filter and shows how it can be used in a trust-region based SQP algorithm. Infeasible QP problems are handled by the use of a feasibility restoration phase. In Sect. 3 algorithmic refinements are presented that are needed to ensure the robustness of the basic algorithm. Consideration of the asymptotic behaviour of the algorithm suggests the inclusion of a second order correction step. Also a new algorithm for the restoration phase is proposed, based on the use of a phase I filter. Finally, Sect. 4 presents some very encouraging numerical results obtained by solving problems from the CUTE test set [5].

We denote the gradient of f by $\mathbf{g} = \mathbf{g}(\mathbf{x}) = \nabla f(\mathbf{x})$, the Jacobian of the constraints by $\mathbf{A} = \mathbf{A}(\mathbf{x}) = \nabla \mathbf{c}^T(\mathbf{x})$, and the Lagrangian function by $\mathcal{L}(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \boldsymbol{\lambda}^T \mathbf{c}(\mathbf{x})$. The Hessian of the Lagrangian is denoted by $\mathbf{W} = \nabla^2 \mathcal{L}(\mathbf{x}, \boldsymbol{\lambda})$. Superscripts $^{(k)}$ refer to iterations indices and $f^{(k)}$ is taken to mean $f(\mathbf{x}^{(k)})$ etc. Quantities relating to local solutions of (P) are superscripted with a $*$.

2. An NLP filter

This section describes the basic concepts that underpin the new approach towards inducing global convergence in SQP.

There are two competing aims in nonlinear programming. The first is the minimization of the objective function f and the second is the satisfaction of the constraints. Conceptually, these two conflicting aims can be written as

$$\text{minimize} \quad f(\mathbf{x}) \tag{1}$$

and

$$\text{minimize} \quad h(\mathbf{c}(\mathbf{x})) \tag{2}$$

where

$$h(\mathbf{c}(\mathbf{x})) := \|\mathbf{c}^+(\mathbf{x})\|_1 := \sum_{j=1}^{m} c_j^+(\mathbf{x})$$

is the l_1 norm of the constraint violation. Here $c_j^+ = \max(0, c_j)$. The problem of satisfiability has now been written as a minimization problem. We use the l_1 norm because it has some convenient features that we exploit in Sect. 3, although other norms could also be considered for use.

A penalty function combines (1) and (2) into a single minimization problem. Instead we prefer to view (1) and (2) as separate aims, akin to a multi-objective optimization problem. However our situation is somewhat different from multi-objective optimization in that it is essential to find a point for which $h(\mathbf{c}(\mathbf{x})) = 0$ if at all possible. In this sense the minimization of $h(\mathbf{c}(\mathbf{x}))$ has priority.

Nevertheless, it is convenient to make use of the concept of *domination* from multi-objective optimization. Let $(f^{(k)}, h^{(k)})$ denote values of $f(\mathbf{x})$ and $h(\mathbf{c}(\mathbf{x}))$ evaluated at $\mathbf{x}^{(k)}$.

Definition 1. *A pair $(f^{(k)}, h^{(k)})$ is said to dominate another pair $(f^{(l)}, h^{(l)})$ if and only if both $f^{(k)} \leq f^{(l)}$ and $h^{(k)} \leq h^{(l)}$.*

In the context of nonlinear programming, this means that $\mathbf{x}^{(k)}$ is at least as good as $\mathbf{x}^{(l)}$ with respect to both (1) and (2). With this concept it is now possible to define a *filter*, which will be used in a trust-region type algorithm as a criterion for accepting or rejecting a trial step.

Definition 2. *A filter is a list of pairs $(f^{(l)}, h^{(l)})$ such that no pair dominates any other. A point $(f^{(k)}, h^{(k)})$ is said to be acceptable for inclusion in the filter if it is not dominated by any point in the filter.*

We stress that only two (later on four) scalars are stored for each entry in the filter. No vectors such as $\mathbf{x}^{(l)}$ are stored and this has negative implications for the use of backtracking in the resulting algorithm. The filter can be represented graphically in the (f, h) plane as illustrated in Fig. 1. Each point in the filter generates a block of non-acceptable points and the union of these blocks represents the set of points that are not acceptable to the filter.

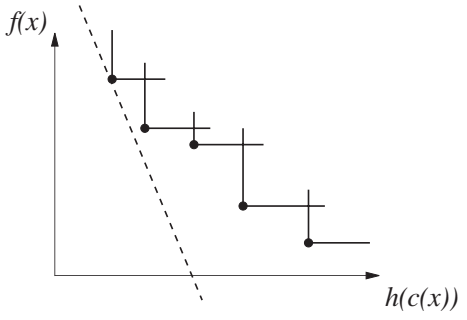

Fig. 1. Example of a filter and a penalty function

The key idea is to use the filter as a criterion for accepting or rejecting a step in an SQP method. Starting with $\mathbf{x}^{(k)}$, the solution of the current QP subproblem

$$(QP)\begin{cases}\underset{\mathbf{d}}{\text{minimize}} & \frac{1}{2}\mathbf{d}^T\mathbf{W}^{(k)}\mathbf{d}+\mathbf{d}^T\mathbf{g}^{(k)}\\ \text{subject to} & \mathbf{A}^{(k)^T}\mathbf{d}+\mathbf{c}^{(k)}\leq\mathbf{0}\\ & \|\mathbf{d}\|_\infty\leq\rho\end{cases}$$

produces a trial step $\mathbf{d}^{(k)}$. Set $\mathbf{x}^{(k+1)}=\mathbf{x}^{(k)}+\mathbf{d}^{(k)}$. The new trial point $\mathbf{x}^{(k+1)}$ is accepted by the filter if the corresponding pair $(f^{(k+1)}, h^{(k+1)})$ is not dominated by any point in the filter. Otherwise the step is rejected and the trust-region radius ρ is reduced. Thus the usual criterion of descent in the penalty function is replaced by the requirement that the new point is acceptable to the filter. The use of the l_∞ norm trust-region in (QP) (rather than the l_2 norm) ensures that (QP) remains tractable as a QP.

In a conventional trust-region algorithm, convergence is induced because reducing the radius of the trust region produces a step that is increasingly biased towards the steepest descent vector (in the relevant norm), and hence ensures sufficient descent when the current point is not stationary. In an SQP trust-region method, reducing the trust-region radius will ultimately give rise to an infeasible QP subproblem if the current point $\mathbf{x}^{(k)}$ is infeasible in the NLP problem. This situation is illustrated in Fig. 2 below. Note that a trust-region algorithm usually arrives at this situation after rejecting a number of consecutive steps. Thus it is not sufficient to simply increase the trust-region radius to regain feasibility. An infeasible QP subproblem also occurs when the linearized NLP constraints are themselves inconsistent. There are various ideas that might be invoked when the QP subproblem is infeasible, but we have chosen simply to minimize $h(\mathbf{c}(\mathbf{x}))$ in this situation. This is referred to as the *restoration phase*, and represents an attempt to get close to the feasible region of the NLP problem. One possible outcome is that the restoration phase finds a non-zero minimum of $h(\mathbf{c}(\mathbf{x}))$, which is taken as an indication that the NLP problem is infeasible (although of course this cannot be guaranteed). Otherwise we assume that the algorithm finds a point at which the QP subproblem is

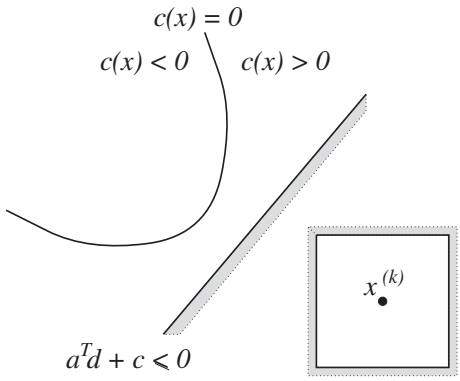

Fig. 2. Infeasible QP caused by small trust-region radius

feasible, and the above method can be continued from this point. We also assume for the moment that the new point is acceptable to the filter, and return to this issue in Sect. 3.

This gives rise to the basic filter SQP algorithm below which starts with an initial guess $\mathbf{x}^{(0)}$, $\boldsymbol{\lambda}^{(0)}$ of the solution and the associated Lagrange multipliers. On subsequent iterations we update the Lagrange multiplier estimates (by equating them to the multipliers of the QP subproblem) whenever the SQP iteration generates a new point that is accepted by the filter.

Algorithm 1. Basic filter SQP
Given $\mathbf{x}^{(0)}$ *and* ρ, *set* $k = 0$.

REPEAT
  IF the QP is infeasible **THEN**
    • Find a new point $\mathbf{x}^{(k+1)}$ in the restoration phase
  ELSE
    Solve (QP) for a step $\mathbf{d}^{(k)}$ and provisionally set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k)}$.
    IF $(f^{(k+1)}, h^{(k+1)})$ is acceptable to the filter **THEN**
      • Accept $\mathbf{x}^{(k+1)}$ and add $(f^{(k+1)}, h^{(k+1)})$ to the filter.
      • Remove points dominated by $(f^{(k+1)}, h^{(k+1)})$ from the filter.
      • Possibly increase the trust-region radius ρ.
    ELSE
      • Reject the step (set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$).
      • Reduce the trust-region radius ρ.
    ENDIF
  ENDIF
  Set $k = k + 1$.
UNTIL *Convergence*

We note that this algorithm is quite simple and there is no difficulty in choosing suitable parameters for adjusting the trust-region radius. The algorithm avoids the various difficulties of penalty function and augmented Lagrangian methods that were discussed in Sect. 1. There is no difficulty in following the curved boundary of the feasible region of the NLP problem, because increases in $h^{(k)}$ are allowed. We also note that a penalty function usually constitutes a stricter criterion for accepting an SQP step. The dashed line in Fig. 1 represents the isovalue of an exact penalty function through the current point in f, h space, and the area to the left of this line corresponds to points that reduce the penalty function. In our experiments we have observed that descent in the exact penalty function is usually a stricter criterion for a successful iteration than acceptability to the filter. In this sense the filter allows a large degree of non-monotonicity and this can be advantageous on some problems. Finally, the above algorithm works with any (indefinite) QP solver and does not require a special purpose QP solver like Sl_1QP.

In practice, almost all the test problems in Sect. 4 can be solved by this basic filter algorithm. However, there are some obvious ways in which the algorithm might fail, or might perform poorly, that can be removed by some simple heuristics. Such extensions are discussed in the next section.

3. Algorithmic extensions

Section 2 has described the basic concept of an NLP filter which we see as a new and interesting contribution which is particularly attractive on account of its simplicity. From this concept to the development of production software for large scale NLP problems is a very large step, involving many messy details which tend to obscure the simplicity of the original idea. The rest of this paper documents our progress in developing such software. This work was started in 1996 and contains features that with hindsight and the development of new convergence theorems, we might wish to change. However, numerical experience to follow is obtained in the manner described, whatever its limitations. In fact, it is our impression that although we may be able to simplify the heuristics in future work, we are unlikely to significantly change the overall quality of the results.

