SIAM J. OPTIM.
Vol. 22, No. 3, pp. 772–794

A SEQUENTIAL LINEAR CONSTRAINT PROGRAMMING ALGORITHM FOR NLP*

ROGER FLETCHER†

Abstract. A new method for nonlinear programming (NLP) using sequential linear constraint programming (SLCP) is described. Linear constraint programming (LCP) subproblems are solved by a new code using a recently developed spectral gradient method for minimization. The method requires only first derivatives and avoids having to store and update approximate Hessian or reduced Hessian matrices. Globalization is provided by a trust region filter scheme. Open source production quality software is available. Results on a large selection of CUTEr test problems are presented and discussed and show that the method is reliable and reasonably efficient.

Key words. nonlinear programming, linear constraint programming, Robinson's method, spectral gradient method, filter

AMS subject classifications. 65K10, 90C30, 90C55

DOI. 10.1137/110844362

1. Introduction.

The main aim of this paper is to describe a method to find a local solution $\mathbf{x}^*$ of the nonlinear programming problem

$$\begin{array}{ll} \text{minimize} & f(\mathbf{x}) \qquad \mathbf{x} \in \mathbb{R}^n \\ \text{subject to} & \mathbf{l} \le \begin{bmatrix} \mathbf{x} \\ \mathbf{c}(\mathbf{x}) \end{bmatrix} \le \mathbf{u} \end{array} \tag{1.1}$$

using sequential linear constraint programming (SLCP) as the underlying mechanism to provide local convergence. Many aspects of our approach have been seen before in various contexts, but it will be argued that the particular combination seen here, which we refer to as `filterSD`, has much to recommend it.

The following main design attributes of `filterSD` have been implemented in Fortran 77:

- no second derivatives (gradient method),
- no approximate Hessian or reduced Hessian matrices,
- SLCP (Robinson's method [11]) for rapid local convergence,
- trust region filter globalization,
- projection steps to speed nonlocal convergence,
- a limited memory spectral gradient method in the LCP null space,
- linear programming (LP) subproblems to set or reset multiplier estimates,
- a feasibility restoration phase when LP is infeasible.

A previous code `filterSQP` (Fletcher and Leyffer [7]) has proved to be very reliable, and many features of this approach are retained, particularly in regard to the use of a filter and trust region approach to promoting global convergence. Filter methods provide a quite distinct alternative to other well established methods, such as penalty and barrier methods, and avoid, for example, the need to choose penalty parameters and to deal with certain issues relating to the possibility of ill-conditioning.

*Received by the editors August 12, 2011; accepted for publication (in revised form) May 11, 2012; published electronically July 10, 2012.
http://www.siam.org/journals/siopt/22-3/84436.html
†Division of Mathematics, University of Dundee, Dundee DD1 4HN, UK (fletcher@maths.dundee.ac.uk).

`filterSQP` requires second derivatives to be supplied, which is not always convenient for users and can also be inefficient for certain types of problem. However, the main disadvantage of `filterSQP` is the need to store and update factors of a reduced Hessian matrix during the quadratic programming (QP) calculation. This can become very expensive in both storage and time if the dimension of the null space becomes large. In going from SQP to SLCP we avoid these difficulties, while retaining the property of local and second order convergence, subject to solving the LCP subproblems to sufficient accuracy. We also retain the benefits of using warm starts to initialize the active set in the LCP solver and the possibility of handling degeneracy in a reliable and provably finite manner. The downside is the likelihood of having to make more function and gradient calls, particularly when the dimension of the null space is large.

The need to provide second derivative information in the LCP solver is addressed by using a new *limited memory spectral gradient method* (see Fletcher [6]). This circumvents issues such as how to store and update approximate Hessian or reduced Hessian matrices and how to deal with nonpositive curvature. Yet the method in [6] has proved to be effective in solving large dimension unconstrained optimization problems with as many as 10^6 variables.

Another feature which we see in `filterSD` is the occasional use of an LP trust region subproblem to set or reset multiplier estimates. Such an idea is seen in the so-called SLP-EQP method (see Fletcher and Sainz de la Maza [9], Chin and Fletcher [2], and Byrd et al. [1]) in which the LP subproblem is used to determine an active set, which defines an equality quadratic programming (EQP) problem, from which a new iterate is calculated. Again there are issues regarding how best to supply Hessian information for the EQP calculation. It is also pointed out in [1] that calls to the LP can become expensive due to frequent changes in the active set of the trust region bounds. In `filterSD`, aside from the initial call of the LP subproblem, multiplier estimates are taken from the previous LCP calculation, and subsequent calls to the LP are only made if the LCP fails to make progress. The motivation for this is that multiplier estimates from failed LCP subproblems are likely to be of little value.

Occasionally it has emerged from numerical experiments that slow progress can result when iterates are close to the feasible region, but the constraint curvature is very severe. This causes the radius of the trust region to become small in order to get filter acceptability. It has been found that the use of additional *projection steps* has been effective in making progress without the need to reduce the trust region. The idea is related to, but differs from, the technique of second order corrections (see, e.g., [3]).

Another idea which we retain from `filterSQP` is the use of *feasibility restoration*. When the LP or LCP subproblem is infeasible, the LCP iteration can no longer proceed, because the idea of reducing the trust region radius cannot be used to restore feasibility. Instead, the code enters a feasibility restoration phase in order to drive the iterates closer to the feasible region. This involves the solution of a *nonlinear feasibility problem*, which is solved by a very similar application of SLCP and projection iterates.

Finally a few remarks about the possibility of global convergence are appropriate. The method has many features in common with the trust region filter SQP method analyzed in [8]. Therefore it might well be possible to make a convergence proof under the assumptions that the LCP problems are always solved exactly and the feasibility restoration scheme terminates in a predictable way. In practice it is necessary to

accept approximate solutions of the LCP and feasibility restoration subproblems, and this is a significant complication. An approach used in other contexts has been to use a forcing sequence of tolerances on the KKT errors converging to zero. My experience is that it is very difficult to make such tolerances work well in practice. The approach used in this paper of setting an upper limit on the number of gradient calls for an LCP problem is robust and seems to work well in practice, but it does not lend itself to making a convergence proof. In fact it has not yet been proved that the LCP solver converges for a regular QP problem, due to the possibility of taking steepest descent steps, although it seems highly unlikely that it would fail to do so.

2. SLCP (Robinson's method with a trust region). Robinson [11] describes a local method for solving (1.1) in which the iterates are $\mathbf{x}^k$ for the variables, $\boldsymbol{\pi}^k$ and $\boldsymbol{\lambda}^k$ for the multipliers of the simple bounds and general constraints, respectively. The convention is that a positive multiplier indicates that the lower bound is active, and a negative multiplier indicates the upper bound. For our purposes we include a trust region constraint so that a sequence of LCP subproblems of the following form is solved:

(2.1)
$$\begin{array}{lll} \text{minimize} & \phi(\mathbf{x}) := f(\mathbf{x}) - \boldsymbol{\lambda}^{kT}\Delta(\mathbf{x},\, \mathbf{x}^k) & \text{nonquadratic objective function} \\ \text{subject to} & \mathbf{l} \le \begin{bmatrix} \mathbf{x} \\ \mathbf{c}^k + A^{kT}(\mathbf{x}-\mathbf{x}^k) \end{bmatrix} \le \mathbf{u} & \text{constraints linearized about } \mathbf{x}^k, \\ & ||\mathbf{x}-\mathbf{x}^k||_\infty \le \rho & \text{trust region constraint,} \end{array}$$

where $\mathbf{c}^k = \mathbf{c}(\mathbf{x}^k)$, $A^k = \nabla \mathbf{c}^T(\mathbf{x}^k)$, and

$$\Delta(\mathbf{x},\, \mathbf{x}^k) = \mathbf{c}(\mathbf{x}) - \mathbf{c}^k - A^{kT}(\mathbf{x}-\mathbf{x}^k) \tag{2.2}$$

denotes the *deviation from linearity*. (In practice (2.1) would be formulated in terms of the displacement $\mathbf{d} = \mathbf{x} - \mathbf{x}^k$ for better accuracy.) The solution of (2.1) determines the next iterate $\mathbf{x}^{k+1}$, and the multipliers of the constraints at the solution to (2.1) become the multiplier iterates $\boldsymbol{\pi}^{k+1}$, $\boldsymbol{\lambda}^{k+1}$ for the next iteration.

This approach is attractive when only first derivatives are available, as it enables us to dispense with the need to specify an exact or approximate Lagrangian Hessian matrix at $\mathbf{x}^k$, $\boldsymbol{\pi}^k$, $\boldsymbol{\lambda}^k$, as the SQP method would require. By solving the LCP subproblem with a spectral gradient method, we also avoid having to make decisions about how to store and update an approximate Hessian or reduced Hessian matrix, and how to deal with negative curvature. However, there is also the disadvantage that, in general, (2.1) is not a finite calculation. Thus decisions need to be made about how accurately to solve the subproblem. Also multiple evaluations of $f(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ are made while solving the LCP problem, as against a single evaluation in the SQP method. If these functions are very expensive to evaluate, then SLCP is less likely to be effective.

Yet Robinson's method is very closely related to the SQP method. Indeed, if the functions $f(\mathbf{x})$ and $\mathbf{c}(\mathbf{x})$ are quadratic, then the two methods are identical. The ith element of the vector $\Delta(\mathbf{x},\, \mathbf{x}^k)$ is $\frac{1}{2}\mathbf{d}^T\nabla^2 c_i\mathbf{d}$, so we can collect terms to give $\phi(\mathbf{x}) = f(\mathbf{x}^k) + \mathbf{d}^T\mathbf{g}(\mathbf{x}^k) + \frac{1}{2}\mathbf{d}^T W^k\mathbf{d}$, where $W^k = \nabla^2 f - \sum_i \lambda_i^k \nabla^2 c_i$, which is the objective function in the SQP subproblem. In general, the objective functions in SQP and Robinson's method differ only in terms which are negligible to second order, so both methods share the same local convergence and order of convergence properties, as Robinson [11] indeed proved.

An early and very successful NLP solver is the MINOS code of Murtagh and Saunders [10]. Rather than using a trust region, MINOS uses a modified form of Robinson's method in which a penalty term $\frac{1}{2}\sigma||\Delta(\mathbf{x}, \mathbf{x}^k)||_2^2$ is included in the objective function. The penalty parameter σ may be increased where necessary in an attempt to promote convergence, although a proof of convergence is not given in [10]. We observe that this is not the usual augmented Lagrangian penalty, as the squared term in MINOS is not the squared norm of the constraint residual, and indeed does not make reference to the bounds $\mathbf{l}$ and $\mathbf{u}$. However, in `filterSD` no squared penalty terms are used. The expectation is that the contribution to the Hessian of $\phi(\mathbf{x})$ is likely to make the Hessian more ill-conditioned, and hence slow the rate of convergence of the spectral gradient method.

3. A new LCP solver `glcpd`. A new Fortran 77 code `glcpd` has been written to solve a general LCP problem such as (2.1). In many ways it is a conventional active set method (ASM), although there are some features of interest. A *working set* of $m \leq n$ active constraints is maintained, and columns of a matrix A collect their normal vectors. In addition, $n - m$ *free variables* are present, which are strictly feasible variables, and are used to parametrize the null space defined by the working set constraints. A matrix V collects the corresponding columns of the $n \times n$ unit matrix. Factors of the (nonsingular) matrix $B = [A\ V]$ are maintained and updated when changes to the working set and/or free variables take place.