Certain algorithmic extensions of the basic algorithm are required to exclude known situations in which the basic filter algorithm might fail, or might converge slowly. We have tried to suggest heuristics that are cheap to implement, and which do not interrupt the algorithm in circumstances when it is working well. To some extent these heuristics detract from the simplicity of the basic algorithm, but something akin to what we suggest is likely to be needed if a global convergence proof is to be established. As yet however we do not have a global SQP convergence proof, and indeed there are some situations which are problematic. When the Hessian $\mathbf{W}^{(k)}$ is indefinite, we may compute local solutions to the QP that are not global. This might result in the predicted reduction

$$\Delta q^{(k)} = \frac{1}{2}\mathbf{d}^{(k)^T}\mathbf{W}^{(k)}\mathbf{d}^{(k)} + \mathbf{g}^{(k)^T}\mathbf{d}^{(k)}$$

being negligible, or even negative, in cases where a significant reduction exists. This situation arises in particular if $\mathbf{d} = \mathbf{0}$ is not feasible in (QP). Another doubtful situation occurs in relation to unblocking as described in Sect. 3.4. Both these situations are very rare and so far have not caused any practical difficulties. A global convergence proof for a filter SQP method is given in [10].

The extensions that we consider are a second order correction step, an upper bound on the constraint violation that provides a bias towards a feasible solution, an effective feasibility restoration phase to deal with infeasible QP problems, a mechanism for removing information from the filter under certain conditions, a sufficient reduction condition and a rule for accepting or rejecting points at the extremes of the filter. These extensions are dealt with in turn in the following subsections. The final SQP filter algorithm is presented towards the end of this section together with some remarks regarding the implementation of the method.

3.1. Second order correction step

An important property of any SQP code is that it usually exhibits second order convergence near the solution. The reason for this is that near the solution, an SQP method resembles Newton's method for solving the Kuhn-Tucker conditions and second order convergence is guaranteed under mild assumptions, as long as the unit step is accepted.

Unfortunately the use of a non-differentiable penalty function can preclude the acceptance of the unit step arbitrarily close to the solution, thereby preventing second order convergence. This is known as the Maratos effect. The difficulty can be avoided by computing a correction to the step that eliminates second order contributions of the nonlinear constraints. This is referred to as the Second Order Correction (SOC) step.

The usual example that illustrates the Maratos effect (take $\tau = 0$ and $\mu > 1$ in the example of [1]) does however cause no difficulty for the filter algorithm. Although the unit step causes an increase in the constraint violation, and hence an increase in the penalty function, it also causes a decrease in the objective function f and so is acceptable to the filter. Nevertheless, it is readily possible to exhibit an example, without introducing any pathological features, in which the SQP unit step increases *both* the objective *and* the constraint violation functions. Such an example is provided by Fig. 9.3.1 of [8], taking the case $\beta = \frac{1}{4}$. Any feasible point close to the origin illustrates the effect.

There are also advantages in using an SOC step when remote from the solution. It often reduces the constraint violation $h^{(k+1)}$ at the new point, improving the chances of accepting the new point with the current value of ρ, and so reduces the likelihood of having to reduce ρ. This is usually beneficial since small values of ρ can lead to slow convergence and possibly the need to enter the restoration phase. Also the SOC step is cheap to compute when a QP hot start facility is available. Thus we have included an SOC step in the algorithm.

The SOC step is computed whenever $\mathbf{x}^{(k+1)} = x^{(k+1,0)}$ is rejected by the filter in Algorithm 1. Following [8, p. 395] the QP that is solved is defined as

$$(QP_2) \quad \begin{cases} \underset{\mathbf{d}}{\text{minimize}} & \frac{1}{2}\mathbf{d}^T\mathbf{W}^{(k)}\mathbf{d} + \mathbf{d}^T\mathbf{g}^{(k)} \\ \text{subject to} & \mathbf{A}^{(k)^T}\mathbf{d} + \mathbf{c}^{(k+1,l)} - \mathbf{A}^{(k)^T}\mathbf{d}^{(k,l)} \leq \mathbf{0} \\ & \|\mathbf{d}\|_\infty \leq \rho, \end{cases}$$

for $l = 0, \ldots$ where $\mathbf{c}^{(k+1,0)} = \mathbf{c}^{(k+1)}$ and $\mathbf{d}^{(k,0)} = \mathbf{d}^{(k)}$. Let $\hat{\mathbf{d}}^{(k)} := \mathbf{d}^{(k,l+1)}$ denote the solution of (QP_2). The new point $\mathbf{x}^{(k+1,l)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k,l+1)}$ is tested in the filter. If $(f^{(k+1,l)}, h^{(k+1,l)})$ is acceptable to the filter, then the step is accepted and the trust-region radius may be increased. Otherwise a sequence of SOC steps is performed generating a sequence of trial points $\mathbf{x}^{(k+1,l)}$, $l = 0, \ldots$ until one of the following holds

1. an acceptable trial point $\mathbf{x}^{(k+1,L)}$ is found,
2. an infeasible QP is detected,
3. the rate of convergence of the SOC steps

$$r = \frac{h^{(k+1,l)}}{h^{(k+1,l-1)}}$$

   is considered to be too slow (i.e. larger than $\frac{1}{4}$ in our implementation) *or*
4. an almost feasible point with $h^{(k+1,l)} < \epsilon$ is generated (ϵ is the tolerance of the NLP solver).

In the first case the next iterate is $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k+1,L)}$ and the filter SQP algorithm continues. In the latter three cases, the steps are rejected and the trust-region radius is reduced (4. ensures the finiteness of this process). For later use, we store the *best* step of this sequence of SOC steps using a penalty function estimate to rank the steps.

3.2. *Upper bound on constraint violation*

We have included an additional necessary condition for accepting a point, namely $h(\mathbf{c}(\mathbf{x})) \leq u$, in which u is an upper bound on the constraint violation function. This is readily implemented by including an entry $(-\infty, u)$ in the filter. One reason for doing this is to prevent the (unlikely) situation in which a sequence of points for which $f^{(k+1)} < f^{(k)}$ and $h^{(k+1)} > h^{(k)}$ with $h^{(k)} \to \infty$ is accepted. We also use the upper bound as a means of solving the difficulty which occurs when the outcome of the restoration phase is a point which is unacceptable to the filter. In this case we allow ourselves to delete entries from the filter which block the acceptance of the new point. To avoid the possibility of cycling, we reduce the upper bound when this occurs. The details of this are explained in Sect. 3.4.

Our intention is that the upper bound should only rarely come into play. Thus the upper bound is initially set to a large value

$$u := \max\left(\texttt{ubd}\ ,\ \texttt{tt} \cdot h(\mathbf{c}(\mathbf{x}^{(0)}))\right)$$

determined by the initial constraint violation. In our implementation we choose $\texttt{ubd} = 100$ and $\texttt{tt} = 1.25$ as default values. Graphically the upper bound condition corresponds to requiring that all infeasible iterates lie in a "tube" around the boundary of the feasible region. This is similar to the tolerance tube idea of Zoppke-Donaldson.

3.3. *Feasibility restoration phase*

One undesirable effect of using a trust-region based approach is that reducing the trust-region radius may cause the QP problem to become infeasible, as illustrated in Fig. 2. Less frequently, the linearizations of the nonlinear constraints may themselves be inconsistent.

Our strategy for dealing with this situation is to enter a *restoration phase* in which we aim to get closer to the feasible region by minimizing $h(\mathbf{c}(\mathbf{x}))$. In our early work, we achieved this by making linear approximations to the nonlinear functions $\mathbf{c}(\mathbf{x})$ and solving a sequence of $l_\infty LP$ subproblems (using the l_∞ norm in the definition of $h(\mathbf{c})$). Two disadvantages of this approach emerged. To avoid having to use a special purpose $l_\infty LP$ solver, we solved the subproblem by a sequence of phase I LP subproblems. On some occasions this could be rather inefficient. A more fundamental disadvantage turns out to be the lack of second order information in the $Sl_\infty LP$ iteration, leading in some cases to slow convergence. One example occurs when the constraints are a system of n nonlinear equations in n unknowns. If there is a non-zero local minimum of h then it is essential to use second order information to obtain rapid convergence.

In developing a strategy to use second order information, we make use of a property of the phase I algorithm in our QP solver. If an infeasible QP is detected, the solver exits with a solution of the problem

$$(LP_1)\quad \begin{cases} \underset{\mathbf{d}}{\text{minimize}} & \displaystyle\sum_{j\in J} \mathbf{a}_j^{(k)^T}\mathbf{d} \\ \text{subject to} & \mathbf{a}_j^{(k)^T}\mathbf{d} + c_j^{(k)} \leq 0\ , \quad j \in J^\perp \\ & \|\mathbf{d}\|_\infty \leq \rho, \end{cases}$$

where $\mathbf{a}_j^{(k)} = \nabla c_j(\mathbf{x}^{(k)})$. Thus the QP solver partitions the constraints into two index sets $J \subset \{1, 2, \ldots, m\}$ and its complement $J^\perp = \{1, 2, \ldots, m\} \setminus J$. The set J contains infeasible constraints ($c_j^{(k)} + \mathbf{a}_j^{(k)^T}\mathbf{d} > 0 \quad j \in J$) whose l_1 sum is minimized at the solution to (LP_1), subject to the constraints in $J^\perp$ being satisfied. Note that this form of phase I problem is very general and includes several phase I problems as a special case. Suitable choices of J lead to the l_1 phase I problem or the "one-at-a-time-method". The l_∞ phase I problem can also be formulated in this way by introducing non-negative weights in the objective.

Our strategy in the restoration phase is to apply an SQP trust-region method to the NLP problem

$$(F)\begin{cases}\underset{\mathbf{x}}{\text{minimize}} & \displaystyle\sum_{j\in J} c_j^+(\mathbf{x}) \\ \text{subject to} & c_j(\mathbf{x}) \le 0\,, \quad j \in J^\perp\end{cases}$$

that is defined by this partitioning into sets J and $J^\perp$. One concern is that the sets J and $J^\perp$ are chosen by the phase I of the QP solver at each iteration. Thus it is possible that J and $J^\perp$ change from iteration k to iteration $k+1$, and this might make it difficult to enforce convergence. We have been able to use the upper bound heuristic in Sect. 3.2 as a way of avoiding this difficulty, and this is described in more detail below.

The restoration phase consists of a sequence of SQP-like iterations that continue whilst the subproblem (QP) is infeasible. Each iteration first checks feasibility of the system

$$(LP)\begin{cases}\mathbf{a}_j^{(k)^T}\mathbf{d} + c_j^{(k)} \le 0\,, \quad j = 1, 2, \ldots, m \\ \|\mathbf{d}\|_\infty \le \rho\end{cases}$$

derived from the constraints of (QP). If a feasible solution is found then the restoration phase ends. Otherwise sets J and $J^\perp$ are determined which solve (LP_1). Next a QP subproblem is determined by adding the second order term $\frac{1}{2}\mathbf{d}^T\mathbf{W}^{(k)}\mathbf{d}$ into the objective function of (LP_1), where

$$\mathbf{W}(\mathbf{x}, \boldsymbol{\lambda}) = \sum_{j\in J} \nabla^2 c_j(\mathbf{x}) + \sum_{j\in J^\perp} \lambda_j \nabla^2 c_j(\mathbf{x}).$$

The multipliers are those obtained from the solution of (LP_1). Note that the solution of (LP_1) can be used to initialize the solution of the new QP. Thus, using the QP hot start facility, the total effort of solving (LP_1) followed by the new QP subproblem, is almost the same as solving the new QP from scratch.

The observations regarding the asymptotic behaviour of the basic SQP algorithm also apply to the SQP iterations in the restoration phase, so we also allow a sequence of SOC steps if the solution of the restoration phase QP is rejected by the restoration phase filter.

Iterations in the restoration phase are continued until *either* a feasible QP is encountered (in which case the algorithm can return to the basic SQP algorithm) *or* an

infeasible Kuhn-Tucker point of (F) is found for some sets J and $J^{\perp}$, in which case the algorithm terminates with the indication that the NLP problem is (locally) infeasible.

We now consider the issue of global convergence of the restoration phase iterations. As in Sect. 2, there are two competing aims for problem (F), that is the need to maintain feasibility of the constraints in $J^{\perp}$ and the need to minimize the weighted sum of constraint violations in J. Unlike Sect. 2 however, a suitable penalty function is the total sum of the constraint violations. Nevertheless, using a filter gives greater flexibility in accepting steps and we have found it preferable to use a filter in the restoration phase as well. This filter is referred to as the *restoration filter* or *phase I filter*. Thus we define

$$h_J := h(\mathbf{c}_J(\mathbf{x})) := \sum_{j \in J} c_j^+(\mathbf{x})$$

and similarly $h_{J^{\perp}}$.

Definition 3. *A phase I filter is a list of pairs* $(h_J, h_{J^{\perp}})$ *such that no pair dominates any other.*

Note that J and $J^{\perp}$ may change from iteration to iteration. Ideally, one would like to update the entries in the filter correspondingly. However, this would require storage of *all* values $c_j(\mathbf{x}^l)$ for *all* entries l in the filter. Nonetheless, we still find that this definition provides a useful filter criterion. However, we found one example where this approach did not work well. In this example, good progress was initially made and the trust-region was increased. Thus more constraints could be satisfied and moved from J into $J^{\perp}$. This is desirable as one aim of the restoration phase is to generate a feasible QP (i.e. $J = \emptyset$) and return to the minimization of the original problem. However, in our example the resulting increase in $h_{J^{\perp}}$ was dramatic, while the corresponding reduction in h_J was small (compared to the magnitude of h_J). As a result this desirable step was rejected. We therefore allow blocking entries to be removed from the filter when the set J changes.

The mechanism of unblocking is outlined in more detail in Sect. 3.4. Unblocking takes place after the sequence of SOC steps has failed to generate an acceptable point. At this stage there are a number of trial steps available to the algorithm which have been generated at this iteration: the LP step from (LP_1), a QP step and possibly a number of SOC steps. Let $\hat{\mathbf{x}}^{(k,L)}$ denote the best trial point of iteration k, i.e. the point with lowest l_1 norm. The filter is then unblocked if $\hat{h}^{(k,L)}$ is sufficiently less than the current upper bound *and* J, $J^{\perp}$ have changed from the previous iteration. Whenever the filter is unblocked the upper bound is reduced to $u = \max\{\hat{h}^{(k,L)}, u/10\}$ to ensure that cycling cannot occur.

A detailed description of the restoration phase can now be given.

Algorithm 2. Feasibility Restoration
Given $\mathbf{x}^{(k)}$, ρ *and a constraint upper bound* u *from the SQP solver.*

REPEAT
Solve (LP_1); obtain index sets J and $J^{\perp}$ (fixed for this iteration).
IF (LP_1) is feasible **THEN**
Return to normal SQP (phase II) and clear the restoration filter.

ELSE
- Solve the phase I QP (adding a second order term to (LP_1)).
- Let the solution be $\mathbf{d}^{(k)}$ and set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k)}$.

IF $(h_J^{(k+1)}, h_{J^\perp}^{(k+1)})$ is acceptable to the phase I filter **THEN**
- Accept $\mathbf{x}^{(k+1)}$ and add $(h_J^{(k+1)}, h_{J^\perp}^{(k+1)})$ to the filter.
- Remove points dominated by $(h_J^{(k+1)}, h_{J^\perp}^{(k+1)})$ from the filter.
- Possibly increase the trust-region radius ρ.

ELSE

Solve sequence of QPs for SOC step $\hat{\mathbf{d}}^{(k)}$; set $\hat{\mathbf{x}}^{(k+1)} = \mathbf{x}^{(k)} + \hat{\mathbf{d}}^{(k)}$

IF $(\hat{h}_J^{(k+1)}, \hat{h}_{J^\perp}^{(k+1)})$ is acceptable to the phase I filter **THEN**
- Accept $\hat{\mathbf{x}}^{(k+1)}$ and add $(\hat{h}_J^{(k+1)}, \hat{h}_{J^\perp}^{(k+1)})$ to the filter.
- Remove points dominated by $(\hat{h}_J^{(k+1)}, \hat{h}_{J^\perp}^{(k+1)})$ from the filter.
- Possibly increase the trust-region radius ρ.

ELSE

IF (J changed from iteration $k-1$ and $\hat{h}^{(k,L)} < u$) **THEN**
- Accept the best SOC step (set $\mathbf{x}^{(k+1)} = \hat{\mathbf{x}}^{(k+1)}$).
- Remove all blocking entries from the restoration filter.
- Reduce the upper bound to $u = \max\{\hat{h}^{(k,L)}, u/10\}$

ELSE
- Reject the step (set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$).
- Reduce the trust-region radius ρ.

ENDIF

ENDIF

ENDIF

ENDIF

Set $k = k + 1$.

UNTIL *Convergence or return to SQP*

See implementation detail 5 in Sect. 3.7 for a discussion of the convergence criteria used. The value of the upper bound u is discarded at the end of the restoration phase and not used in the SQP algorithm. Note that each time the restoration filter is unblocked, the upper bound u is reduced and this acts as a way of forcing reduction in the l_1 sum of constraint violations.

The trust-region radius ρ is changed in Algorithm 2. At its return to SQP, the current value of ρ reflects how well a first order Taylor series approximates the nonlinear constraints about $\mathbf{x}^{(k)}$. We therefore use this value of ρ in the next SQP step.

3.4. Removing entries from the filter

Consider the (unlikely) scenario that (P) has a global solution $\mathbf{x}^{**}$ and a worse local solution $\mathbf{x}^*$, that $\mathbf{x}^{(0)}$ is a feasible point fairly close to $\mathbf{x}^{**}$, but subsequent iterates $\mathbf{x}^{(k)}$ are converging to $\mathbf{x}^*$. Then, if $f^{(0)} \leq f^*$, the filter entry $(f^{(0)}, h^{(0)})$ prevents the SQP iteration from converging to $\mathbf{x}^*$. Of course we would like to backtrack to $\mathbf{x}^{(0)}$ in this

situation, but our decision not to store $\mathbf{x}^{(0)}$ with the filter information precludes this. We refer to $(f^{(0)},\ h^{(0)})$ as a *blocking entry* in the filter. To enable convergence to $\mathbf{x}^*$ to take place, we insist that the point resulting from the restoration phase is included in the (phase II) filter, and any blocking entries like $(f^{(0)},\ h^{(0)})$ are removed.

However, deleting blocking entries might allow cycling (e.g. if the algorithm were subsequently to return to $\mathbf{x}^{(0)}$). To prevent this we use the upper bound mechanism introduced in Sect. 3.2. If a blocking entry is removed, we reduce the upper bound u by

$$u = \max(h^{(k)},\ u/10)$$

where $h^{(k)}$ is the constraint violation function at the new point that is added to the filter. The idea is illustrated in Fig. 3. Because the upper bound constraint acts like a filter entry with $f = -\infty$ and a sufficient reduction criterion is imposed, this ensures that cycling cannot occur. If the upper bound were repeatedly reduced, then the effect of this test and the sufficient reduction condition of the next section would be to drive $\mathbf{x}^{(k)}$ arbitrarily close to the boundary of the feasible region. In this case it can be expected that the SQP trust-region method with SOC steps would ultimately cause f to be decreased monotonically.

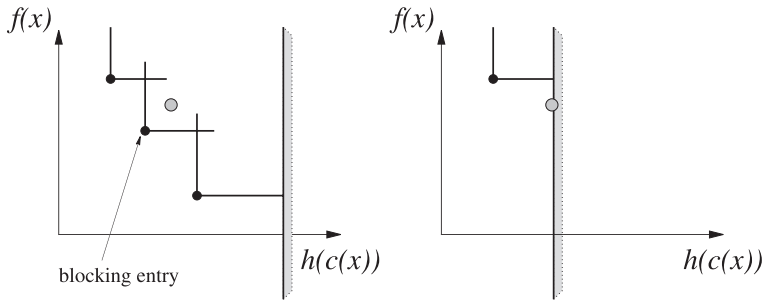

Fig. 3. Removing blocking entries from the filter

Whenever unblocking occurs there is a choice of steps available (from QP and SOC steps) that have been rejected by the filter. We take advantage of this by using the step with the least penalty function value (or total constraint violation in phase I) to unblock the filter.

The heuristics in this and the next two subsections were chosen with the intention that they would only come into play very occasionally, and numerical experience so far indicates that this is indeed the case. We regard them largely as an indication of what might be required to produce a global convergence proof for a filter-type algorithm.

We have also considered other approaches to unblocking in our earlier work on filter algorithms. One such approach was to shift the constraints in the NLP problem, which would become

$$(P_\theta) \begin{cases} \underset{\mathbf{x}}{\text{minimize}} & f(\mathbf{x}) \\ \text{subject to} & \mathbf{c}(\mathbf{x}) \leq \theta \mathbf{e}, \end{cases}$$

where $\theta \geq 0$ is a shift parameter chosen so that the current point is just feasible. Then subsequent SQP iterations are applied to (P_θ) until either the blocking point can be removed, or (P_θ) is solved. Only in the latter case is unblocking carried out. We discontinued work on this idea for various reasons, one of which being that it is better suited to the use of the l_∞ norm to define $h(\mathbf{c})$. This would give a less convenient restoration phase NLP problem than that in (F) requiring an l_∞ QP solver. The use of J, $J^\perp$ is more general and less restrictive for the QP solver being used.