At the start of an ASM iteration from a feasible point, the multiplier vector $\boldsymbol{\lambda}$ defined by

$$B\boldsymbol{\lambda} = \mathbf{g}$$

is calculated, where $\mathbf{g}$ refers to the current gradient vector (the scope of the notation here is only local to this section). Components of $\boldsymbol{\lambda}$ corresponding to free variables are the *reduced gradients*, and components corresponding to a working set constraint give the slope of an *edge* generated by the relaxation of that constraint (see, e.g., [3]). On any iteration, either the *reduced steepest descent direction* is chosen, in which all the free variables are changed, or else a conventional *steepest edge direction* is chosen, obtained by relaxing one of the working set constraints whose multiplier is out of kilter. The decision is made by choosing the steepest of the possible choices. Steepest edge weights are maintained to enable true slopes to be calculated.

A fairly standard line search is now made. An upper bound on the step length is calculated to maintain feasibility, and a step is determined which improves the objective function and where possible provides a positive curvature estimate. While the working set is unchanged, and reduced steepest descent directions are chosen, step lengths are determined in accordance with a recent limited memory spectral gradient method based on *Ritz values* (see Fletcher [6]). Tests have indicated that this method is preferable to CG and is comparable to l-BFGS for large dimension null spaces. In this method a bundle of $r \leq n - m$ back reduced gradients is stored and is used to calculate r Ritz values. The reciprocals of these values provide step lengths for a sequence of up to r reduced steepest descent steps, referred to as a *sweep*. If a step fails to improve on the objective function value at the start of the sweep, or if the step is infeasible, then a line search is made and the sweep is terminated. New reduced gradients are collected during the sweep and provide Ritz values for the next sweep. The r Ritz values can be considered as best estimates to the $n - m$ eigenvalues of a local reduced Hessian matrix. The method terminates finitely if the objective is quadratic and $r = n - m$.

If a change in the working set takes place, previous Ritz values are still used to provide initial step lengths for a new sweep. If the null space has increased in dimension, these Ritz values still lie in the spectrum of the supposed new reduced Hessian. This may not be the case if the null space dimension is decreased, but either way it is preferable to use these values rather than restarting from an arbitrary single curvature estimate. After the initial sweep, new Ritz values are computed which are appropriate to the Hessian of the new null space.

When a previously inactive constraint becomes active in the line search, a change in the working set is made (except when a working set constraint moves to its opposite bound). If the iteration is a steepest edge iteration, then the new constraint replaces the one being relaxed, and a standard simplex update is made to B. If the iteration is a reduced steepest descent iteration, then the null space dimension is decreased, and the new constraint replaces one of the free variables, chosen by a pivot test.

When the iteration is a steepest edge iteration, and a new constraint does not become active in the line search, then the null space dimension is increased. If the constraint being relaxed is a simple bound, then this becomes a new free variable. Otherwise the constraint being relaxed is replaced by an inactive variable, chosen again by a pivot test, and a simplex update is made to B.

Another feature of interest is that the method has three phases $(0, 1, 2)$. In phase 0 feasibility with respect to the simple bounds $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$ is determined (this is not immediately available in a warm or hot start but is always possible). Phase 1 then goes on to obtain feasibility with respect to the general constraints. This may not always be possible, in which case the code exits with a pareto solution, of the form described in section 8. The iterations in these phases minimize a sum of infeasibilities, subject to maintaining feasibility in any feasible constraints, in a similar way to that described above. A postprocessor `l1sold` to `glcpd` has been written which calculates a best L1 solution to minimizing the general constraint residuals, subject to the simple bounds being satisfied. Finally, if phase 1 finds a feasible point, then phase 2 goes on to find an approximate local solution to the LCP problem as described above.

A complicating feature is that of *degeneracy*, which arises when a step of zero length is taken in the line search. In this case a modification of Wolfe's method [12] is used to resolve the situation. Moreover, degeneracy in iterations of the postprocessor `l1sold` is also a possibility, and a further modification of Wolfe's method has been developed which has some interesting features. These modifications will be described in a future paper which is in preparation.

Other features incorporated into `glcpd` and `l1sold` are cold-warm-hot starts and either sparse or dense linear algebra based on LIU (L-Implicit-U) factors (see Fletcher [4, 5]). `glcpd` is also used in the `filterSD` code to solve the LP and projection subproblems. In fact, these could have been solved by a new LP/QP solver `qlcpd`, which has a similar design to `glcpd`. However, this code was not available when `filterSD` was first written, and it is not straightforward to make the necessary changes. Possibly this may be done in a later release of `filterSD`.

4. The LP subproblem. If the solution $\mathbf{x}$ and multipliers $\boldsymbol{\pi}$, $\boldsymbol{\lambda}$ of the LCP subproblem are accepted, then they become $\mathbf{x}^{k+1}$ and $\boldsymbol{\pi}^{k+1}$, $\boldsymbol{\lambda}^{k+1}$ for the next iteration. However, initially, or if the solution of the LCP problem (possibly followed by a projection step) is not accepted by the filter, then the trust region radius ρ is reduced, and new multiplier estimates are calculated from an LP subproblem.

Thus, any sequence of LCP steps is preceded by the solution of the LP subproblem

(4.1)
$$\begin{array}{lll} \text{minimize} & \mathbf{g}^{kT}(\mathbf{x}-\mathbf{x}^k) & \text{linearized } f(\mathbf{x})-f^k \\ \text{subject to} & \mathbf{l} \le \begin{bmatrix} \mathbf{x} \\ \mathbf{c}^k + A^{kT}(\mathbf{x}-\mathbf{x}^k) \end{bmatrix} \le \mathbf{u} & \text{linearized constraints about } \mathbf{x}^k, \\ & ||\mathbf{x}-\mathbf{x}^k||_\infty \le \rho & \text{trust region constraint,} \end{array}$$

where $\mathbf{g}^k = \nabla f(\mathbf{x}^k)$, in order to set multiplier estimates. If the LP subproblem (or the LCP subproblem) is infeasible, then a *feasibility restoration* calculation is initiated.

5. Filter globalization. The filter technique, for promoting global convergence in an NLP solver, was first introduced in 1996 and was used in the design of the `filterSQP` code (see Fletcher and Leyffer [7]). A proof of global convergence of a modified form of the method is given by Fletcher, Leyffer, and Toint [8]. It is this later formulation that has been adapted for use in the design of `filterSD`.

First, a brief description of the filter concept is given. We define a *constraint violation function* which we shall take to be the L1 sum of general constraint violations

$$h(\mathbf{x}) = \sum_{i=1}^{m} \max(l_{n+i} - c_i(\mathbf{x}),\, 0,\, c_i(\mathbf{x}) - u_{n+i})$$

(the simple bounds on the variables are never violated in `filterSD`). For each point $\mathbf{x}^k$ sampled by the algorithm, the pair of values $(h^k,\, f^k)$ is recorded ($h^k = h(\mathbf{x}^k)$ and $f^k = f(\mathbf{x}^k)$). We say that one pair of values *dominates* any other pair if it is better in both f and h. The basic idea of a filter algorithm is not to accept any pair that is dominated by a pair from a previous iteration. To this end we need only store pairs $(h^k,\, f^k)$ that are not dominated by any other previous pair. This set of pairs is known as the *filter*, denoted by $\mathcal{F}^k$.

To formulate a proof of global convergence requires some additional features. Some notion of *sufficient improvement* is needed; for this we require a new pair $(h,\, f)$ to satisfy *either*

$$h \le \beta h^j \qquad or \qquad f + \gamma h \le f^j \qquad \forall j \in \mathcal{F}^k \tag{5.1}$$

if it is to be accepted, where $\beta < 1$ and $\gamma > 0$ are fixed parameters with β close to 1 and γ close to zero. Also h is required to satisfy an upper bound condition $h \le U$, where U is user supplied.

It was also realized that the filter algorithm could stall if pairs with $h^k = 0$ were added to the filter. For SQP, a modified filter algorithm is proposed in [8], in which an acceptable pair is not always added to the filter. To this end [8] introduces the terminology that a step is an *f-type step* if $\Delta q > \tau$ and an *h-type step* otherwise, where Δq is the reduction in f predicted by the model, and $\tau \ge 0$ is close to zero. (In fact [8] takes $\tau = 0$, but other researchers have defined τ as a function of h^k.)

If $\mathbf{x}^k$ is a feasible point, then $\Delta q > 0$ and an f-type step results. Generally an f-type step is associated with $\mathbf{x}^k$ being close to the feasible region $\mathcal{R}$, and f is predicted to improve, but h might increase, as illustrated in Figure 5.1. Conversely, in an h-type step, h is reduced but f is likely to increase, as in Figure 5.2.

In SLCP we no longer have a quadratic model, but $\phi(\mathbf{x})$ essentially fills that role. Thus we may define Δq by

$$\Delta q = \phi(\mathbf{x}^k) - \phi(\mathbf{x}) = f(\mathbf{x}^k) - \phi(\mathbf{x}), \tag{5.2}$$

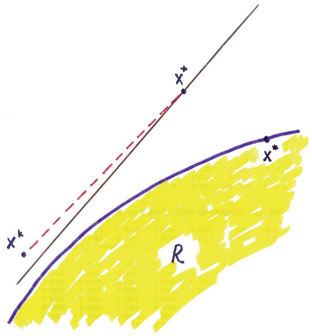

FIG. 5.1. *f-type step.*

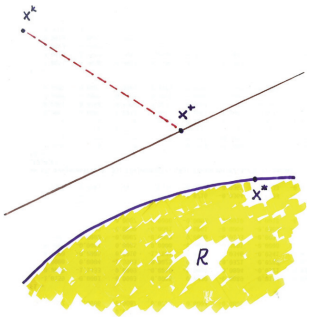

FIG. 5.2. *h-type step.*

where $\mathbf{x}$ is the LCP solution. If the functions f and $\mathbf{c}$ are quadratic, this agrees with the SQP definition. Moreover, because multipliers estimate the marginal change in f on perturbing the constraints, the term $\boldsymbol{\lambda}^{kT}\Delta(\mathbf{x}, \mathbf{x}^k)$ estimates the change in f on projecting $\mathbf{x}$ onto the nonlinear constraint manifold. Hence Δq predicts the change to $f(\mathbf{x}^k)$ of a combined LCP+projection step.

Thus, in `filterSD` we follow the modified filter algorithm given by [8], with the *predicted reduction* Δq given by (5.2), and the *actual reduction* $\Delta f = f^k - f(\mathbf{x})$. The rules for manipulating the filter may be summarized as follows:

- (h^k, f^k) is not stored in the current filter $\mathcal{F}^k$,
- a new point (h, f) must be acceptable to both $\mathcal{F}^k$ and (h^k, f^k) and must satisfy a user supplied upper bound $h \le U$,
- an f-type step must also satisfy $\Delta f \ge \sigma \Delta q$,
- if (h, f) is accepted, (h^k, f^k) is added to $\mathcal{F}^k$ only if the step is an h-type step,

where σ is a preset parameter in $(0, 1)$.

6. Projection steps. Of course, in a basic SQP trust region filter algorithm, if (h, f) fails to satisfy the filter acceptability test, then the trust region radius ρ is reduced, and the SQP step is repeated. However, in certain situations this can lead to very short steps and very slow convergence. It has been noted above that Δq predicts the improvement in f of a combined LCP+projection step. Therefore, if $\Delta q > \tau$, it can be worthwhile to explore the possibility of taking a projection step, to both improve feasibility and reduce f.