3.5. Sufficient reduction

Clearly, the condition that a new point is not dominated by any entry in the filter allows the possibility of an oscillating sequence of points with accumulation points in (f, h) space that are not Kuhn-Tucker points. A standard way to avoid this in a penalty function algorithm is to require a sufficient reduction in the penalty function on each iteration. A similar idea can be used with a filter algorithm. The aim is to produce an envelope below the filter that prevents points arbitrarily close to the filter from being accepted. This idea is illustrated in Fig. 4 where the envelope is shown by the dashed line.

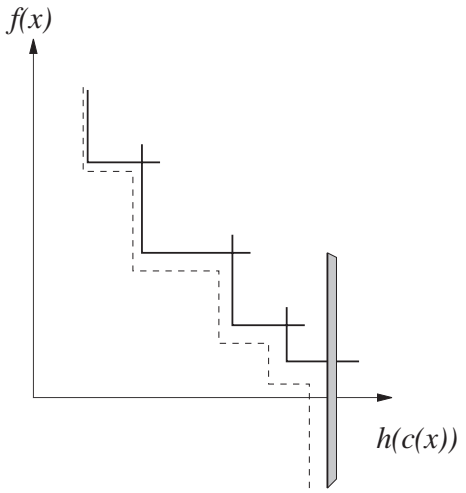

Fig. 4. Envelope created by sufficient reduction conditions

A new iterate $\mathbf{x}^{(k+1)}$ is said to be acceptable to the filter, if

$$h^{(k+1)} \leq \beta h^{(l)} \tag{3}$$

or

$$f^{(k+1)} \leq f^{(l)} - \max\left(\alpha_1 \Delta q^{(l)},\ \alpha_2 h^{(l)}\ \mu^{(l)}\right) \tag{4}$$

holds for all filter entries l. Here, β, α_1 and α_2 are positive constants which we have taken to be 0.99, 0.25 and 0.0001 respectively.

Equation (3) creates an envelope in the h direction in the filter. The envelope in the f direction is created by computing the predicted reduction $\Delta q^{(l)}$ in f at the QP generated by $\mathbf{x}^{(l)}$, choosing the sign so that $\Delta q^{(l)} > 0$ if a decrease in f is predicted. However, it is possible that the QP predicts an increase in f, in which case it is necessary to define the envelope in an alternative way. To this end we compute an estimate of the penalty parameter $\mu^{(l)}$ which is the least power of ten larger than $\|\boldsymbol{\lambda}^{(l)}\|_\infty$ and cut this value off to lie in the interval $[10^{-6}, 10^6]$. The value of $h^{(l)}\,\mu^{(l)}$ can then be used as a predicted reduction in f. Both $\Delta q^{(l)}$ and $\mu^{(l)}$ are stored along with $f^{(l)}$ and $h^{(l)}$ in the filter.

The rationale behind the second term, $\alpha_2 h^{(l)}\,\mu^{(l)}$ is as follows: $\mu^{(l)}$ measures the marginal effect of changes in f due to changes in h. A QP step predicts a reduction of $h^{(l)}$ to zero, equivalent to a predicted reduction in f of $h^{(l)}\,\mu^{(l)}$.

3.6. Beyond the extreme points of the filter

The filter is a reliable oracle for deciding whether or not to accept a step as long as the new constraint violation lies within the range of constraint violations recorded in the filter. However the current heuristics do not exclude the possibility of generating a sequence of filter entries in which $\{f^{(k)}\}$ is monotonically increasing and $\{h^{(k)}\}$ is monotonically decreasing, without converging to a Kuhn-Tucker point. As long as the sufficient reduction criterion in h is satisfied by the sequence, the points will be accepted by the filter, a point brought to our attention by Richard Byrd.

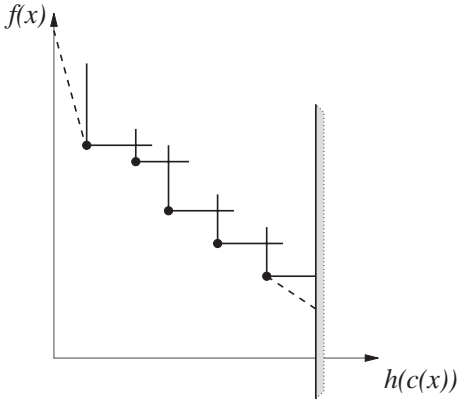

Fig. 5. North-West & South-East corner rules

We have therefore introduced an additional heuristic when $h^{(k+1)}$ provides sufficient reduction from $h^{(1)}$, where $h^{(1)}$ now refers to the leftmost entry in the filter. We make an overestimate $\mu = 1000\,\mu^{(1)}$ of the penalty parameter above, and require $\mathbf{x}^{(k+1)}$ to provide a reduction in the resulting exact penalty function. The value of the exact penalty

function corresponding to the leftmost entry $f^{(1)}$, $h^{(1)}$ is $f^{(1)} + \mu h^{(1)}$ and the isovalue of this function is illustrated by the dashed line in Fig. 5. The new point is then accepted if in addition to sufficient reduction from $h^{(1)}$ it satisfies

$$f^{(k+1)} + \mu h^{(k+1)} \leq f^{(1)} + \mu h^{(1)}.$$

We refer to this as the *North-West corner rule*. Hopefully this will be sufficient to avoid the adverse scenario.

A similar rule is applied in the South-East corner of the filter, even though the upper bound heuristic should be sufficient to exclude any non-convergent sequences. However it seems a good idea to include a similar cut, since the initial upper bound on h may be very large. Thus we again use the above test, but with μ defined by $\mu = \mu^{(L)}/1000$, where $\mu^{(L)}$ is the penalty parameter of the rightmost filter entry, as introduced in the previous section.

Presently, we allow the NW/SE corner rules to become violated during the unblocking of the filter. This is necessary, as the NW corner rule might conceivably exclude the global minimum point from being selected. A consequence of this is that there still exists the possibility that a sequence with $f \to \infty$ and $h \to 0$ may occur. However, we view this as very unlikely to occur in practice, as unblocking can only takes place on return from phase I. The situation in phase I is more satisfactory as unblocking enforces an upper bound on the total constraint violation. This acts like an upper bound on h_J and avoids the adverse scenario.

More recent work has shown that we can dispense with the NW/SE corner rule. We hope to discontinue using this feature of our present code in the future guided by the convergence proof in [10].

3.7. The complete SQP algorithm

The complete SQP algorithm based on the filter concept can now be stated. Note that function and gradient evaluations are not included as explicit statements in the algorithm, and Lagrange multiplier estimates are taken from the most recent successful QP. The term "acceptable to the filter" is taken to mean that the new point is not dominated by any entry in the filter (including the upper bound entry) and satisfies both the sufficient reduction and North-West and South-East corner rules.

Algorithm 3. Filter SQP
Given $\mathbf{x}^{(0)}$ *and* ρ. *Set* $k = 0$.

REPEAT
Solve (QP) for a step $\mathbf{d}^{(k)}$.
IF (QP) is infeasible **THEN**
Enter restoration phase (see Algorithm 2). Return to normal SQP
when $\mathbf{x}^{(k+1)}$ is found whose corresponding QP is feasible.
ELSE
Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{d}^{(k)}$.
IF ($f^{(k+1)}, h^{(k+1)}$) is acceptable to the filter **THEN**

- Accept $\mathbf{x}^{(k+1)}$ and add $(f^{(k+1)}, h^{(k+1)})$ to the filter.
- Remove points dominated by $(f^{(k+1)}, h^{(k+1)})$ from the filter.
- Possibly increase the trust-region radius ρ.

ELSE
IF $(h^{(k+1)} > 0)$ **THEN**
- Solve sequence of QPs for SOC step $\hat{\mathbf{d}}^{(k)}$; set $\hat{\mathbf{x}}^{(k+1)} = \mathbf{x}^{(k)} + \hat{\mathbf{d}}^{(k)}$.

IF $(\hat{f}^{(k+1)}, \hat{h}^{(k+1)})$ is acceptable to the filter **THEN**
- Accept $\hat{\mathbf{x}}^{(k+1)}$ and add $(\hat{f}^{(k+1)}, \hat{h}^{(k+1)})$ to the filter.
- Remove points dominated by $(\hat{f}^{(k+1)}, \hat{h}^{(k+1)})$ from the filter.
- Possibly increase the trust-region radius ρ.

ENDIF
ENDIF
IF (no acceptable point) **THEN**
IF (1^{st} iteration after restoration phase) **THEN**
- Accept the best SOC step (set $\mathbf{x}^{(k+1)} = \hat{\mathbf{x}}^{(k+1)}$).
- Remove all blocking entries from the filter.
- Reduce the upper bound to $u = \max(\hat{h}^{(k+1)}, u/10)$

ELSE
- Reject the step (set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$).
- Reduce the trust-region radius ρ.

ENDIF
ENDIF
ENDIF
ENDIF
Set $k = k + 1$.
UNTIL *Convergence*

Further details regarding the specific implementation of the algorithm are listed below.

Implementation Details:

1. Whenever the trust-region is reduced, the trust-region radius ρ is set to $\rho = \min(\rho, \|\mathbf{d}^{(k)}\|_\infty)/2$ and whenever it is judged that the radius should be increased it is doubled (that is $\rho = 2\rho$). This is broadly in line with other trust-region algorithms, and these algorithms are usually insensitive towards the exact choice of these constants. We have experimented with more sophisticated strategies but these did not show a significant improvement.
2. The usual condition for increasing the trust-region radius is to require that $\rho = \|\mathbf{d}^{(k)}\|_\infty$ and that there is "good agreement" between the actual and predicted decrease in the penalty function. As no penalty function is employed in the filter SQP algorithm, a different approach has been used.
   In the present implementation ρ is increased whenever $\rho = \|\mathbf{d}^{(k)}\|_\infty$ *and* the SQP step is accepted. If an SOC step is accepted, then the trust-region is only increased if the rate of convergence of the SOC step (r in Sect. 3.1) is deemed to be sufficiently good (i.e. below $1/10$). We have observed strong oscillations in the trust-region values on some problems and these have been alleviated by this tactic.