Thus, in the context of `filterSD`, let $\mathbf{x}^+$ denote the solution of the LCP subproblem with $h^+ \le U$. If (h^+, f^+) fails the filter test and is an f-type step, then a projection step is carried out. The *projection subproblem* takes the form

(6.1)

$$\begin{array}{lll} \text{minimize} & \tfrac{1}{2}(\mathbf{x}-\mathbf{x}^+)^T(\mathbf{x}-\mathbf{x}^+) & \text{squared step norm from } \mathbf{x}^+ \\ \text{subject to} & \mathbf{l} \le \begin{bmatrix} \mathbf{x} \\ \mathbf{c}^+ + A^{+T}(\mathbf{x}-\mathbf{x}^+) \end{bmatrix} \le \mathbf{u} & \text{constraints linearized about } \mathbf{x}^+, \\ & \|\mathbf{x}-\mathbf{x}^k\|_\infty \le \rho & \text{trust region constraint about } \mathbf{x}^k. \end{array}$$

Let $\mathbf{x}^{++}$ denote the solution of the projection subproblem, and let

$$\Delta f^+ = f^k - (h^+ f^{++} - h^{++} f^+)/(h^+ - h^{++})$$

denote the estimate from a linear model $f = ah+b$ of the reduction in f to be obtained by another projection step which zeros h. Then the following rules are used to decide how to proceed:

If $\mathbf{x}^{++}$ is acceptable set $\mathbf{x}^{k+1} = \mathbf{x}^{++}$ and return to the LP subproblem.

Else if $h^{++} \leq 0.8h^{+}$ **and** $\Delta f^{+} \geq \sigma \Delta q$ repeat the projection step from $\mathbf{x}^{++}$.

Else reduce ρ and repeat the LP step from $\mathbf{x}^{k}$.

The importance of using projection steps is illustrated by the following example. It is taken from the CUTEr problem PFIT1 with $\mathbf{x}^0 = [1.922655, 1.94114, 1.476798]^T$ and $\rho = 1$ during feasibility restoration. PFIT1 has three nonlinear equations, and we solve the NLP problem

$$\begin{array}{ll} \text{minimize} & c_1(\mathbf{x}) \\ \text{subject to} & c_2(\mathbf{x}) = 0, \\ & c_3(\mathbf{x}) = 0, \\ & c_1(\mathbf{x}) \geq 0 \end{array}$$

to drive c_1 down to zero. The initial value of $\mathbf{c}$ is

$$\mathbf{c}^0 = [1.535e{-}2, \quad 7.932e{-}5, \quad 9.393e{-}5]^T.$$

Without projection, after 100 iterations,

$$\mathbf{c}^{100} = [1.294e{-}2, \quad 5.262e{-}5, \quad 6.275e{-}5]^T,$$

and $\rho = 2.85e{-}3$. Thus only a 20% reduction in c_1 has been obtained, and the trust region radius has become very small. However, when projection steps are used, after 12 iterations

$$\mathbf{c}^{12} = [6.520e{-}4, -4.864e{-}6, -5.300e{-}6]^T,$$

and $\rho = 1.514e{-}1$, in which c_1 is close to zero. With projection steps in place, `filterSD` solves PFIT1 in 29 iterations, with $h^{29} \sim 10^{-12}$.

The projection calculation is related to, but different from, the Second Order Correction (SOC) method (see, e.g., [3]). The SOC method makes a perturbation to the constraint residuals so that the solution of the LCP subproblem about $\mathbf{x}^k$ eliminates second order constraint terms. However, it uses the Jacobian at $\mathbf{x}^k$, which can be a disadvantage when the constraints are highly nonlinear and the Jacobian is changing rapidly. Also, in an SLCP context, it is not a finite calculation and requires a sequence of nonlinear function evaluations, which seems undesirable. One might carry out a projection using the step $-Y\mathbf{c}^{+}$ (see [3, p. 318] for notation) but this does not change any of the free variables. By contrast, the projection step (6.1) makes use of the Jacobian evaluated at $\mathbf{x}^{+}$ and gives a unique step which does not preclude changes to the free variables. The downside is that the SOC solution can be calculated by a hot start using existing factors of B at $\mathbf{x}^k$, whereas (6.1) requires factors of B at $\mathbf{x}^{+}$ to be evaluated. Maybe the use of a projection step is inefficient for some problems, but for problems such as PFIT1 above, it is essential to enable solutions to be obtained in a reasonable time.

7. Phase 2 summary. At this point it is convenient to summarize how the various subproblems are linked together. This is done in the following boxed display. Phase 2 refers to the case where `filterSD` is not in feasibility restoration. Some further clarification is provided by the following comments. The rules for changing

the trust region radius ρ are fairly standard and have not been given a great deal of thought. When no acceptable point results from an LCP or projection step, the trust region radius is reduced by $\rho = \alpha||\mathbf{d}||_\infty$, where $\mathbf{d}$ is the LCP step and $\alpha \in [0.1, 0.5]$. Mostly α is obtained by chopping $\frac{1}{2}h^k/h^+$ into this interval, except where $\alpha = 0.5$ is specified. If the trust region constraint in the LCP problem is active, and a successful step results, then the trust region radius is doubled.

The algorithm differs from a regular SQP algorithm in that an exact solution to the LCP subproblem cannot be expected (except when f and $\mathbf{c}$ are quadratic and the null space dimension is small). How best to terminate the subproblem is a nontrivial issue which is discussed in more detail in section 9. Termination of the outer SLCP iteration is implemented by the user supplying a tolerance `htol`, and the iteration terminates when both $h^k \leq$ `htol` and $||\mathbf{d}||_\infty \leq$ `htol`. (Note that it is not sufficient just to test $h^k = 0$; it may be that $\mathbf{x}^k$ is strictly feasible with respect to an inequality constraint that should be active).

1. `Solve LP subproblem`
   no solution add $(h^k,\ f^k)$ into $\mathcal{F}$; enter feasibility restoration
   otherwise collect multipliers $\boldsymbol{\pi}^k, \boldsymbol{\lambda}^k$; goto step 2
2. `Solve LCP subproblem`
   no solution add $(h^k,\ f^k)$ into $\mathcal{F}$; enter feasibility restoration
   solution $\mathbf{x}^+$ **has** $h^+ > U$ reduce ρ; goto step 1
   solution fails filter test
      $h^+ = 0$ **or** $\Delta q \leq \tau$ reduce ρ; goto step 1
      otherwise goto step 3
   otherwise goto step 4
3. `Solve projection subproblem`
   no solution let $\rho = \rho/2$; goto step 1
   solution $\mathbf{x}^{++}$ **has** $h^{++} > U$ reduce ρ; goto step 1
   solution fails filter test
      $h^{++} \leq 0.8h^+$ **and** $\Delta f^+ \geq \sigma \Delta q$ let $\mathbf{x}^+ = \mathbf{x}^{++}$; goto step 3
      otherwise let $\rho = \rho/2$; goto step 1
   otherwise let $\mathbf{x}^+ = \mathbf{x}^{++}$; goto 4
4. `Solution` $\mathbf{x}^+$ `satisfies filter test`
      if $\Delta q \leq \tau$, add $(h^k,\ f^k)$ into $\mathcal{F}$
      if $||\mathbf{d}||_\infty = \rho$, let $\rho = 2\rho$
      let $\mathbf{x}^{k+1} = \mathbf{x}^+$; let $k = k + 1$
      if $\mathbf{x}^+$ arises from a projection step, goto step 1
      test for termination
      collect multipliers $\boldsymbol{\pi}^k, \boldsymbol{\lambda}^k$; goto step 2
5. `End`

The algorithm is initialized by a user supplied approximation $\mathbf{x}^0$ for which $h^0 \leq U$. (The code exits if $h^0 > U$, in which case the user has the choice of re-entering with a different $\mathbf{x}^0$ or resetting U to a larger value.) The filter is initialized to $\mathcal{F} = \emptyset$. The user is also invited to supply estimates of Ritz values for the LCP solver. This allows some (albeit limited) knowledge of the reduced Hessian to be supplied. More usually, the user will just specify a single Ritz value of 1.

8. Feasibility restoration. Feasibility restoration (phase 1) is entered if the LP subproblem or the LCP subproblem is infeasible. The aim of this part of the algorithm is to drive $\mathbf{x}^k$ towards feasibility, and in particular to find $\mathbf{x}^k$ and ρ such that the LP subproblem is feasible, in which case phase 2 can resume. The entire feasibility restoration calculation counts as a single h-type step in phase 2. It may be, however, that the constraints in the NLP problem are inconsistent, in which case the lesser aim of finding a minimizer $\mathbf{x}^*$ of the L1 sum of general constraint infeasibilities $h(\mathbf{x})$ subject to $\mathbf{l} \leq \mathbf{x} \leq \mathbf{u}$ is followed. It is only realistic to find a *locally infeasible solution* to this problem, in which case the entire NLP calculation terminates, even though it may be that there exist feasible points elsewhere.

The feasibility restoration calculation proceeds as follows. A property of the L1 solution of the LP subproblem is that it identifies a set, denoted by J, of *infeasible general constraints*, and a complementary set $J^\perp$ of *feasible general constraints*. A *nonlinear feasibility problem* (NFP) is now solved with the aim of driving the infeasibilities in the J constraints to zero, subject to the $J^\perp$ constraints and the simple bounds being satisfied. When the general constraints may be written as $\mathbf{c}(\mathbf{x}) \leq \mathbf{0}$, the NFP is

$$\begin{array}{lll} \text{minimize} & \sum c_i(\mathbf{x}) & i \in J \\ \text{subject to} & c_i(\mathbf{x}) \leq 0 & i \in J^\perp, \\ & c_i(\mathbf{x}) \geq 0 & i \in J, \\ & \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}. & \end{array} \tag{8.1}$$

Now (8.1) is a regular NLP problem so can be solved (approximately) by an LP/SLCP/projection approach, similar to that used in phase 2, and with its own *phase* 1 *filter*. This filter is initialized to the empty set on entry from phase 2, but is not re-initialized if an LP in the phase 1 calculation is solved. If any J constraints become *active* or *feasible* during the solution of the NFP, then the aim of driving these constraints to zero has been achieved. In this case the LP is re-solved, usually resulting in J being changed. If the LP is feasible ($J = \emptyset$), then phase 2 is re-entered.

The notations $h_J(\mathbf{x})$ and $h_{J^\perp}(\mathbf{x})$ denote the sum of infeasibilities in the J and $J^\perp$ constraints, respectively. In the phase 1 filter the entries are denoted by $(h_{J^\perp}, h_J)$ rather than (h, f) If J changes, it may be that $(h^k_{J^\perp}, h^k_J)$ is no longer acceptable to $\mathcal{F}$. In this case another rule is added:

- if $(h^k_{J^\perp}, h^k_J)$ is not acceptable to $\mathcal{F}$, reduce ρ and re-solve the LP subproblem.

Otherwise the structure of phase 1 follows the same pattern as for phase 2.

9. Practical experience and discussion. The `filterSD` code has been made available from COIN-OR under the open source Eclipse Public License. Release 1.1 of the code has been extensively tested on a wide range of CUTEr NLP test problems of up to 12005 variables, on a MacBook Pro with a 2.53 GHz Intel Core 2 Duo processor and 4 GB 1067 MHz DDR3 memory, in double precision Fortran 77, using the optimized gfortran compiler. A detailed summary of the outcomes is given in the appendix. All the problems are run with a common set of input parameters (see the appendix) which have not been subjected to any "tuning". The column headings in the appendix list first the problem name and number of variables n and general constraints m. Next f^* and h^* refer to the final function and constraint values, except when `ifail` $= 3$, in which case they refer to $h^*_{J^\perp}$ and h^*_J, respectively. Next `rgnorm` and `k` report the values of the L2 reduced gradient norm and the null space dimension, respectively, on exit from the last call of `glcpd`. Next `itn` is the number of LCP (outer)

iterations (the maximum value of the iteration counter k in section 7). Then `#f` and `#g` are the total number of function and gradient calls throughout the entire calculation. A blank entry in the final column indicates that the code is reporting a local solution, else a failure number is given (`ifail` $= 3$ refers to a locally infeasible solution).

Of the 355 problems, 315 report a successful outcome, 23 report local infeasibility, and 17 report some other outcome. Of the 315, all meet the required tolerance on h^* ($1e{-}6$), often by a considerable margin. Second order convergence of h^k to zero is often observed. The tolerance `rgtol` of $1e{-}5$ on `rgnorm` is often, but not always obtained, mainly because there is a limit `mxgr` $= 100$ on the number of gradient calls for each call of `glcpd`. This is needed because the scaling of the reduced gradient is difficult to predict, and a suitable value of `rgtol` is difficult to set. Hence an alternative termination test is used based on the LCP step size $||\mathbf{d}^k||_\infty$. Thus termination from `filterSD` occurs when both h^k and $||\mathbf{d}^k||_\infty$ are less than a tolerance `htol`. Monitoring the output usually indicates that very high accuracy in f^* is obtained. Also, it is always open to the user to re-enter `filterSD` with a larger value of `mxgr` and/or a tighter tolerance on `rgnorm`.

Again for the group of 315 problems, the number of major iterations is often very small, with only 71 cases requiring more than 20 iterations, and only 23 cases requiring more than 50 iterations. This property is often seen with an SQP algorithm, and it is reassuring to see it hold for an SLCP algorithm. On the negative side, unless the dimension of the null space is small, the lack of reduced Hessian information necessitates many more gradient evaluations to solve each LCP problem. Likewise, although Ritz values are passed from one LCP call to the next, these contain much less information than would be provided by the reduced Hessian matrix. Thus the number of function and gradient calls can be much larger than is required by some other codes that provide approximate or exact reduced Hessian information. To some extent this can be alleviated by using the parameter `mxgr` to limit the number of gradient calls made by the LCP solver. Another possibility is to relax the tolerance `rgtol` on the reduced gradient. In practice it is often the case that the cost of a function or gradient evaluation is much less than other costs involved in solving the LCP and LP subproblems, and overall run times seem to be very reasonable, as does reliability.

Next, the 23 problems for which `ifail` $= 3$ is reported are considered. Many are apparently true instances of local infeasibility that have been recognized as such by previous SQP codes, and which stop at a KKT point or demonstrate rapid local convergence to the solution of the NFP (8.1). The ability to converge rapidly is a valuable feature of the feasibility restoration scheme based on identifying partitions J and $J^\perp$. The problems `FLOSP2HH`, `FLOSP2HL`, and `FLOSP2HM` are known to have inconsistent linear constraints and hence are globally infeasible. Only the following problems give an anomalous outcome. COOLHANS and MADSSCHJ have usually been solved by an SQP solver, and DISCS has sometimes been solved, but not always. For `filterSD`, in all three cases, a rapid rate of convergence is observed to what appears to be a well defined point of local infeasibility. The solutions reported by POWELLBS and POWELLSQ are essentially low accuracy solutions of ill-posed problems. The Jacobian of POWELLSQ is singular at the solution, and that of POWELLBS has a condition number of about 10^9. Also the variables in POWELLBS differ in magnitude by a factor of about 10^6. The VANDERM4 problem suffers from difficulties similar to those of VANDERM1 (see below) but shows linear convergence with rate constant about 0.25 to what also appears to be a well defined point of local infeasibility.

For the remaining 17 problems that fail with `ifail` $\neq 3$, various reasons can be seen for the apparent lack of success. Quite a few fail because of insufficient

care in defining the problem or because the default parameters are unsuitable. Thus CSFIT1 fails due to division by a zero variable in the `functions` subroutine. This could easily have been avoided by bounding the variable away from zero. (The same can happen with CSFIT2.) The problem HS99EXP is a reformulation of HS99 in which the number of variables is increased from 7 to 31. Now HS99 is solved very rapidly with all solution variables in the vicinity of 0.5. However, the new variables in HS99EXP take values of up to 10^8. This imbalance in scaling the variables is very unfavorable for a trust region method and makes it very difficult to converge the original variables. The MESH problem is an unbounded NLP, but the situation causes the LCP solver to become unbounded (hence `ifail` $= 11$) while h^k is non-zero (albeit small). The COSHFUN problem fails because the limit of 999 iterations is reached (`ifail` $= 5$). The difficulty arises because values of $\mathbf{x}$ around 10^{10} give rise to an f value of around -10^{10}, with h around 10^4. The difficulty could easily have been avoided by putting more realistic bounds on $\mathbf{x}$ (the solution values are around ± 1) or, as shown in the appendix, by setting `ubd` $= 1$. The HVYCRASH problem also fails with `ifail` $= 11$ because the `functions` subroutine returns a constraint value of about 10^{27} from values of $\mathbf{x}$ of about 10. The difficulty is avoided by resetting the `fmin` parameter to -10^{100}, but it does also suggest that some care in reformulating the problem might be beneficial. Similar remarks hold for the SEMICON1 problem. The ELEC problem fails with `ifail` $= 20$, which on examination is caused by a function value of `+Inf` from a vector $\mathbf{x}^k$ with maximum modulus element $6.3e{-}3$. Again this indicates a failure in formulating the problem in a suitable way. In fact ELEC is quickly solved with `ubd` $= 1$.

VANDERM1 is a problem involving a system of nonlinear equations

$$\sum_{i=1}^{N} x_i^k = a_k \qquad k = 1, 2, \ldots, N,$$

of order $N = 100$, with the a_k chosen so that the solution is $x_i^* = i/N$. Additionally constraints $x_i \geq x_{i-1}$ are included. When $k = 100$, and using $\mathbf{x}^*$, the terms in the sum vary from 10^{-200} when $i = 1$ to 1 when $i = N$. Thus massive cancellation and hence loss of significance occurs. This is a situation which in practice would be handled by the use of orthogonal polynomials rather than power functions. I do not see that failure to solve VANDERM1 has much to inform the design of a practical NLP solver. In VANDERM2 the residuals of the equations are squared, something that creates a null Jacobian at the solution, which is certainly not to be recommended, and my comments apply a fortiori.

Apropos of this, I do not think we should expect an NLP solver to be able to take any problem, however bizarre, and solve it, or find a point of local infeasibility. To this end I circulated a problem that I devised by simplifying the HATFLDF test problem. The shortest form of the problem is just to solve the two equations

$$x_1 x_2 (x_1 - 1) = 0.024,$$
$$x_1 x_2 (x_1^2 - 1) = 0.067$$

from an initial point $\mathbf{x}^0 = (0.98, 1.2)^T$. By dividing the equations, the (unique) solution is easily seen to be when $x_1 = 43/24$. The difficulty occurs because of the pole at $x_1 = 1$. Thus (at least for any descent method on one of the common norms), x_2 has to decrease to $-\infty$ and then return from $+\infty$ to its solution value, which is not usually possible.

There remain a few other problems with `ifail` $\neq 3$ to discuss. Of these, ORTHRDS2 and SPMSQRT have `ifail` $= 6$ (that is, $\rho \leq$ `htol`) and a common explanation. In both cases the outer iterates are converging *linearly* to a local solution and the trust region is inactive. However, when $||\mathbf{d}||_\infty$ is within a factor of 2 of `htol`, the step fails to be accepted by the filter, which is due I think to round-off error in f. Thus the next step reduces ρ and causes `ifail` $= 6$. It is likely that the slow convergence is due to a lack of second order sufficiency at the solution.

Finally, we look at problems that show some adverse aspects of `filterSD`. Both EIGENC and QR3DBD report a nontrivial growth in round-off errors. Maybe there is a message here for the design of the linear algebra. It has been possible to solve EIGENC by relaxing the error tolerance `tol` for recognizing zero. However, the numerical stability of the linear algebra remains a cause for concern, although it is reassuring that this feature has adversely affected so few problems. The iterations in HANGING make steady progress for some time; then during an entry to phase 1, ρ reduces to near zero, and `ifail` $= 6$ is triggered. This indicates some concern with how phase 1 is operating in this instance. HANGING can also be solved by using a smaller value of `ubd`.

A related concern is the time required to solve some of the largest problems. While the GASOIL problem is easily solved in 20 outer iterations and exhibits second order convergence, the time taken is almost 5 hours. The reason is that the LIU factors of the basis matrix require more than $3 \cdot 10^7$ double precision locations to store, and any operations with these factors are very time consuming. Thus it is of interest to know whether this is inherent in the problem structure or could be improved with an alternative factorization scheme. Similar storage problems are observed with PINENE (about $1.2 \cdot 10^7$ locations) and MARINE and METHANOL (about $9 \cdot 10^6$ locations). Now PINENE (24 minutes) and METHANOL (11 minutes) solve successfully and quite quickly because very few outer iterations are required. However, MARINE requires 201 outer iterations and over 6 hours of computation. Likewise, although GLIDER requires fewer than $5 \cdot 10^6$ locations, the problem fails to solve in 999 outer iterations and requires almost 6 hours of computation. The size of the factor storage is the main cause of these poor timings. On the other hand, MINSURFO, which has more variables than GLIDER, but no general constraints, is solved in 1.4 seconds, showing the effectiveness of the spectral gradient method.

The reason why GLIDER (and also CHAIN) requires so many iterations is not clear. The iterations of GLIDER remain throughout in phase 1 and require frequent changes to the set J. It may be that further consideration of the feasibility restoration algorithm should be given. One observation is that when a successful LCP step is made, the trust region constraint is active, and h^{k+1} is close to U, it is inadvisable to increase ρ because the next iteration is likely to make $h > U$. I have used a test to prevent this in a previous SQP code, with some benefit. Possibly some such test will be incorporated in a future release of the code.

A certain amount can be done to avoid these poor results by careful choice of parameters, in particular `ubd` and `mxgr`. The appendix shows a number of cases where significant improvement has been obtained. Also careful attention to scaling the constraints and variables can be beneficial, although this is not readily done when using standardized CUTEr test problems.

10. Comparison with `filterSQP`. To make an assessment of the reliability of `filterSD` and the effect on run times when the number of function and gradient calls is large, the code is compared with the `filterSQP` code of Fletcher and Leyffer [7] on

TABLE 10.1
Failure to find a satisfactory solution for `filterSD` *and* `filterSQP`.

Outcome	Both codes	Only filterSQP	Only filterSD
local infeasibility	21	20	4
maxit (999) iterations	2	5	1
other failures	1	8	7

the same 355 CUTEr test problems. Both codes are run with the same, most recent sparse matrix linear algebra code based on Schur complement updates. The same default options are used with both codes. (For `mxgr` = `100` and `rgtol` = 10^{-5} there is no equivalent parameter choice for `filterSQP`.) A full set of results for `filterSQP` is available on request from the author.

First, the reliability issue is considered. Table 10.1 indicates how many test problems report an outcome which is to some extent unsatisfactory. Cases where termination close to a local solution has occurred are regarded as being satisfactory. Both codes essentially identify the MESH problem as being unbounded. For this reason the MESH problem is not included.

Reasons for the "other failures" of `filterSD` have been discussed in the previous section. The tally for `filterSQP` includes three cases (BRITGAS (`+Inf`), NET2 (`NaN`), and YORKNET (`NaN`)) in which an IEEE exception arises in the Hessian evaluation. These problems are all solved by `filterSD` without any difficulty. It is interesting to see that `filterSD` reports far fewer instances of local infeasibility than `filterSD`. This may be due to the use of `l1sold` to find best L1 solutions when the linearised constraints are infeasible. All in all, `filterSD` seems to show up quite well in regard to reliability.