3. The algorithm requires an indefinite QP solver and there are no serious restrictions on what could be used. We have used some particular features of the phase I part of our QP solver, but we envisage that similar ideas could be used with other approaches to phase I.
4. The NLP solver always starts by solving a Linear Programming problem which contains only the linear constraints of (*P*) (including simple bounds on the variables). As a consequence, all linear constraints remain feasible throughout the SQP process. Our solver allows upper and lower bounds on all constraints, and so enables equality constraints to be readily incorporated in the algorithm.
5. The solver terminates if the infeasibility (or $h_{J^\perp}$ in the restoration phase) and the (normalized) Kuhn-Tucker residual is less than ϵ where ϵ is a user provided tolerance. For further details see [9].
   The algorithm also terminates if the step $\|\mathbf{d}\|_\infty$ or the trust-region radius ρ are less than ϵ. In these cases an assessment of the quality of the solution can be gained by also examining the maximum modulus Lagrange multiplier. If this is large, then it is possible that a solution has been obtained but the active constraints are nearly dependent.

4. Numerical results

A fully tested and documented production code for filter SQP has been written. The code has been interfaced to CUTE and the modelling language AMPL [11]. It is also available under NEOS at `http://www-neos.mcs.anl.gov/neos/`.

This section presents some numerical experience that has been obtained with an implementation of the filter SQP algorithm. All NLP test problems are taken from the CUTE test set and we are grateful to Conn, Gould and Toint for writing and distributing CUTE freely. The ready availability of a large test set has certainly made the task of developing and testing NLP software a lot easier.

In the tests exact second derivatives were used. All routines are written in FORTRAN 77 and run on a SPARCstation 4 with 128 Mb memory under Solaris 2.x compiled with the -O option. The QP solver used is bqpd (see [7]) combined with a sparse linear algebra package. For the initial trust region radius we choose $\rho = 10$ and the tolerance was set to $\epsilon = 1.\text{E} - 6$. We used LANCELOT's default tolerance of `1.E-5`.

We have also implemented a version of Fletcher's Sl_1QP method [6]. The penalty parameter for this is chosen such that μ is the least power of 10 that is greater than $\|\boldsymbol{\lambda}^*\|_\infty$, where $\boldsymbol{\lambda}^*$ is the vector of Lagrange multipliers at the solution. These are the most favourable circumstances under which to run Sl_1QP, in that $\boldsymbol{\lambda}^*$ would not normally be known in practice, and a poor guess could considerably increase the solution time. The linear constraints and simple bounds are not included in the l_1 penalty function. Instead an initial LP is solved to provide a linear feasible starting point as for filter SQP.

4.1. Comparison to LANCELOT *and Sl*$_1$*QP on small test problems*

The algorithms were compared on *all* 186 smooth CUTE problems with $n, m \leq 25$ and nonlinear constraints. After eliminating the problems for which the solvers obtained

different local minima, 127 problems remained. Table 1 summarises these results. The relative ranking based on CPU time of the solvers is given in Fig. 6. A breakdown of the results for Filter SQP and Sl_1QP is shown in Fig. 7 below. More detailed results are available from `http://www.maths.dundee.ac.uk/~sleyffer`.

Table 1. Summary of 127 CUTE small test problems

	Crashes	Results for 127 problems			
	out of 186	f-evals	c-evals	g-evals	CPU
Filter SQP	0	1797	2815	1567	37.95
S*l*1QP	2	6168	6193	3966	107.45
LANCELOT	12	–	–	8542	82.47

The performance on these problems can be summarized as follows. The new filter SQP is more reliable and faster than both Sl_1QP and LANCELOT. Of the latter two, Sl_1QP is more reliable than LANCELOT. This is also confirmed by the relative ranking in Fig. 6.

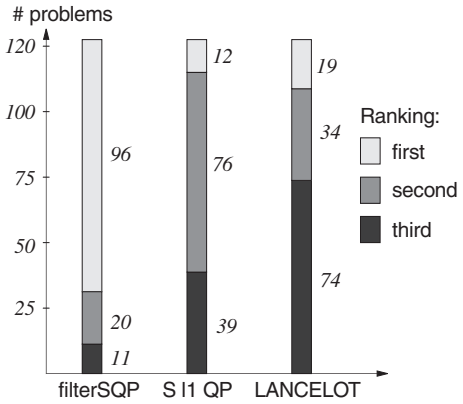

Fig. 6. Relative ranking of the three solvers

Finally, we compare the number of gradient evaluation of filter SQP and Sl_1QP in Fig. 7. This figure shows how many problems are solved for a certain range of number of gradient evaluations. It is noteworthy that filter SQP solves more problems with fewer gradient evaluations than Sl_1QP, even though the latter uses a penalty parameter derived from $\boldsymbol{\lambda}^*$, which would not normally be available to the solver.

4.2. Comparison to LANCELOT *on large test problems*

Next, the new filter SQP solver is compared to LANCELOT [2] (with default settings) on *all* problems in CUTE with nonlinear constraints and up to $n, m \leq 3100$ variables or constraints. Problems of variable size were solved with the largest available size in

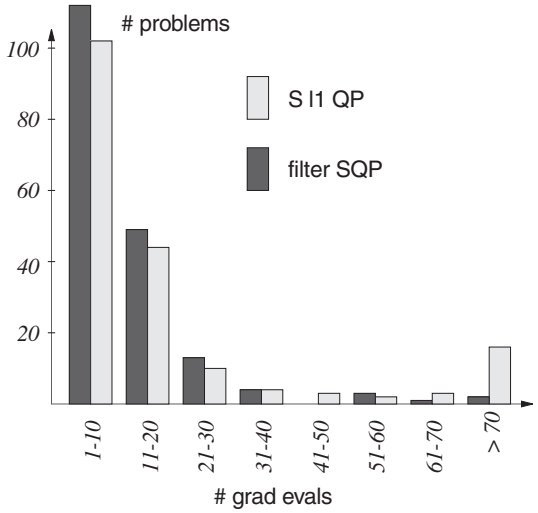

Fig. 7. Number of gradient evaluations for 186 small test problems

the above range. Some problems were removed from the test set because they produced IEEE arithmetic exceptions or were not differentiable. These problems are listed in Table 2. This resulted in a total of 159 remaining test problems (out of these problems, the two solvers obtained different minima on 45).

Table 2. Removed test problems

PROBLEM	Reason for removal
CONCON	Non-smooth problem `X*ABS(X)` etc.
DITTERT	IEEE exception at starting point (filter SQP)
LHAIFAM	IEEE exception at starting point (filter SQP)
MCONCON	Non-smooth problem `X*ABS(X)` etc.
NET2	IEEE exception at starting point (filter SQP)
NET3	IEEE exception at starting point (filter SQP)
ORTHRDM2	Zero Jacobian at start (crash LANCELOT)
YORKNET	Successive IEEE exceptiones (filter SQP)

Tables 6 to 9 give the results for these 159 medium or large sized nonlinear problems. Table 3 gives a description of the headers of these tables. Table 4 and 5 give the problem characteristics of the larger problems. Problems 1 to 22 are fixed sized problems with $n, m \leq 100$, problems 23 to 45 are large fixed sized problems and the remainder are variable sized problems.

Figure 8 shows the amount of CPU time used by filter SQP as a proportion of LANCELOT's CPU time on a log-scale. The problems are ordered as in the result tables from left to right. Finally, Fig. 9 shows how often one solver outperforms the other. In particular, there are 23 problems on which filter SQP is more than 100 times faster than LANCELOT with only 5 problems for which LANCELOT is 100 times faster.

Table 3. Description of headers of the result tables

Header	Description
PROBLEM	The CUTE name of the problem being solved.
n	Number of variables of the problem.
m	The total number of constraints (excl. simple bounds).
m_{nln}	Number of nonlinear constraints.
k	The dimension of the null-space at the solution.
QPs	Number of (phase II) QP problems solved.
# inf	Number of phase I QP problems solved.
SOCS	Number of second order correction steps.
# f	Number of objective function evaluations.
# c	Number of constraint evaluations.
# g	Number of gradient/Jacobian evaluations.
ρ	The final trust region radius ρ.
iter	Number of iterations needed to solve the problem.
CPU	Seconds of CPU time needed for the solve.

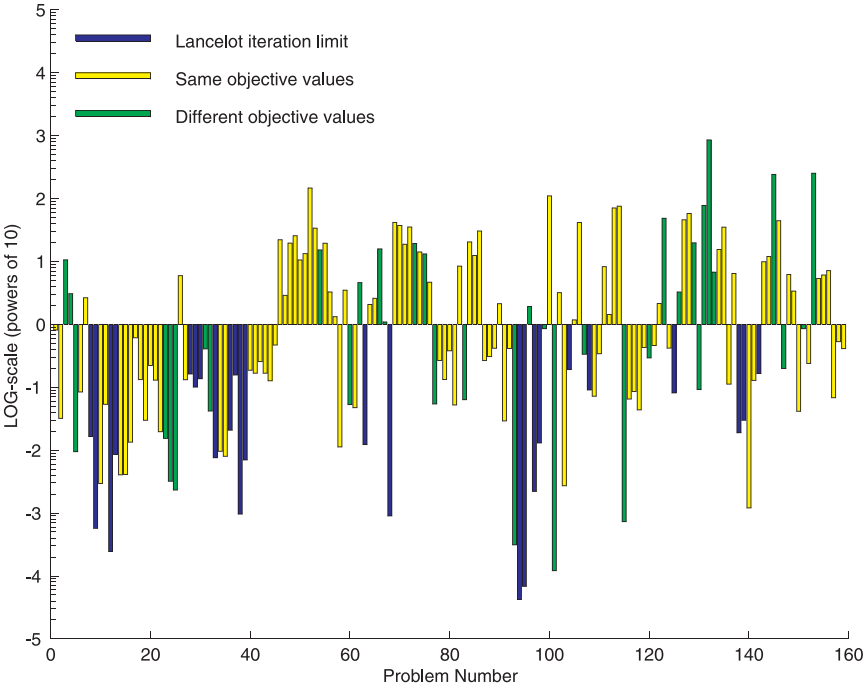

Fig. 8. CPU times of filter SQP as a proportion of LANCELOT times

Overall, filter SQP outperforms LANCELOT more often than not. However, these results are not quite as encouraging as our earlier results suggested. In an earlier version of this paper we had tested the two solvers on a subset of test problems which included all fixed sized problems but only a small arbitrary selection of variable sized problems. As it turns out, some of these variable sized problems turn out to be more favourable to LANCELOT than to filter SQP and this has tilted the balance somewhat.

Examining the problem instances where LANCELOT is significantly faster than filter SQP, we notice particularly that there are three groups of problems in Tables 6 to 9 where this occurs. These illustrate two specific types of situation which favour LANCELOT.