Next the timing issue is considered, and the extent to which it is affected by an increased number of function and gradient calls. A total of 264 of the 355 problems are solved by both codes in less than 1 second (for whatever reason). It is felt that little useful information is contained in these timings. The three problems for which the Hessian evaluation fails are also discounted. The timings (seconds) for the remaining 88 problems are shown in Table 10.2. A superscript (a, b, c, or d) shows that the timings are not directly comparable for the reason given in the table. There are six cases where `filterSD` finds an inferior local solution, and six cases for `filterSQP`. There are 39 problems for which both codes find the same feasible local solution. Of these there are 18 cases in which the times differ by a factor of less than 2, and otherwise 13 cases in which `filterSQP` wins, and 8 cases in which `filterSD` wins. To the latter might be added four cases in which `filterSD` finds a feasible local solution, which `filterSQP` reaches the iteration limit in a much longer time. Timings when both codes declare local infeasibility are not comparable since the partition J in `filterSD` is derived from a local L1 solution of the LP subproblem, whereas that in `filterSQP` is from a local pareto solution. Mention should be made of the three FLOSP2Hx problems which are known to be infeasible because the subset of linear constraints is itself inconsistent. `filterSQP` checks for this immediately and hence terminates very quickly, whereas `filterSD` does not.

Given the lack of access to Hessian information, other than through Ritz values, and the need to solve an LCP subproblem by repeated function and gradient calls, the `filterSD` code shows up much better than might have been expected, with timings that are not all that much worse than for `filterSQP`.

TABLE 10.2
Comparative timings (seconds) for `filterSD` *and* `filterSQP`.

Problem	filterSD	fil-SQP	Problem	filterSD	fil-SQP	Problem	filterSD	fil-SQP
READING7	4.011	1.729[a]	ROTDISC	10.38	1.858	SMMPSF	1.126	0.795[d]
ARTIF	5.065	3.356[b]	BRATU2D	1.588	0.704	BRATU2DT	3.674[b]	2.030[b]
BROYDNBD	3.364	3.644	CATENA	6.386	7.882[d]	CATENARY	6.390	16.49[c]
CLNLBEAM	1.665	0.200[a]	CORKSCRW	1.603	0.701	DRCAVTY1	7.489	6.453
DRCAVTY2	162.3	63.14[b]	DRCAVTY3	174.0	433.5[b]	DRUGDIS	8.298	5.305
DTOC1NA	3.155	9.808	DTOC1NB	3.555	10.10	DTOC1NC	335.0	6.750
DTOC1ND	7.917	6.741	DTOC2	98.53[a]	15.54	DTOC4	3.225	6.084
DTOC5	0.604	3.459	DTOC6	12.61	6.571	EIGENC	136.4[d]	11.22
EIGENC2	16.99	13.17	EIGENCCO	35.95	62.67	FLOSP2HH	23.10[b]	0.133[b]
FLOSP2HL	8.325[b]	0.130[b]	FLOSP2HM	14.11[b]	0.133[b]	FLOSP2TH	15.14	157.9[b]
FLOSP2TL	7.632	4.445	FLOSP2TM	9.383	417.9[c]	GAUSSELM	7.176[a]	6.524
GILBERT	0.347	17.36	HADAMARD	20.42	0.019[b]	HANGING	144.8[d]	259.4
HVYCRASH	19.39[d]	29.39[b]	JUNKTURN	58.97[b]	30.11[b]	LCH	0.682	51.29
LUBRIF	82.39[b]	42.18[b]	MADSSCHJ	1.557[b]	1.776	MINC44	19.53	4.311
MINPERM	4.153	2.553	MSQRTA	25.73	517.8[b]	MSQRTB	33.93	3.406
NGONE	1.463	0.071[a]	OPTCDEG2	1.026	0.420	OPTCDEG3	1.505	1.004
OPTCTRL3	58.46	8.716[a]	ORTHREGA	36.51	42.88	ORTHREGC	2.700	6.269
ORTHREGD	1.847	8.928[a]	ORTHREGF	7.574[a]	23.42	ORTHRGDM	399.1	986.1[c]
ORTHRGDS	2.688[a]	12.99	POROUS1	3.669	3.321	POROUS2	5.267	2.934
QR3D	3.641	0.802	QR3DBD	17.64[d]	53.44[d]	READING1	3.087	1.650
READING4	3.237	0.515	READING9	1.443	0.312[a]	SEMICON1	0.054[d]	2.439
SEMICON2	1.289[c]	0.544	SINROSNB	2.460	23.95[c]	SPMSQRT	13.37[b]	0.419[b]
SREADIN3	3.356	1.288	SSNLBEAM	14.60[a]	32.60	SVANBERG	2.679	1.827
UBH5	4.279	4.025	VANDERM1	5.847[d]	1.921[b]	VANDERM2	5.854[d]	1.920[b]
VANDERM3	1.315	1.718[b]	ZAMB2	3.393	0.691	CATMIX	2.747[a]	8.517
CHAIN	44.15[c]	46.26	CHANNEL	336.6	15.39[b]	ELEC	40.40[d]	867.4
GASOIL	17321	3804.	GLIDER	23290[c]	20763[c]	MARINE	21943	68315
METHANOL	664.9	289.0	MINSURFO	1.379	243.3	PINENE	1436.	1143.
POLYGON	11.60	1.106[d]	ROBOTARM	95.56	599.8[b]	ROCKET	308.6	74.37[d]
STEERING	266.7	7.117[b]						

Key to superscripts:
a: finds an inferior local solution. b: declares (local) infeasibility.
c: terminates at maxit (999) iterations. d: terminates prematurely.

TABLE 10.3
Timings (seconds) for alternative parameter choices for `filterSD`

Problem	Default	New	Problem	Default	New	Problem	Default	New
DTOC1NC	335.0	6.787	DTOC2	98.53	41.40	EIGENC	136.4	125.7
HANGING	144.8	413.4	HVYCRASH	19.39	46.17	ORTHRGDM	399.1	84.54
SEMICON1	0.054	0.905	CHAIN	44.15	1.755	CHANNEL	336.6	141.8
ELEC	15.56	9.988	MARINE	21943	8592.	ROCKET	308.6	76.15
STEERING	266.7	168.1						

Moreover, there are various problems reported in the appendix for which the performance of `filterSD` can be significantly improved by departing from the default parameter choices. Timings for these problems are shown in Table 10.3, comparing the default parameter choice with the new choice specified in the appendix. Some significant improvements can be seen. In particular, increasing `mxgr` to 400 for CHAIN, CHANNEL, MARINE, and ROCKET leads to a substantial decrease in the number of outer iterations, and a corresponding improvement in timings. In the case of CHANNEL and MARINE, this is achieved despite an increase in the number of function and gradient calls.

Appendix. These performance statistics are obtained with default parameters `maxit` $= 999$, `mxgr` $= 100$, `mlp` $= 50$, `mxf` $= 100$, `ubd` $=$ `1.D4`, `htol` $=$ `1.D` $- 6$, `rgtol` $=$ `1.D` $- 5$, and `tol` $=$ `1.D` $- 12$. The limit on the size of the filter is `mxf` $= 100$ since two problems require more space than the `glcpd` default value of 50. In practice it has been unusual for the default limit to be exceeded.

summary for small Hock-Schittkowski problems

problem	n	m	f^*	h^*	rgnorm	k	itn	#f	#g	ifail
HS10	2	1	-1.000000	0.622E-14	0.000E+00	1	9	10	10	
HS100	7	4	680.6301	0.239E-14	0.760E-06	5	6	94	88	
HS100LNP	7	2	680.6301	0.177E-13	0.679E-05	5	8	175	145	
HS100MOD	7	4	678.6796	0.000E+00	0.812E-06	6	5	80	70	
HS101	7	5	1809.765	0.111E-15	0.277E-05	5	19	602	534	
HS102	7	5	911.8806	0.389E-15	0.426E-04	4	28	709	638	
HS103	7	5	543.6680	0.201E-15	0.110E-02	3	29	791	607	
HS104	8	5	3.951163	0.187E-15	0.663E-05	4	11	279	247	
HS106	8	6	7049.248	0.291E-10	0.725E-05	2	17	58	54	
HS107	9	6	5055.012	0.134E-13	0.387E-09	1	5	61	44	
HS108	9	13	-0.866025E+00	0.250E-15	0.169E-14	1	19	382	229	
HS109	9	10	5362.069	0.373E-10	0.107E-03	1	12	442	382	
HS11	2	1	-8.498464	0.888E-15	0.647E-05	1	6	16	14	
HS111	10	3	-47.76109	0.125E-15	0.572E-06	7	11	489	435	
HS111LNP	10	3	-47.76109	0.125E-15	0.572E-06	7	11	489	435	
HS113	10	8	24.30621	0.581E-12	0.596E-05	4	5	65	59	
HS114	10	11	-1768.807	0.235E-12	0.806E-05	1	12	74	62	
HS116	13	14	97.58751	0.151E-13	0.180E-06	1	12	77	66	
HS117	15	5	32.34868	0.444E-14	0.994E-06	4	6	188	151	
HS12	2	1	-30.00000	0.000E+00	0.445E-07	1	8	23	22	
HS13	2	1	1.000003	0.000E+00	0.000E+00	0	33	33	33	
HS14	2	2	1.393465	0.222E-15	0.000E+00	0	6	6	6	
HS15	2	2	306.5000	0.000E+00	0.000E+00	0	3	4	4	
HS16	2	2	23.14466	0.000E+00	0.000E+00	0	5	5	5	
HS17	2	2	1.000000	0.000E+00	0.000E+00	0	8	26	23	
HS18	2	2	5.000000	0.000E+00	0.119E-09	1	9	39	34	
HS19	2	2	-6961.814	0.000E+00	0.000E+00	0	6	7	7	
HS20	2	3	40.19873	0.111E-15	0.000E+00	0	4	4	4	
HS22	2	2	1.000000	0.000E+00	0.000E+00	0	5	5	5	
HS23	2	5	2.000000	0.000E+00	0.000E+00	0	6	6	6	
HS26	3	1	0.123987E-07	0.333E-11	0.713E-05	2	5	61	52	
HS27	3	1	0.400000E-01	0.000E+00	0.000E+00	2	7	117	69	
HS29	3	1	-22.62742	0.000E+00	0.144E-05	2	7	63	59	
HS30	3	1	1.000000	0.000E+00	0.000E+00	1	20	22	21	
HS31	3	1	6.000000	0.000E+00	0.285E-06	2	5	16	13	
HS32	3	2	1.000000	0.000E+00	0.000E+00	0	3	15	9	
HS33	3	2	-4.000000	0.000E+00	0.000E+00	0	5	5	5	
HS34	3	2	-0.834032E+00	0.000E+00	0.000E+00	0	7	18	13	
HS39	4	2	-1.000000	0.111E-13	0.640E-09	2	20	121	102	
HS40	4	3	-0.250000E+00	0.555E-16	0.164E-04	1	15	51	49	
HS42	4	2	13.85786	0.493E-13	0.397E-05	2	4	16	13	
HS43	4	3	-44.00000	0.155E-14	0.500E-05	2	8	57	44	
HS46	5	2	0.176459E-06	0.534E-15	0.497E-04	3	11	234	198	
HS47	5	3	-0.267142E-01	0.666E-15	0.667E-05	2	28	295	255	
HS56	7	4	-0.626889E-43	0.155E-12	0.488E-30	3	28	460	366	
HS57	2	1	0.284597E-01	0.000E+00	0.979E-07	1	5	56	40	
HS59	2	3	-6.749505	0.000E+00	0.469E-08	2	9	94	61	
HS6	2	1	0.123260E-31	0.000E+00	0.222E-15	1	4	8	6	
HS60	3	1	0.325682E-01	0.780E-11	0.506E-05	2	5	35	33	
HS61	3	2	0.000000E+00	0.125E+01	0.000E+00	2	2	2	2	fail 3
HS63	3	2	961.7152	0.000E+00	0.476E-07	1	9	24	22	
HS64	3	1	6299.842	0.000E+00	0.445E-03	2	10	77	74	
HS65	3	1	0.953529E+00	0.355E-14	0.128E-05	2	5	24	22	

HS66	3	2	0.518163E+00	0.666E-15	0.604E-08	1	4	14	12
HS67	3	14	-1162.119	0.227E-12	0.757E-08	1	12	122	102
HS68	4	2	-0.920425E+00	0.527E-16	0.732E-06	2	12	150	109
HS69	4	2	-956.7129	0.278E-16	0.185E-04	2	5	196	132
HS7	2	1	-1.732051	0.444E-15	0.350E-15	1	10	18	17
HS70	4	1	0.269086E+00	0.000E+00	0.290E-06	3	2	12	9
HS71	4	2	17.01402	0.222E-13	0.599E-07	1	5	15	13
HS72	4	2	727.6794	0.282E-17	0.162E-05	2	25	346	300
HS73	4	3	29.89438	0.000E+00	0.000E+00	0	3	4	4
HS74	4	5	5126.498	0.568E-13	0.219E-05	1	11	32	29
HS75	4	5	5174.413	0.284E-13	0.000E+00	0	11	23	23
HS77	5	2	0.241505E+00	0.151E-11	0.436E-06	3	6	61	59
HS78	5	3	-2.919700	0.133E-14	0.837E-04	2	20	197	172
HS79	5	3	0.787768E-01	0.444E-15	0.422E-06	2	5	36	33
HS8	2	2	-1.000000	0.444E-14	0.000E+00	0	6	6	6
HS80	5	3	0.539498E-01	0.144E-14	0.204E-05	2	6	30	29
HS81	5	3	0.539498E-01	0.167E-14	0.130E-05	2	11	103	90
HS83	5	3	-30665.54	0.000E+00	0.000E+00	0	5	12	12
HS84	5	3	-0.528034E+07	0.000E+00	0.000E+00	0	8	26	26
HS85	5	21	-2.215605	0.142E-13	0.000E+00	0	8	17	17
HS88	2	1	1.362657	0.000E+00	0.629E-12	1	15	341	179
HS89	3	1	1.362657	0.000E+00	0.954E-06	2	15	309	207
HS90	4	1	1.362657	0.000E+00	0.567E-06	3	10	316	191
HS91	5	1	1.362657	0.000E+00	0.431E-06	4	12	326	249
HS92	6	1	1.362657	0.000E+00	0.223E-05	5	16	498	379
HS93	6	2	135.0760	0.138E-13	0.245E-05	4	4	213	182
HS95	6	4	0.156195E-01	0.000E+00	0.000E+00	0	2	2	2
HS96	6	4	0.156195E-01	0.000E+00	0.000E+00	0	2	2	2
HS97	6	4	3.135809	0.000E+00	0.000E+00	0	3	6	6
HS98	6	4	3.135809	0.000E+00	0.000E+00	0	3	6	6
HS99	7	2	-0.831080E+09	0.146E-10	0.198E+01	5	4	46	44

Summary for other small CUTE problems

problem	n	m	f^*	h^*	rgnorm	k	itn	#f	#g	ifail
AIRCRFTA	8	5	0.000000E+00	0.241E-15	0.000E+00	0	4	4	4	
ALJAZZAF	3	1	75.00500	0.000E+00	0.000E+00	1	17	34	32	
ALLINITC	4	1	30.49654	0.289E-12	0.510E-05	1	22	82	77	
ALSOTAME	2	1	0.820850E-01	0.000E+00	0.000E+00	0	5	6	6	
ARGAUSS	3	15	0.693889E-17	0.338E-03	0.000E+00	0	4	4	4	fail 3
BT1	2	1	-1.000000	0.356E-18	0.597E-09	1	9	25	21	
BT10	2	2	-1.000000	0.000E+00	0.000E+00	0	7	7	7	
BT11	5	3	0.824892E+00	0.167E-15	0.318E-05	2	8	59	54	
BT12	5	3	6.188119	0.118E-12	0.357E-05	2	5	14	12	
BT13	5	1	0.000000E+00	0.730E-06	0.348E-13	1	48	267	253	
BT2	3	1	0.325682E-01	0.178E-14	0.850E-07	2	11	116	109	
BT4	3	2	-45.51055	0.178E-14	0.616E-05	1	8	30	29	
BT5	3	2	961.7152	0.000E+00	0.832E-08	1	8	21	19	
BT6	5	2	0.277045E+00	0.235E-11	0.732E-05	3	6	53	51	
BT7	5	3	306.5000	0.160E-12	0.193E-05	2	11	146	79	
BT8	5	2	1.000000	0.182E-11	0.000E+00	3	20	22	21	
BT9	4	2	-1.000000	0.111E-13	0.640E-09	2	20	121	102	
BYRDSPHR	3	2	-4.683300	0.000E+00	0.525E-09	1	11	40	36	
CANTILVR	5	1	1.339956	0.000E+00	0.348E-05	4	20	558	458	
CB2	3	3	1.952224	0.300E-14	0.192E-06	1	6	22	17	
CB3	3	3	2.000000	0.000E+00	0.000E+00	0	7	11	9	
CHACONN1	3	3	1.952224	0.444E-15	0.838E-07	1	5	16	13	
CHACONN2	3	3	2.000000	0.000E+00	0.000E+00	0	7	11	9	
CLUSTER	2	2	0.000000E+00	0.270E-17	0.000E+00	0	11	11	11	
CONGIGMZ	3	5	28.00000	0.000E+00	0.000E+00	0	9	28	24	
COOLHANS	9	9	0.297707E-13	0.646E-02	0.768E-06	1	19	69	68	fail 3
CSFI1	5	4	-0.105000E+02	0.336E+02	0.147E+01	1	2	5	5	fail11

(CSFI1 fails due to division by zero in the constraint evaluation)

CSFI2	5	4	55.01761	0.444E-15	0.000E+00	0	13	195	174	
DEMYMALO	3	3	-3.000000	0.000E+00	0.000E+00	0	8	9	9	
DIPIGRI	7	4	680.6301	0.000E+00	0.760E-06	5	6	94	88	
DIXCHLNG	10	5	1079.875	0.111E-15	0.165E-04	5	23	448	391	
ERRINBAR	18	9	28.04526	0.711E-12	0.537E-06	3	18	307	270	
EXPFITA	5	22	0.113661E-02	0.107E-13	0.302E-06	1	3	26	19	
FLETCHER	4	4	0.000000E+00	0.100E+01	0.000E+00	0	2	2	2	fail 3
GIGOMEZ1	3	3	-3.000000	0.778E-12	0.000E+00	0	7	9	9	
GOTTFR	2	2	0.000000E+00	0.111E-15	0.000E+00	0	8	12	12	
GROWTH	3	12	0.301981E-13	0.244E+01	0.000E+00	0	8	16	16	fail 3
HAIFAS	13	9	-0.450000E+00	0.105E-11	0.609E-05	8	10	33	25	
HALDMADS	6	42	0.122371E-03	0.777E-15	0.000E+00	0	14	168	135	
HATFLDF	3	3	0.000000E+00	0.468E-15	0.000E+00	0	20	49	43	
HEART6	6	6	0.000000E+00	0.102E-13	0.000E+00	0	59	342	296	
HEART8	8	8	0.000000E+00	0.115E-11	0.000E+00	0	16	97	86	
HIMMELBC	2	2	0.000000E+00	0.111E-12	0.000E+00	0	6	7	7	
HIMMELBD	2	2	0.586198E-13	0.243E+01	0.361E-06	1	6	9	9	fail 3
HIMMELBE	3	3	0.000000E+00	0.000E+00	0.000E+00	0	3	3	3	
HIMMELBK	24	14	0.518143E-01	0.948E-15	0.000E+00	0	5	18	18	
HIMMELP2	2	1	-62.05394	0.000E+00	0.255E-06	2	4	26	19	
HIMMELP3	2	2	-59.01318	0.000E+00	0.000E+00	0	4	23	17	
HIMMELP4	2	3	-59.01318	0.000E+00	0.000E+00	0	4	23	18	
HIMMELP5	2	3	-59.01318	0.000E+00	0.000E+00	0	10	18	18	
HIMMELP6	2	5	-59.01318	0.000E+00	0.000E+00	0	8	19	19	
HYPCIR	2	2	0.000000E+00	0.555E-15	0.000E+00	0	7	9	9	
KIWCRESC	3	2	0.111192E-10	0.155E-14	0.472E-05	1	12	31	30	
LEWISPOL	6	9	0.330409E-20	0.469E-04	0.000E+00	0	8	8	8	fail 3
LOOTSMA	3	2	0.000000E+00	0.400E+01	0.000E+00	2	1	1	1	fail 3
MADSEN	3	6	0.616432E+00	0.000E+00	0.263E-06	1	10	25	21	
MAKELA1	3	2	-1.414214	0.555E-16	0.426E-05	1	12	34	29	
MAKELA2	3	3	7.200000	0.000E+00	0.111E-14	1	5	10	8	
MAKELA3	21	20	0.100447E-26	0.545E-12	0.634E-13	1	30	54	51	
MARATOS	2	1	-0.100000E+01	0.668E-16	0.223E-05	1	4	10	8	
MATRIX2	6	2	0.454747E-12	0.000E+00	0.135E-05	2	21	45	44	
MINMAXBD	5	20	115.7064	0.178E-14	0.232E-06	2	34	170	153	
MIFFLIN1	3	2	-0.100000E+01	0.000E+00	0.465E-05	1	6	15	13	
MIFFLIN2	3	2	-1.000000	0.000E+00	0.000E+00	1	18	48	42	
MINMAXRB	3	4	-0.666134E-15	0.133E-14	0.000E+00	0	3	3	3	
MISTAKE	9	13	-1.000000	0.633E-13	0.130E-07	4	30	627	459	
MWRIGHT	5	3	1.288383	0.000E+00	0.682E-05	2	12	160	140	
NYSTROM5	18	20	0.000000E+00	0.122E-06	0.000E+00	0	12	59	59	
PFIT1	3	3	0.000000E+00	0.404E-11	0.000E+00	0	29	87	82	
PFIT2	3	3	0.000000E+00	0.169E-13	0.000E+00	0	31	87	84	
PFIT3	3	3	0.000000E+00	0.591E-13	0.000E+00	0	28	80	79	
PFIT4	3	3	0.000000E+00	0.360E-13	0.000E+00	0	39	99	97	
POLAK1	3	2	2.718282	0.000E+00	0.101E-05	1	7	15	15	
POLAK3	12	10	5.933003	0.311E-14	0.620E-07	9	21	458	379	
POLAK4	3	3	0.242801E-15	0.732E-13	0.312E-08	1	4	7	5	
POLAK5	3	2	50.00000	0.000E+00	0.315E-07	1	15	348	186	
POLAK6	5	4	-44.00000	0.444E-14	0.148E-07	2	32	1489	1140	
POWELLBS	2	2	0.111022E-15	0.157E-03	0.267E-03	1	13	20	20	fail 3
POWELLSQ	2	2	0.511591E-12	0.715E-04	0.155E-14	1	23	25	24	fail 3
RES	20	14	0.000000E+00	0.103E-14	0.000E+00	12	1	1	1	
RK23	17	11	0.833333E-01	0.593E-15	0.000E+00	1	8	26	23	
ROBOT	14	2	5.462841	0.445E-12	0.365E-07	5	7	114	102	
ROSENMMX	5	4	-44.00000	0.400E-14	0.156E-06	2	93	872	773	
S316-322	2	1	0.000000E+00	0.100E+01	0.000E+00	2	1	1	1	fail 3
SNAKE	2	2	-0.684100E-25	0.684E-29	0.000E+00	0	3	3	3	
SPIRAL	3	2	0.171145E-09	0.101E-25	0.305E-05	2	7	637	364	
TENBARS1	18	9	2302.549	0.380E-12	0.528E-08	4	38	671	610	
TENBARS2	18	8	2277.946	0.802E-12	0.101E-05	5	41	909	737	