1. The BRATU* problems (48–50, 55, 56) represent discretizations of PDEs. LANCELOT's conjugate gradient solver can be expected to perform well in this situation and this shows in the CPU times. Note that in terms of iterations, LANCELOT is not significantly faster than filter SQP. The better performance is due to the fact that each iteration of LANCELOT is much faster than a QP solve.
2. The DTOC* problems (70–76) have a large null-space (with dimension of about 1000). This is unfavourable for our QP solver which repeatedly has to factorize a dense reduced Hessian of dimension 1000. LANCELOT again reaps the benefit of conjugate gradients in this case. Likewise, the ORTHR* problems (129–133, 135) and SSNLBEAM (149) also have a large null-space and the same comments apply.

We stress that our code works reliably in these situations and the discrepancy is simply due to the relative performance of a direct method as against the use of conjugate gradients. This discussion shows the difficulty of assessing relative performance by a mere comparison of overall results. Clearly, LANCELOT could be made to appear either better or worse by increasing or decreasing the number of such test problems in the test set. How often these types of problem arise in practice is difficult to assess and is clearly application dependent. However, we note that the fixed size test problems, many of which are practical applications, show up filter SQP very favourably.

There are also circumstances under which LANCELOT performs very poorly. The first occurs with the FLOS* problems which are linearly infeasible. This is discovered immediately by filter SQP as the initial LP is infeasible whereas LANCELOT does not discover this. A second set of circumstances where LANCELOT performs poorly arises if many inequality constraints are present. This is due to the fact that it adds a slack variable for each inequality. Thus the OET* problems which have $n = 5$ variables and $m = 1002$ constraints become problems in 1007 variables for LANCELOT. A remedy for this is provided in [3].

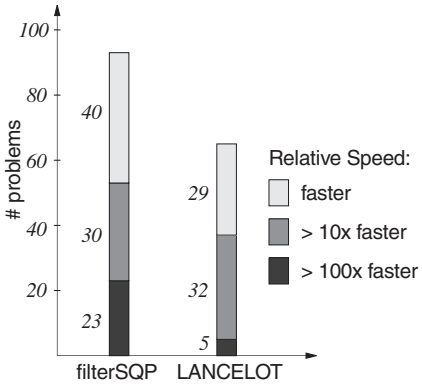

Fig. 9. Relative performance of filter SQP and LANCELOT for large test problems

We also observed that filter SQP outperforms LANCELOT on problems with many equations. By switching to a genuine feasibility restoration phase, filter SQP exhibits fast local convergence even if the NLP problem is (locally) inconsistent. Of course the performance of LANCELOT can also be improved for some problems by switching to options which force accurate solution of the subproblems.

Finally, filter SQP fails on 6 out of 159 large problems. These are CATENARY, COSHFUN, SINROSNB and UBH5, where the iteration limit is reached and EIGENC and QR3DBD where the trust-region radius becomes very small (10^{-6}). For the first 4 problems, the failure is due to filter SQP failing to get close enough to the feasible region in 1000 iterations. Tightening the upper bound parameters to ubd $= 10$ and tt $= 10^{-4}$ filter SQP was able to solve CATENARY, COSHFUN and SINROSNB in 569, 34 and 34 iteration respectively. UBH5 was solved in 28 iterations with ubd $= 10^3$ and tt $= 10$. EIGENC and QR3DBD crash because successive QP are very ill-conditioned which results in loss of feasibility with respect to the linear constraints. We were able to obtain a (local) minimum of the constraint violation for EIGENC by treating all constraints as nonlinear (final $\rho = 1.19209$E-4, 192 iterations). However, filter SQP still failed for QR3DBD. On the other hand, LANCELOT fails on 24 problems.

On three problems (HS99EXP, MESH, COSHFUN), filter SQP reports a final trust-region radius in excess of 1.E6. HS99EXP appears to be poorly scaled; we observe second order convergence at the end. MESH is unbounded (in the sense that there exists feasible points with a function value less than -1.E20). Finally, we conjecture that COSHFUN's objective is unbounded in a neighbourhood of the feasible region: we observe function values less than -1.E11 at points relatively close to the feasible region.

In the present set of experiments scaling has not been used, although ultimately we regard this as desirable and expect that it would improve the performance on badly scaled problems. The scaling option allows the user to supply orders of magnitude estimates of the variables, which are used to scale the trust-region and other aspects of the problem. Surprisingly enough, poor scaling seems to cause difficulty on only a small number of problems.

4.3. Performance of filter SQP

A very encouraging feature of the SQP algorithm is the small number of gradient evaluations (or similarly Hessian evaluations or QP solves) that are required to solve most problems. Figures 7 and 10 give information about the number of gradient evaluations required for smaller and larger problems. A large proportion of problems were solved with fewer than 10 gradient evaluations. The fact that this also holds for the larger problems makes SQP – in our view – suitable for large scale optimization.

An interesting question is to what extent the heuristics introduced in Sects. 3.6, 3.4 and 3.2 affect the algorithm. The two tables below give the number (and percentage) of steps that are rejected by these heuristics for the small and large test problems respectively. Overall, the NW and SE corner rule interfere very rarely with the algorithm. Unblocking is similarly rare, though it happens more often (in relative terms) in the feasibility restoration phase. The reason for this is that the unblocking rule is less

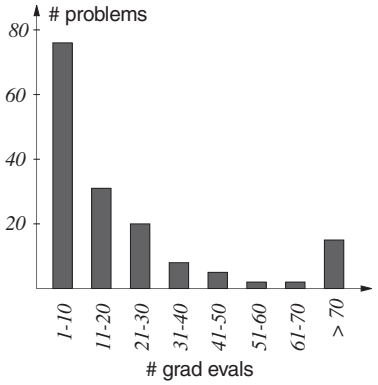

Fig. 10. Number of gradient evaluations for large test problems

stringent than in phase II. The upper bound affects about 10 % of iterations of filter SQP. This appears to be more significant. However, we observe that the upper bound plays an important part early in the iteration, but rarely interferes with later iterations, once the algorithm is relatively close to the feasible region. We believe that the heuristics do not overly restrict the performance of the algorithm.

Occurrences of filter heuristics for small test problems

	NW-corner	SE-corner	upper bnd	unblock	iter
phase I	0	0	80 (11%)	12 (2%)	701
phase II	9 (0.4%)	7 (0.3%)	233 (11%)	27 (1%)	2106
Total	9 (0.3%)	8 (0.3%)	313 (11%)	39 (1.4%)	2807

Occurrences of filter heuristics for large test problems

	NW-corner	SE-corner	upper bnd	unblock	iter
phase I	0	0	1050(29%)	150 (4%)	3636
phase II	25 (0.4%)	16 (0.3%)	933 (17%)	59 (1%)	5620
Total	25 (0.3%)	16 (0.2%)	1983 (21%)	209 (2.3%)	9256

The tables show that unblocking the filter is an infrequent event. This is important, as repeated unblocking reduces the upper bound and would ultimately force the SQP solver to follow the constraints very closely, resulting in small steps and slow convergence.

We have also run filter SQP with the NW/SE corner heuristics switched off. These experiments indicate that the NW/SE corner rule does not slow down the solution. Only for 15 problems did we observe a significant change in the number of iterations with the overall bias in favour of the NW/SE corner rule.

Another important question is how much storage is needed for the filter. In our experiments we used an upper bound of 50 on the length of the filter. However, 227 out of the total of 247 small and large test problems were solved with filter sizes less than 10. The largest filter size used in all problems was 43. This is a very modest storage requirement especially when considering the size of problems that are being solved.

Table 4. Problem characteristics (1)

Problem Characteristics																					
#	PROBLEM	n	m	m_{nln}	k	#	PROBLEM	n	m	m_{nln}	k	#	PROBLEM	n	m	m_{nln}	k
1	AIRPORT	84	42	42	42	2	CORE1	65	59	24		3	DECONVC	61	1	1	35
4	DISC2	29	23	23	9	5	DISCS	36	66	66		6	DNIEPER	61	24	24	
7	HATFLDG	25	25	25		8	HS99EXP	31	21	21	2	9	HYDCAR20	99	99	99	
10	HYDCAR6	29	29	29		11	LAKES	90	78	18	12	12	LAUNCH	25	28	10	
13	MESH	41	48	20		14	METHANB8	31	31	31		15	METHANL8	31	31	31	
16	MRIBASIS	36	55	11		17	NET1	48	57	20		18	OPTCNTRL	32	20	10	
19	ORTHREGB	27	6	6	5	20	PRODPL0	60	29	4		21	PRODPL1	60	29	4	
22	SWOPF	83	92	49	2	23	BINSTAR1	257	250	250		24	BINSTAR2	157	150	150	
25	BRIDGEND	2734	2727	1423		26	BRITGAS	450	360	360		27	CORE2	157	134	60	2
28	CRESC100	6	200	200	1	29	CRESC132	6	2654	2654		30	CRESC50	6	100	100	
31	ELATTAR	7	102	102		32	GROUPING	100	125	100		33	HAIFAM	99	150	150	
34	HELSBY	1408	1399	741	1	35	LEAKNET	156	153	80		36	READING6	102	50	50	8
37	READING7	1002	500	500		38	ROTDISC	905	1081	360		39	SMMPSF	720	263	11	
40	SSEBNLN	194	96	24		41	ZAMB2-10	270	96	96	15	42	ZAMB2-11	270	96	96	59
43	ZAMB2-8	138	48	48	30	44	ZAMB2-9	138	48	48	8	45	ARGTRIG	100	100	100	
46	ARTIF	1002	1000	1000		47	BDVALUE	1002	1000	1000		48	BRATU2D	1024	900	900	
49	BRATU2DT	1024	900	900	1	50	BRATU3D	1000	512	512		51	BROYDN3D	1000	1000	1000	
52	BROYDNBD	1000	1000	1000	1	53	CATENA	501	166	166	165	54	CATENARY	501	166	166	327
55	CBRATU2D	1058	882	882		56	CBRATU3D	686	250	250		57	CHANDHEQ	100	100	100	
58	CHEMRCTA	1000	1000	996		59	CHEMRCTB	1000	1000	998		60	CLNLBEAM	1503	1000	500	
61	CORKSCRW	906	700	100	1	62	COSHFUN	61	20	20		63	DIXCHLNV	100	50	50	49
64	DRCAVTY1	1225	961	961		65	DRCAVTY2	1225	961	961		66	DRCAVTY3	1225	961	961	3
67	DRUGDIS	3004	2000	2000		68	DRUGDISE	63	50	50		69	DTOC1NA	2998	1996	1996	54
70	DTOC1NB	2998	1996	1996	42	71	DTOC1NC	2998	1996	1996	998	72	DTOC1ND	2998	1996	1996	50
73	DTOC2	2998	1996	1996	98	74	DTOC4	2999	1998	999	999	75	DTOC5	1999	999	999	998
76	DTOC6	2001	1000	1000	1000	77	EG3	1001	2000	2000		78	EIGENA	110	110	110	
79	EIGENA2	110	55	55		80	EIGENACO	110	55	55		81	EIGENB	110	110	110	

Table 5. Problem characteristics (2)

Problem Characteristics																	
#	PROBLEM	n	m	m_{nln}	k	#	PROBLEM	n	m	m_{nln}	k	#	PROBLEM	n	m	m_{nln}	k
82	EIGENB2	110	55	55	52	83	EIGENBCO	110	55	55		84	EIGENC	462	462	462	
85	EIGENC2	462	231	231	214	86	EIGENCCO	462	231	231	228	87	EIGMAXA	11	11	11	
88	EIGMAXB	11	11	11		89	EIGMAXC	22	22	22		90	EIGMINA	101	101	101	
91	EIGMINB	101	101	101		92	EIGMINC	22	22	22		93	FLOSP2HH	1323	1243	361	
94	FLOSP2HL	1323	1243	361		95	FLOSP2HM	1323	1243	361		96	FLOSP2TH	1323	1243	361	1
97	FLOSP2TL	1323	1243	361		98	FLOSP2TM	1323	1243	361		99	GAUSSELM	506	1135	385	
100	GILBERT	1000	1	1	999	101	HADAMARD	401	1010	210		102	HANGING	1800	1150	1150	941
103	HET-Z	2	1002	1002		104	HVYCRASH	2004	1500	1500	751	105	INTEGREQ	102	100	100	
106	JUNKTURN	1010	700	700	294	107	LCH	600	1	1	198	108	LUBRIF	751	500	249	1
109	MADSSCHJ	201	398	398	1	110	MANNE	1095	730	365		111	MINC44	1113	1032	1013	71
112	MINPERM	583	520	502	64	113	MSQRTA	529	529	529		114	MSQRTB	529	529	529	
115	NGONE	100	1273	1225	1	116	OET2	3	1002	1002		117	OET4	4	1002	1002	
118	OET5	5	1002	1002	1	119	OET6	5	1002	1002		120	OET7	7	1002	1002	
121	OPTCDEG2	1202	800	400	8	122	OPTCDEG3	1202	800	400	49	123	OPTCTRL3	1202	800	400	399
124	OPTCTRL6	122	80	40	39	125	OPTMASS	1210	1005	201		126	ORTHRDS2	203	100	100	103
127	ORTHREGA	2053	1024	1024	995	128	ORTHREGC	1005	500	500	503	139	ORTHREGD	1003	500	500	503
140	ORTHREGE	36	20	20	16	131	ORTHREGF	1205	400	400	798	132	ORTHRGDM	2003	1000	1000	
133	ORTHRGDS	1003	500	500	490	134	POROUS1	1024	900	900		135	POROUS2	1024	900	900	
136	QR3D	610	610	610		137	QR3DBD	457	610	610		138	READING1	2002	1000	1000	
149	READING3	2002	1001	1000		150	READING4	1001	1000	1000	4	141	READING5	1001	1000	1000	
142	READING9	2002	1000	1000		143	SEMICON1	502	500	500		144	SEMICON2	502	500	500	
145	SINROSNB	1000	999	999	101	146	SPMSQRT	1000	1664	1664		147	SREADIN3	2002	1001	1000	
148	SSNLBEAM	3003	2000	1000	959	159	SVANBERG	500	500	500	53	160	TFI1	3	501	501	2
151	TRAINF	808	402	201		152	TRAINH	808	402	201	17	153	UBH5	1010	700	100	105
154	VANDERM1	100	199	100		155	VANDERM2	100	199	100		156	VANDERM3	100	199	100	
157	VANDERM4	9	17	9		158	ZAMB2	1326	480	480	75	159	ZIGZAG	604	500	100	

Table 6. Detailed results for fixed sized test problems

#	PROBLEM	Filter SQP								LANCELOT	
		# QPs	# inf	SOCS	# f	# c	# g	ρ	iter	CPU	iter	CPU
1	AIRPORT	11			12	12	12	10.0	11	4.2	62	5.1
2	CORE1	8	9		8	24	16	2560.	15	1.1	996	33.7
3	DECONVC	118		96	211	211	68	3.305	118	22.8	40	2.1
4	DISC2	58	120	61	58	263	98	0.6257	159	6.3	72	2.0
5	DISCS	2	16	6	1	40	17	40.0	17	2.5	1423	267.6
6	DNIEPER	2			3	3	3	10.0	2	0.2	66	2.0
7	HATFLDG	14	14	7	15	30	15	7.133	23	0.7	16	0.3
8	HS99EXP	15	17		16	47	31	1.342E+9	31	0.8	5000	51.4
9	HYDCAR20	9	4	1	11	16	12	80.0	12	3.5	5000	6152.3
10	HYDCAR6	4	4		5	10	7	40.0	7	0.5	1348	154.2
11	LAKES	10	13	2	11	37	22	1.638E5	21	4.2	3140	78.4
12	LAUNCH	7			8	8	8	320.0	7	0.2	5000	772.9
13	MESH	36			36	37	37	3.436E+11	35	0.8	5000	99.9
14	METHANB8	2			3	3	3	10.0	2	0.1	230	26.3
15	METHANL8	4	2		5	6	5	20.0	5	0.3	673	68.3
16	MRIBASIS	3			3	4	4	10.0	2	0.2	120	12.8
17	NET1	6	7		6	18	12	320.0	11	2.0	84	3.3
18	OPTCNTRL	4			4	5	5	10.0	3	0.1	23	0.8
19	ORTHREGB	1			2	2	2	10.0	1	0.1	104	1.7
20	PRODPL0	11	2	2	13	15	10	2.667	11	0.5	34	2.1
21	PRODPL1	9	2	2	10	12	8	2.667	9	0.4	66	3.2
22	SWOPF	5			5	6	6	10.0	4	0.7	277	34.0
23	BINSTAR1	15	2		16	17	16	10.0	16	12.0	3063	782.9
24	BINSTAR2	3			4	4	4	10.0	3	1.4	2360	437.4
25	BRIDGEND	10	6		10	20	15	2560.	14	1738.0	4104	745171.0
26	BRITGAS	55	4	35	80	82	33	0.3125	57	376.7	99	63.5
27	CORE2	37	8	23	58	72	28	23.32	43	22.1	1065	168.8
28	CRESC100	378	9	629	990	996	242	1.514	383	54.5	5000	336.1
29	CRESC132	295	3	187	478	481	158	20.0	297	363.3	5000	3619.5
30	CRESC50	523	5	424	939	943	301	20.0	526	27.0	5000	197.9
31	ELATTAR	74	16	36	94	126	49	5.0	88	7.3	370	18.0
32	GROUPING	1			2	2	2	10.0	1	0.2	47	4.5
33	HAIFAM	41		21	63	63	24	20.0	41	28.4	5000	3734.2
34	HELSBY	12		1	13	14	13	640.0	11	325.3	827	33809.4
35	LEAKNET	7		1	8	9	8	40.0	6	2.2	483	278.7
36	READING6	8			9	9	9	10.0	8	1.5	5000	74.2
37	READING7	8		2	11	11	9	10.0	8	410.2	5000	2623.0
38	ROTDISC	16	4	3	19	25	17	1738.	18	367.0	5000	381236.0
39	SMMPSF	24	7	7	30	39	22	432.5	28	126.2	5000	18063.5
40	SSEBNLN	9			9	10	10	2560.	8	1.6	66	8.5

Table 7. Detailed results for fixed (41–45) & variable sized test problems (part 1)

#	PROBLEM	Filter SQP								LANCELOT	
		# QPs	# inf	SOCS	# f	# c	# g	ρ	iter	CPU	iter	CPU
41	ZAMB2-10	5			5	6	6	20.0	4	3.3	37	20.1
42	ZAMB2-11	5			5	6	6	40.0	4	3.7	53	14.7
43	ZAMB2-8	5		1	6	7	6	20.0	4	1.0	56	6.0
44	ZAMB2-9	4			4	5	5	20.0	3	0.8	34	6.6
45	ARGTRIG	3			4	4	4	10.0	3	3.8	13	8.1
46	ARTIF	9	5	2	9	19	12	20.0	12	152.8	33	6.9
47	BDVALUE	1			2	2	2	10.0	1	9.1		3.1
48	BRATU2D	3			4	4	4	10.0	3	145.0	2	7.4
49	BRATU2DT	9	2	2	10	14	9	5.340E-2	11	415.2	5	16.3
50	BRATU3D	3			4	4	4	10.0	3	36.2	3	3.4
51	BROYDN3D	4			5	5	5	10.0	4	48.3	5	3.6
52	BROYDNBD	1	52	19	1	119	41	1.250	53	1021.0	16	7.0
53	CATENA	125	44	68	127	165	73	10.0	151	479.5	151	14.2
54	CATENARY	133	906	502	132	2245	501	1.250	1000	2954.0	810	193.9
55	CBRATU2D	3			4	4	4	10.0	3	132.9	3	6.8
56	CBRATU3D	3			4	4	4	10.0	3	6.4	4	2.0
57	CHANDHEQ	11			12	12	12	10.0	11	14.6	13	11.0
58	CHEMRCTA	4	2		4	6	5	10.0	4	82.7	765	7337.2
59	CHEMRCTB	4	2		4	6	5	10.0	4	65.9	122	18.8
60	CLNLBEAM	2			2	3	3	10.0	1	33.5	350	632.2
61	CORKSCRW	13	2	2	14	16	12	2.506	13	137.6	178	2899.4
62	COSHFUN	999	2	1412	2408	2409	515	8.389E+7	1000	27.2	126	5.9
63	DIXCHLNV	15		7	23	23	16	10.0	15	8.2	5000	670.6
64	DRCAVTY1	9	12	7	10	36	16	6.108E-2	19	1928.0	54	930.2
65	DRCAVTY2	12	30	17	14	74	25	3.649E-2	39	2911.0	96	1126.6
66	DRCAVTY3	5	122	68	5	313	71	1.096E-2	126	21860.0	194	1385.0
67	DRUGDIS	5	5		6	13	9	160.0	9	1291.0	1125	1189.4
68	DRUGDISE	5	17		4	36	20	4.096E+4	20	1.5	5000	1683.9
69	DTOC1NA	8			9	9	9	10.0	8	1751.0	9	41.9
70	DTOC1NB	8			9	9	9	10.0	8	1875.0	12	50.5
71	DTOC1NC	4			5	5	5	10.0	4	1022.0	13	54.6
72	DTOC1ND	9		2	12	12	8	0.4892	9	2219.0	24	63.0
73	DTOC2	10		5	16	16	7	1.340	10	2238.0	33	116.2
74	DTOC4	4			4	5	5	10.0	3	834.1	15	59.2
75	DTOC5	3			4	4	4	10.0	3	494.4	24	37.7
76	DTOC6	9			10	10	10	10.0	9	1263.0	139	271.3
77	EG3	20		3	24	24	18	0.2143	20	84.2	96	1554.8
78	EIGENA	1			2	2	2	10.0	1	0.3	16	0.9
79	EIGENA2	2			3	3	3	10.0	2	0.3	38	2.3
80	EIGENACO	2			3	3	3	10.0	2	1.1	24	2.8