TENBARS3	18	8	2247.129	0.831E-12	0.505E-06	4	36	598	546
TENBARS4	18	9	368.4932	0.341E-12	0.390E-06	5	39	728	658
TRIGGER	7	6	0.000000E+00	0.147E-13	0.000E+00	0	28	54	52
TRUSPYR1	11	4	11.22874	0.178E-13	0.137E-07	2	14	154	146
TRUSPYR2	11	11	11.22874	0.213E-13	0.000E+00	0	12	96	96
TRY-B	2	1	1.000000	0.000E+00	0.000E+00	1	8	8	8
TWOBARS	2	2	1.508652	0.000E+00	0.525E-05	1	6	22	20
WOMFLET	3	3	0.000000E+00	0.000E+00	0.000E+00	1	7	24	20
ZECEVIC3	2	2	97.30945	0.666E-15	0.157E-06	1	11	40	35
ZECEVIC4	2	2	7.557508	0.000E+00	0.277E-06	1	6	19	16
ZY2	3	2	2.000000	0.000E+00	0.000E+00	0	5	7	7

Summary for medium sized CUTE problems

problem	n	m	f^*	h^*	rgnorm	k	itn	#f	#g	ifail
AIRPORT	84	42	47952.70	0.210E-13	0.362E-04	42	12	773	615	
CORE1	65	59	91.05624	0.100E-10	0.000E+00	0	28	544	470	
DECONVC	61	1	0.263613E-13	0.913E-15	0.683E-07	36	4	461	266	
DISC2	29	23	0.446425E-12	0.142E+02	0.199E-05	17	46	1619	1171	fail 3
DISCS	36	66	0.195515E-12	0.800E+01	0.656E-06	5	435	2569	2515	fail 3
DNIEPER	61	24	18744.01	0.482E-12	0.000E+00	0	3	22	22	
HATFLDG	25	25	0.000000E+00	0.567E-14	0.000E+00	0	12	33	32	
HS99EXP	31	21	-0.114485E+13	0.319E-05	0.344E+07	7	138	2009	1207	fail 6
HYDCAR20	99	99	0.000000E+00	0.855E-12	0.000E+00	0	10	27	27	
HYDCAR6	29	29	0.000000E+00	0.161E-12	0.000E+00	0	10	45	45	
LAKES	90	78	0.350525E+06	0.233E-09	0.107E+00	12	48	956	830	
LAUNCH	25	28	9.004905	0.338E-13	0.000E+00	3	7	35	35	
MESH	41	48	-0.295148E+20	0.229E-03	0.344E+09	2	35	89	75	fail11
(MESH is unbounded)										
METHANB8	31	31	0.000000E+00	0.720E-12	0.000E+00	0	3	3	3	
METHANL8	31	31	0.000000E+00	0.743E-12	0.000E+00	0	5	8	8	
MRIBASIS	36	55	18.21790	0.174E-12	0.000E+00	5	3	3	3	
NET1	48	57	0.941194E+06	0.123E-11	0.000E+00	0	13	123	123	
OPTCNTRL	32	20	550.0000	0.108E-15	0.000E+00	0	4	13	13	
ORTHREGB	27	6	0.000000E+00	0.353E-13	0.000E+00	21	2	2	2	
PRODPL0	60	29	58.79010	0.444E-15	0.000E+00	0	9	13	13	
PRODPL1	60	29	35.73897	0.193E-13	0.000E+00	0	7	11	11	
SWOPF	83	92	0.678602E-01	0.578E-13	0.923E-05	2	5	48	41	

Summary for large CUTE problems

problem	n	m	f^*	h^*	rgnorm	k	itn	#f	#g	ifail
BRITGAS	450	360	0.000000E+00	0.528E-12	0.000E+00	46	89	6916	5896	
CORE2	157	134	72.90000	0.864E-11	0.172E-17	2	24	1413	1164	
CRESC100	6	200	0.744359E+00	0.853E-12	0.108E-03	1	15	313	247	
CRESC50	6	100	0.786247E+00	0.586E-11	0.000E+00	0	38	621	500	
ELATTAR	7	102	74.20618	0.269E-14	0.845E-13	6	17	63	57	
GROUPING	100	125	13.85040	0.000E+00	0.000E+00	0	1	1	1	
LEAKNET	156	153	8.002076	0.532E-12	0.000E+00	0	9	38	35	
NET2	144	160	0.118674E+07	0.382E-11	0.000E+00	0	13	356	353	
READING6	102	50	-144.6597	0.932E-14	0.985E-05	8	13	601	511	
READING7	1002	500	-1307.706	0.515E-13	0.000E+00	0	8	510	510	
ROTDISC	905	1081	7.872068	0.370E-09	0.000E+00	0	62	6148	5718	
SMMPSF	720	263	0.103293E+07	0.262E-09	0.000E+00	0	38	3182	3168	
SSEBNLN	194	96	0.161706E+08	0.000E+00	0.000E+00	0	11	362	362	
YORKNET	312	256	17933.91	0.430E-11	0.917E-04	9	41	1866	1430	
ZAMB2-10	270	96	-1.582377	0.115E-12	0.608E-05	20	7	594	490	
ZAMB2-11	270	96	-1.116141	0.858E-13	0.976E-05	59	8	739	563	
ZAMB2-8	138	48	-0.152936E+00	0.591E-13	0.773E-05	30	5	286	190	
ZAMB2-9	138	48	-0.354585E+00	0.673E-13	0.207E-05	13	5	379	297	

Summary for variable sized CUTE problems

problem	n	m	f^*	h^*	rgnorm	k	itn	#f	#g	ifail
ARGTRIG	100	100	0.000000E+00	0.114E-12	0.000E+00	0	4	4	4	
ARTIF	1002	1000	0.000000E+00	0.771E-11	0.000E+00	46	22	1160	881	
BDVALUE	1002	1000	0.000000E+00	0.109E-13	0.000E+00	0	3	3	3	
BRATU2D	1024	900	0.000000E+00	0.491E-13	0.000E+00	0	4	4	4	
BRATU2DT	1024	900	0.154660E-12	0.146E-02	0.476E-10	4	10	15	15	fail 3
BRATU3D	1000	512	0.000000E+00	0.768E-13	0.000E+00	0	4	4	4	
BROYDN3D	1000	1000	0.000000E+00	0.600E-09	0.000E+00	785	5	5	5	
BROYDNBD	1000	1000	0.000000E+00	0.175E-14	0.000E+00	1	10	13	13	
CATENA	501	166	-0.348530E+06	0.207E-11	0.507E-02	331	242	26694	23711	
CATENARY	501	166	-0.348403E+06	0.990E-12	0.440E-03	331	408	43023	38485	
CBRATU2D	1058	882	0.000000E+00	0.341E-13	0.000E+00	441	4	4	4	
CBRATU3D	686	250	0.000000E+00	0.200E-13	0.000E+00	125	4	4	4	
CHANDHEQ	100	100	0.000000E+00	0.238E-11	0.000E+00	0	20	20	20	
CHEMRCTA	1000	1000	0.000000E+00	0.712E-10	0.000E+00	0	3	104	104	
CHEMRCTB	1000	1000	0.000000E+00	0.137E-09	0.000E+00	0	4	104	104	
CLNLBEAM	1503	1000	346.3335	0.578E-14	0.708E-02	495	12	828	582	
CORKSCRW	906	700	44.36876	0.387E-12	0.194E-08	1	18	1296	1131	
COSHFUN	61	20	-0.399937E+09	0.161E+04	0.000E+00	41	999	2639	2565	fail 5
COSHFUN with ubd=1.D0										
	61	20	-0.773267E+00	0.110E-12	0.601E-07	41	87	8008	7176	
DITTERT	327	264	-1.000000	0.000E+00	0.000E+00	0	3	12	11	
DIXCHLNV	100	50	17697.23	0.394E-12	0.215E+05	50	21	282	279	
DRCAVTY1	1225	961	0.000000E+00	0.843E-13	0.000E+00	0	9	18	18	
DRCAVTY2	1225	961	0.000000E+00	0.416E-11	0.000E+00	0	146	1066	1037	
DRCAVTY3	1225	961	0.000000E+00	0.323E-09	0.000E+00	0	130	1903	1757	
DRUGDIS	3004	2000	4.275953	0.152E-12	0.000E+00	0	14	980	980	
DRUGDISE	63	50	0.232302E-14	0.137E+01	0.814E-03	1	10	33	33	fail 3
DTOC1NA	2998	1996	2.086569	0.335E-13	0.568E-06	998	4	132	126	
DTOC1NB	2998	1996	3.583103	0.359E-13	0.944E-05	998	5	262	234	
DTOC1NC	2998	1996	17.57963	0.380E-13	0.742E-05	998	178	18377	16841	
DTOC1NC with ubd=1.D3										
	2998	1996	17.57963	0.206E-13	0.809E-05	998	8	411	391	
DTOC1ND	2998	1996	23.77031	0.103E-12	0.888E-05	998	11	958	875	
DTOC2	2998	1996	0.497223E+00	0.835E-15	0.365E-04	998	293	33058	29493	
DTOC2 with mxgr=400										
	2998	1996	0.497223E+00	0.906E-13	0.170E-04	998	65	28367	25763	
DTOC4	2999	1998	2.874890	0.612E-13	0.737E-05	999	4	104	93	
DTOC5	1999	999	1.534946	0.373E-13	0.377E-05	999	5	19	18	
DTOC6	2001	1000	17176.45	0.104E-12	0.143E-05	1000	27	1804	1646	
EG3	1001	2000	0.127838E+00	0.294E-14	0.000E+00	925	19	75	75	
EIGENA	110	110	0.000000E+00	0.000E+00	0.000E+00	2	2	2	2	
EIGENA2	110	55	0.000000E+00	0.000E+00	0.000E+00	101	2	8	6	
EIGENACO	110	55	0.000000E+00	0.000E+00	0.000E+00	101	2	8	6	
EIGENB	110	110	0.000000E+00	0.900E+01	0.000E+00	89	2	2	2	fail 3
EIGENB2	110	55	18.00000	0.000E+00	0.000E+00	83	2	4	3	
EIGENBCO	110	55	9.000000	0.000E+00	0.000E+00	91	2	4	3	
EIGENC	462	462	0.000000E+00	0.106E+02	0.685E-05	77	133	12254	11898	fail20
EIGENC with tol=1.D-8										
	462	462	0.000000E+00	0.143E-12	0.000E+00	0	121	7773	7479	
EIGENC2	462	231	0.491367E-09	0.148E-11	0.477E-04	231	120	13099	12133	
EIGENCCO	462	231	0.138447E-08	0.427E-14	0.183E-03	231	168	18290	16780	
EIGMAXA	11	11	-1.000000	0.000E+00	0.000E+00	0	2	4	4	
EIGMAXB	11	11	-0.810141E-01	0.191E-15	0.000E+00	0	8	11	11	
EIGMAXC	22	22	-1.000000	0.607E-07	0.000E+00	0	4	8	8	
EIGMINA	101	101	1.000000	0.000E+00	0.000E+00	0	2	4	4	
EIGMINB	101	101	0.967435E-03	0.198E-11	0.000E+00	0	8	22	22	
EIGMINC	22	22	1.000000	0.607E-07	0.000E+00	0	4	8	8	
FLOSP2HH	1323	1243	0.375404E-06	0.100E+01	0.551E-35	11	23	599	595	fail 3
FLOSP2HL	1323	1243	0.951607E-09	0.100E+01	0.000E+00	11	7	95	95	fail 3
FLOSP2HM	1323	1243	0.242630E-08	0.100E+01	0.000E+00	11	12	364	362	fail 3

```
FLOSP2TH 1323 1243   0.000000E+00 0.424E-06 0.000E+00   8  19   678   678
FLOSP2TL 1323 1243   0.000000E+00 0.200E-09 0.000E+00   8   6   107   107
FLOSP2TM 1323 1243   0.000000E+00 0.628E-08 0.000E+00   8  13   453   453
GAUSSELM  506 1135  -10.27928     0.310E-13 0.000E+00   0  37  2960  2948
GILBERT  1000    1   482.0273     0.283E-13 0.433E-05 999  25  1233  1138
HADAMARD  401 1010   1.138155     0.351E-10 0.000E+00   0  72  6631  6631
HANGING  1800 1150  -0.999042E+04 0.714E+03 0.171E+031664 239 24329 23849 fail 6
HANGING with ubd=1.D0
         1800 1150  -10922.42     0.183E-10 0.532E-03 690 716 73116 70460
HET-Z       2 1002   1.000000     0.000E+00 0.000E+00   1   2     2     2
HVYCRASH 2004 1500  -0.217281E+00 0.172E+02 0.166E+00  95  16  1275  1190 fail11
   (HVYCRASH returns a c/s value of ~1.D27 when x~1.D1)
HVYCRASH with fmin=-1.D100
         2004 1500  -0.218500E+00 0.348E-13 0.414E-17 477  43  3760  3398
INTEGREQ  102  100   0.000000E+00 0.156E-12 0.000E+00   0   3     3     3
JUNKTURN 1010  700   0.121612E-13 0.104E+01 0.358E-06 297 143 15986 14217 fail 3
LCH       600    1  -4.318289     0.611E-16 0.337E-04 599  49  4232  3875
LUBRIF    751  500   0.285115E-10 0.850E+01 0.872E-08   2  86  8058  7686 fail 3
MADSSCHJ  201  398   0.100044E-10 0.110E+06 0.282E+03 200  23   427   102 fail 3
MANNE    1095  730  -0.974573E+00 0.157E-12 0.000E+00   1   2     2     2
MINC44   1113 1032   0.385245E-03 0.102E-14 0.517E-04  71  31  2118  1977
MINPERM   583  520   0.936657E-03 0.102E-14 0.337E-06  64  27  1367  1250
MSQRTA    529  529   0.000000E+00 0.225E-12 0.000E+00   0  15   708   708
MSQRTB    529  529   0.000000E+00 0.210E-12 0.000E+00   0  25   674   669
NGONE     100 1273  -0.642870E+00 0.244E-14 0.296E-07  43  69  5857  4924
OET2        3 1002   0.871596E-01 0.333E-15 0.000E+00   0   6    49    49
OET4        4 1002   0.429543E-02 0.115E-12 0.000E+00   0  13   746   734
OET5        5 1002   0.265009E-02 0.111E-15 0.329E-06   1  15   527   517
OET6        5 1002   0.206974E-02 0.222E-15 0.000E+00   0  24   902   875
OET7        7 1002   0.206974E-02 0.155E-14 0.353E-05   2  27   631   579
OPTCDEG2 1202  800   229.5734     0.713E-13 0.564E-05   5  14  1268   945
OPTCDEG3 1202  800   46.14567     0.851E-13 0.543E-06   7  16  2398  1563
OPTCTRL3 1202  800   19013.46     0.196E-12 0.612E+03 399 924103897 92450
OPTCTRL6  122   80   2048.155     0.336E-13 0.307E+02  39  97 11162  9797
OPTMASS  1210 1005  -0.123266E+00 0.117E-11 0.858E-05 604   8   330   330
ORTHRDM2  203  100   7.775725     0.313E-11 0.183E-03 103   9   528   507
ORTHRDS2  203  100   0.304371E+02 0.684E-09 0.102E-03 103  30  2743  2183 fail 6
ORTHREGA 2053 1024   5661.433     0.135E-12 0.161E-021029  41  4282  4005
ORTHREGC 1005  500   18.79065     0.170E-12 0.290E-05 505  17  1776  1585
ORTHREGD 1003  500   151.2351     0.147E-10 0.636E-03 503  16  1415  1325
ORTHREGE   36   20   4.249977     0.127E-11 0.196E-06  16  21  1361  1213
ORTHREGF 1205  400   23.75133     0.309E-13 0.217E-04 805  37  3796  3461
ORTHRGDM 2003 1000   1024.991     0.439E-06 0.159E-06 134 459 48398 45249
ORTHRGDM with mxgr=200
         2003 1000   405.8523     0.153E-06 0.142E-04 505 105 19889 18690
ORTHRGDS 1003  500   300.6445     0.803E-10 0.230E-01 503  37  3385  2988
POROUS1  1024  900   0.000000E+00 0.963E-10 0.000E+00   0  11    19    19
POROUS2  1024  900   0.000000E+00 0.386E-09 0.000E+00   0   7   118   113
QR3D      610  610   0.000000E+00 0.647E-13 0.000E+00   0  20   120   117
QR3DBD    457  610   0.000000E+00 0.115E+01 0.128E-08  72  54  1313  1298 fail 6
READING1 2002 1000  -0.160480E+00 0.413E-11 0.944E-05   8   9   531   527
READING3  102   51  -0.152963E+00 0.139E-13 0.116E-05   1   7   130   117
READING4 1001 1000  -0.290472E+00 0.554E-12 0.168E-07   4  23  1867  1712
READING5 1001 1000  -0.224864E-16 0.800E-13 0.000E+00   0   5     5     5
READING9 2002 1000  -0.442082E-01 0.104E-13 0.955E-05 214   9   796   796
SEMICON1  502  500   0.000000E+00 0.132E+03 0.512E+00 392  12    45    43 fail11
   (SEMICON1 returns a c/s value of ~1.D87 when x~1.D3)
SEMICON1 with fmin=-1.D100
          502  500   0.000000E+00 0.744E-11 0.000E+00 247 130   943   683
SEMICON2  502  500   0.000000E+00 0.155E-11 0.000E+00   0  41   161   161
SINROSNB 1000  999   0.199602E+06 0.150E-10 0.124E-04   2  11   183   123
SPMSQRT  1000 1664   0.000000E+00 0.115E+02 0.000E+00   4  41   472   463 fail 6
```

```
SREADIN3 2002 1001  -0.152571E+00 0.315E-11 0.944E-05   8  14  1008  1003
SSNLBEAM 3003 2000   346.2345     0.667E-14 0.305E-02 750  33  2785  2386
SVANBERG  500  500   835.1869     0.182E-11 0.181E-04 102  22  2390  2073
TFI1        3  501   5.334687     0.000E+00 0.215E-05   2   9    52    43
TRAINF    808  402   3.103098     0.335E-13 0.000E+00   0   4    34    34
TRAINH    808  402   12.30984     0.613E-13 0.118E-07  17  10   840   731
UBH5     1010  700   1.116324     0.125E-10 0.173E-06 297  52  5029  4580
VANDERM1  100  199   0.000000E+00 0.832E-03 0.119E-07   2 105  4188  3447 fail 6
VANDERM2  100  199   0.000000E+00 0.832E-03 0.119E-07   2 105  4188  3447 fail 6
VANDERM3  100  199   0.000000E+00 0.732E-06 0.337E-09   6  22   747   637
VANDERM4    9   17   0.166604E-06 0.820E+02 0.132E-03   1  56   739   598 fail 3
ZAMB2    1326  480  -4.142201     0.835E-12 0.781E-05 214  55  6219  5291
ZIGZAG    604  500   52.30551     0.260E-12 0.000E+00   0  12   501   497

Summary for COPS problems in CUTE
problem     n    m     f^*           h^*     rgnorm     k itn    #f    #g ifail
-------------------------------------------------------------------------------
CAMSHAPE  800 1603  -4.273832     0.691E-13 0.000E+00   0   8   170   170
CATMIX   2403 1600  -0.463903E-01 0.343E-13 0.672E-05   1   8  1268   534
CHAIN     802  401  -0.135161E+07 0.238E+04 0.141E-01 315 999166274101039 fail 5
CHAIN with mxgr=400
          802  401   5.068622     0.145E-13 0.178E-06 399  13  3825  3467
CHANNEL  9600 9598   1.000000     0.155E-10 0.000E+00 601  13   254   249
CHANNEL with mxgr=400
         9600 9598   1.000000     0.685E-11 0.000E+009576   7   807   807
ELEC      600  200   0.474442E+10 0.300E-01       NaN 402 165 20577 15682 fail20
ELEC with ubd=1.D0
          600  200   18439.12     0.444E-11 0.620E-03 400  43  4387  4172
GASOIL  1040310398   0.523660E-02 0.565E-12 0.515E-07   3  20   368   336
GLIDER   5214 4808  -0.149907E+03 0.684E+02 0.110E-04  80 999101009100899 fail 5
MARINE  1121511192   0.197466E+08 0.288E-08 0.667E+01  21 201 19124 15658
MARINE with mxgr=400
        1121511192   0.197465E+08 0.275E-08 0.304E+01  22  92 22797 21312
METHANOL1200511997   0.902229E-02 0.343E-12 0.559E-05   4   5    49    46
MINSURFO 5306    0   2.506949     0.000E+00 0.897E-053544   7   603   533
PINENE   8805 8795   19.87217     0.321E-11 0.179E-02   5  11   197   174
POLYGON   200 5049  -0.784248E+00 0.537E-14 0.212E-03  88  52  5161  4514
ROBOTARM 4412 3202   9.141026     0.198E-11 0.000E+00   0  23  1790  1790
ROCKET   2407 2002  -1.012836     0.224E-11 0.260E-07  18 239 24195 23640
ROCKET with mxgr=400
         2407 2002  -1.012836     0.238E-11 0.943E-08  27  26  4660  4386
STEERING 2006 1600   0.554572E+00 0.809E-12 0.400E-05 399 180 20178 17925
STEERING with tol=-1.D-8
         2006 1600   0.554572E+00 0.781E-12 0.993E-05 399 107 11766 10460
```

REFERENCES

[1] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz, *An active-set algorithm for nonlinear programming using linear programming and equality constrained subproblems*, Math. Programming B, 100 (2004), pp. 27–48.

[2] C. M. Chin and R. Fletcher, *On the global convergence of an SLP-filter algorithm that takes EQP steps*, Math. Programming, 96 (2003), pp. 161–177.

[3] R. Fletcher, *Practical Methods of Optimization*, Wiley, Chichester, 1987.

[4] R. Fletcher, *Dense factors of sparse matrices*, in Approximation Theory and Optimization, M. D. Buhmann and A. Iserles, eds., Cambridge University Press, Cambridge, 1997.

[5] R. Fletcher, *Block triangular orderings and factors for sparse matrices in LP*, in Numerical analysis 1997, D. F. Griffiths, D. J. Higham, and G. A. Watson, eds., Pitman Res. Notes in Math. 380, Longman, Harlow, 1998, pp. 91–110.

[6] R. Fletcher, *A limited memory steepest descent method*, Math. Programming, DOI 10.1007/s10107-011-0479-6, (2011), 26 pp.

[7] R. Fletcher and S. Leyffer, *Nonlinear programming without a penalty function*, Math. Programming, 91 (2002), pp. 239–270.

[8] R. Fletcher, S. Leyffer, and Ph. L. Toint, *On the global convergence of a filter-SQP algorithm*, SIAM J. Optim., 13 (2002), pp. 44–59.
[9] R. Fletcher and E. Sainz de la Maza, *Nonlinear programming and nonsmooth optimization by successive linear programming*, Math. Programming, 43 (1989), pp. 235–256.
[10] B. A. Murtagh and M. A. Saunders, *A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints*, Math. Programming Stud., 16 (1982), pp. 84–117.
[11] S. M. Robinson, *A quadratically convergent method for general nonlinear programming problems*, Math. Programming, 3 (1972), pp. 145–156.
[12] P. Wolfe, *A technique for resolving degeneracy in linear programming*, J. Soc. Indust. Appl. Math., 11 (1963), pp. 205–211.