Table 8. Detailed results for variable sized test problems (part 2)

#	PROBLEM	Filter SQP								LANCELOT	
		# QPs	# inf	SOCS	# f	# c	# g	ρ	iter	CPU	iter	CPU
81	EIGENB	1	7		1	15	8	10.0	8	1.5	173	29.2
82	EIGENB2	45		28	66	66	27	0.3125	45	37.1	45	4.4
83	EIGENBCO	2			3	3	3	10.0	2	1.1	124	17.7
84	EIGENC	1	80	30	1	145	37	3.815E-5	81	18850.0	618	922.8
85	EIGENC2	50	14	33	63	86	37	2.5	61	3608.0	582	290.6
86	EIGENCCO	87	7	44	108	113	49	1.250	91	12010.0	315	395.6
87	EIGMAXA	2	2		3	4	3	10.0	3	0.1	6	0.2
88	EIGMAXB	7	2	1	9	10	8	10.0	8	0.1	19	0.4
89	EIGMAXC	1	5		1	11	6	10.0	6	0.2	9	0.4
90	EIGMINA	1	19		1	39	20	10.0	20	2.3	7	1.1
91	EIGMINB	7			8	8	8	10.0	7	0.8	577	27.9
92	EIGMINC	1	5		1	11	6	10.0	6	0.2	10	0.5
93	FLOSP2HH	1			1	2	2	10.0		19.2	1469	61251.4
94	FLOSP2HL	1			1	2	2	10.0		19.2	5000	453376.8
95	FLOSP2HM	1			1	2	2	10.0		19.0	5000	277375.2
96	FLOSP2TH	2	214	116	1	534	140	2.659E+4	215	31310.0	432	16337.8
97	FLOSP2TL	5	4		5	11	8	80.0	7	1004.0	5000	456646.6
98	FLOSP2TM	14	22	5	14	53	27	1.024E+4	32	3504.0	5000	268839.2
99	GAUSSELM	45	2	22	62	63	29	1.843	46	1101.0	334	1295.8
100	GILBERT	19		1	21	21	20	20.0	19	2284.0	30	20.7
101	HADAMARD	2	1		1	4	3	10.0	2	1.4	1503	11364.6
102	HANGING	90	16	57	122	138	62	2.5	99	15350.0	1438	4825.8
103	HET-Z	1			2	2	2	10.0	1	0.2	166	59.2
104	HYYCRASH	2	54	20	1	130	42	2.048E+4	55	4816.0	5000	25110.6
105	INTEGREQ	2			3	3	3	10.0	2	2.5	3	2.1
106	JUNKTURN	56	11	30	70	81	32	0.1562	62	2926.0	106	70.5
107	LCH	1			2	2	2	20.0	1	3.5	35	10.5
108	LUBRIF	2	37	18	1	93	25	5.0	38	3208.0	5000	35361.4
109	MADSSCHJ	82	6	38	85	88	45	80.0	85	1316.0	1784	18204.9
110	MANNE	2			2	3	3	10.0	1	9.6	11	28.2
111	MINC44	40	2	23	62	64	27	0.1875	40	1176.0	89	142.1
112	MINPERM	24	2	13	36	38	17	0.2500	24	300.9	255	210.7
113	MSQRTA	8	113	71	10	318	72	1.250	120	32250.0	46	454.7
114	MSQRTB	6	112	70	7	309	72	2.5	117	26030.0	44	344.6
115	NGONE	9		5	10	11	6	1.250	8	2.3	2153	3128.5
116	OET2	5			6	6	6	10.0	5	3.6	69	55.3
117	OET4	5			6	6	6	10.0	5	7.6	53	88.5
118	OET5	21		26	48	48	14	0.1692	21	36.8	2264	841.4
119	OET6	52	2	36	81	82	32	1.250	53	86.9	167	204.1
120	OET7	91		71	160	160	55	0.5107	91	176.8	682	603.0

Table 9. Detailed results for variable sized test problems (part 3)

#	PROBLEM	Filter SQP								LANCELOT	
		# QPs	# inf	SOCS	# f	# c	# g	ρ	iter	CPU	iter	CPU
121	OPTCDEG2	4			4	5	5	10.0	3	65.9	138	143.0
122	OPTCDEG3	9	7		9	21	15	640.0	14	204.3	81	94.9
123	OPTCTRL3	32	2	28	60	62	23	40.0	32	1858.0	89	38.3
124	OPTCTRL6	5			5	6	6	10.0	4	1.6	61	3.8
125	OPTMASS	10		4	10	11	7	1.250	9	80.1	5000	986.2
126	ORTHRDS2	92	2	127	196	197	59	2.5	93	130.9	6 539	40.1
127	ORTHREGA	26	4	16	41	43	20	2.176	28	5973.0	199	130.6
128	ORTHREGC	34		20	46	46	22	0.1562	34	1288.0	45	22.2
129	ORTHREGD	42		12	47	47	33	2.5	42	1573.0	289	79.4
130	ORTHREGE	46	8	26	66	70	30	10.0	50	3.0	1033	32.1
131	ORTHREGF	48	10	25	59	64	30	0.6250	53	4033.0	231	51.8
132	ORTHRGDM	132	24	150	240	260	81	9.766E-2	146	92450.0	185	108.2
133	ORTHRGDS	37	2	18	45	46	24	5.0	38	1403.0	412	208.1
134	POROUS1	15	4	4	15	18	12	2.5	16	706.3	30	45.7
135	POROUS2	8	8	5	8	25	10	2.5	14	766.0	32	21.9
136	QR3D	6	2	1	7	8	7	20.0	7	138.9	240	1237.6
137	QR3DBD	1	38	19	1	88	21	0.	39	1085.0	245	168.9
138	READING1	9		3	12	13	8	2.0	8	329.9	5000	17474.1
139	READING3	8		4	13	13	7	1.0	8	294.6	5000	9892.3
140	READING4	5			5	6	6	10.0	4	76.9	1787	63432.3
141	READING5	6			6	7	7	10.0	5	60.0	150	466.3
142	READING9	3			3	4	4	10.0	2	90.4	5000	548.8
143	SEMICON1	14	181	49	15	397	111	74.71	194	442.8	760	44.7
144	SEMICON2	8	32	15	9	79	25	20.0	39	101.9	117	8.5
145	SINROSNB	996	7	42	1011	1016	517	1.491E-2	1000	10710.0	161	44.5
146	SPMSQRT	1	23	9	1	52	17	0.5067	24	401.9	21	9.1
147	SREADIN3	7			8	8	8	10.0	7	253.8	35	1277.7
148	SSNLBEAM	31	2	2	26	27	24	5.467	24	3739.0	242	605.0
149	SVANBERG	14	2	11	26	27	12	0.2769	15	389.9	84	115.6
150	TFI1	15			16	16	12	3.787	15	1.0	105	23.2
151	TRAINF	5	6		5	15	10	160.0	9	37.5	54	44.1
152	TRAINH	6	6		6	16	11	160.0	10	48.4	560	202.7
153	UBH5	80	940	515	87	2351	513	0.6250	1000	82230.0	88	325.9
154	VANDERM1	1	13		1	27	14	10.0	14	107.3	22	20.0
155	VANDERM2	1	15		1	31	16	10.0	16	122.2	22	20.1
156	VANDERM3	1	21	1	1	44	22	10.0	22	153.4	24	21.5
157	VANDERM4	1	24	3	1	51	22	5.0	25	0.7	806	10.3
158	ZAMB2	7			7	8	8	20.0	6	118.2	43	222.9
159	ZIGZAG	15	4	9	22	25	14	4.304	16	132.0	93	321.2

Acknowledgements. We are grateful to a referee and the associate editor for their helpful comments on an earlier draft of the paper. Support through EPSRC grant number GR/K51204 is gratefully acknowledged.

References

1. Chamberlain, R.M., Powell, M.J.D., Lemarechal, C., Petersen, H.C. (1982): The watchdog technique for forcing convergence in algorithms for constrained optimization. Math. Program. Study **16**, 1–17
2. Conn, A.R., Gould, N.I.M., Toint, Ph.L. (1992): LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A). Springer, Heidelberg, New York
3. Conn, A.R., Gould, N.I.M., Toint, Ph.L. (1994): A note on exploiting structure when using slack variables. Math. Program. **67**(1), 89–97
4. Conn, A.R., Gould, N.I.M., Toint, Ph.L. (1997): Methods for nonlinear constraints in optimization calculations. In: Duff, I.S., Watson, G.A. (eds.) The State of the Art in Numerical Analysis. Clarendon Press, Oxford, pp. 363–390
5. Conn, A.R., Gould, N.I.M., Toint, Ph.L. (1996): Numerical experiments with the lancelot package (Release A) for large-scale nonlinear optimization. Math. Program. **73**(1), 73–110
6. Fletcher, R. (1981): Numerical experiments with an l_1 exact penalty function method. In: Meyer, R.R., Mangasarian, O.L., Robinson, S.M. (eds.) Nonlinear Programming 4. Academic Press, New York
7. Fletcher, R. (1993): Resolving degeneracy in quadratic programming. Ann. Oper. Res. **47**, 307–334
8. Fletcher, R. (1987): Practical Methods of Optimization, 2nd edition. John Wiley, Chichester
9. Fletcher, R., Leyffer, S. (1998): User manual for filterSQP. Numerical Analysis Report NA/181, Dundee University, April 1998
10. Fletcher, R., Leyffer, S., Toint, Ph.L. (1998): On the global convergence of Filter-SQP algorithm. Numerical Analysis Report NA/97, University of Dundee, UK, November 2000. Submitted to SIOPT
11. Fourer, R., Gay, D.M., Kernighan, B.W. (1993): AMPL: A modelling Language for Mathematical Programming. boyd & fraser publishing company, Massachusetts
12. Han, S.P. (1977): A globally convergent method for nonlinear programming. J. Optim. Theory Appl. **22**(3), 297–309
13. Powell, M.J.D. (1978): A fast algorithm for nonlinearly constrained optimization calculations. In: Watson, G.A. (ed.) Numerical Analysis, 1977, pp. 144–157, Springer, Berlin
14. Wilson, R.B. (1963): A simplicial algorithm for concave programming. PhD thesis, Harvard University Graduate School of Business Administration
15. Zoppke-Donaldson, C. (1995): A Tolerance-Tube Approach to Sequential Quadratic Programming with Applications. PhD thesis, Department of Mathematics and Computer Science, University of Dundee, Dundee, Scotland, UK, December 1995