Numerical Performance of an SLP–Filter Algorithm that takes EQP steps

Choong Ming Chin and Roger Fletcher*

**Department of Mathematics, University of Dundee, Dundee DD1 4HN, Scotland, UK.*

Numerical Analysis Report NA/202, August 2001

Abstract

This paper describes the implementation of a trust region based SLP filter algorithm that takes EQP steps. The prototype code SLPSQP is a particular implementation and it is interfaced with CPLEX version 4.0 to solve linear programming subproblems. To obtain EQP steps, we exploit the CPLEX callable library facilities. Computational results on a wide range of CUTE test problems are very encouraging and numerical comparisons with filterSQP and LANCELOT show that the algorithm is also efficient and reliable.

Keywords nonlinear programming, filter, SLP, EQP, filterSQP, LANCELOT.

1 Introduction

In this paper we present some numerical results obtained by solving a selection of CUTE test problems using a trust region based SLP filter algorithm that takes EQP steps. For the theory behind the algorithm, see Chin and Fletcher [2]. The prototype code SLPSQP solves Nonlinear Programming (NLP) problems of the following form

$$P \left\{ \begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} & f(\mathbf{x}) \\ \text{subject to} & c_i(\mathbf{x}) \leq 0 \ , \quad i = 1, 2, \ldots, m \end{array} \right.$$

where $f(\mathbf{x})$ is a linear or nonlinear objective function and $c_i(\mathbf{x})$, $i = 1, 2, \ldots, m$ is a linear or nonlinear constraint.

As our algorithm uses the EQP active set strategy therefore we first selected 50 quadratic programming problems from the CUTE suite to be tested. The reason for solving this class of problems is that QP problems are the simplest NLPs where they often give useful insight into the behaviour of our algorithm. For the complete numerical results, see Chin [1]. In addition, since our main objective in this study is to use the filter strategy to solve Problem P efficiently, therefore in the next set of problems each test problem chosen has at least one nonlinear constraint. To test the performance of our algorithm, we have selected 300 CUTE test problems and we divide the problems into two groups. The first group consists of 200 small scale problems and the remaining problems are medium and large scale problems. For further information concerning the dimension as well as the number of constraints each problem has, see Chin [1].

In the tests, exact first and second derivatives are used. In order to indicate the potential of the algorithm, the prototype code SLPSQP is compared with filterSQP (see

Fletcher and Leyffer [5]) and LANCELOT (see Conn, Gould and Toint [3]). The main reason for choosing these two packages is that both filterSQP and LANCELOT use first and second derivatives in their computations. Thus, a more realistic comparison can then be made in terms of reliability and efficiency of our algorithm.

We organize this paper as follows. In Section 2, we describe the filter algorithm on which our code is based whilst the implementation details of the code are given in Section 3. Finally in Section 4, we present the numerical results from applying SLPSQP to the selected test problems, and the numerical comparison for the three codes is also presented.

2 The SLPSQP Algorithm

The main objective in developing this alternative method is to dispense with the idea of using penalty functions to induce convergence for NLP. In our approach, a filter method is used where a step generated from solving a subproblem is accepted so long as it produces a sufficient decrease in either the objective function or in a constraint violation function. This approach can be seen as an advantage since the filter method allows a certain amount of non-monotonicity compared to a penalty function approach. In fact, this filter idea is not entirely new as it is first introduced for NLP by Fletcher and Leyffer [4] where they solve Problem P by forming a sequence of QP-like subproblems and then use the filter as a means to decide the suitability of the steps being generated.

Instead of using the SQP method as a basic iterative framework in conjunction with the filter approach, the method we propose is to solve Problem P iteratively in two stages. The first subproblem to be solved is a linear programming (LP) subproblem constructed using only first order Taylor series approximations on the objective function and the nonlinear constraints. In addition, we also incorporate an ℓ_∞ trust region constraint (i.e. by adding bounds on the variables) in the LP subproblem so as to control the size of the step generated at each iteration. To compensate for the loss of second order information and also to obtain rapid convergence of the algorithm, an equality based QP subproblem is solved in the second stage using only a subset of linearized constraints which are equalities at the solution of the LP subproblem.

The major reasons for considering solving a sequence of LP and QP-like subproblems on each iteration are:

- Solving an LP subproblem in general is computationally less intensive than solving an inequality based QP subproblem since we do not have to update a reduced Hessian matrix if active set method is used. Hence, it is particularly attractive for solving large, sparse nonlinear programs. Moreover, by adding a trust region constraint to the LP subproblem, asymptotically we can locate accurate estimates of the nonlinear constraints which are equalities at the solution.

- There are no iterations required to solve an equality based QP subproblem. Hence, this reduces the computational and informational requirements to calculate the QP step.

- The use of second order information usually exhibits rapid convergence when the QP step is used near the solution. Thus the efficiency of the proposed method can be enhanced.

An important feature of this study is the way in which the computation of the QP step is interfaced with the technique of solving an LP-like subproblem. Although solving trust region LP subproblems can give us asymptotically accurate estimates of nonlinear constraints which are active at the solution, the LP step is usually not used directly in the filter test because of the possibility of slow convergence. Instead, the step found from solving the LP subproblem is only used if the QP step is rejected by the filter and if this step is also unsuccessful then the algorithm reduces the trust region radius.

Following such a strategy, we are now in a position to describe our algorithm in greater detail. At the k-th iteration, the LP subproblem at the current point $\mathbf{x}_k$ and a current trust region radius ρ is defined as

$$LP(\mathbf{x}_k,\rho)\left\{\begin{array}{ll}\underset{\mathbf{d}\in\mathbb{R}^n}{\text{minimize}} & \nabla f(\mathbf{x}_k)^T\mathbf{d} \\ \text{subject to} & \nabla c_i(\mathbf{x}_k)^T\mathbf{d}+c_i(\mathbf{x}_k)\leq 0\ , \quad i=1,2,\ldots,m \\ & \|\mathbf{d}\|_\infty\leq\rho\end{array}\right.$$

and we denote the solution (if it exists) of $LP(\mathbf{x}_k,\rho)$ as $\mathbf{d}^{LP}$. The linearized predicted reduction given by $\mathbf{d}^{LP}$ is defined as

$$\Delta l=-\nabla f(\mathbf{x}_k)^T\mathbf{d}^{LP}.$$

To make use of second order information, the next step is to solve a QP subproblem of the form

$$QP(\mathbf{x}_k,\infty)\left\{\begin{array}{ll}\underset{\mathbf{d}\in\mathbb{R}^n}{\text{minimize}} & \nabla f(\mathbf{x}_k)^T\mathbf{d}+\frac{1}{2}\mathbf{d}^T\mathbf{W}(\mathbf{x}_k,\boldsymbol{\lambda}^{LP})\mathbf{d} \\ \text{subject to} & \nabla c_i(\mathbf{x}_k)^T\mathbf{d}+c_i(\mathbf{x}_k)=0\ , \qquad i\in\mathcal{A}(\mathbf{x}_k)\end{array}\right.$$

to obtain $\mathbf{d}^{QP}$. The working set $\mathcal{A}(\mathbf{x}_k)$ is defined as

$$\mathcal{A}(\mathbf{x}_k)=\{i\ :\ \nabla c_i(\mathbf{x}_k)^T\mathbf{d}^{LP}+c_i(\mathbf{x}_k)=0\}$$

and the Hessian matrix matrix $\mathbf{W}(\mathbf{x}_k,\boldsymbol{\lambda}^{LP})$ we use in the QP subproblem is the Hessian of the Lagrangian function and is defined as

$$\mathbf{W}(\mathbf{x}_k,\boldsymbol{\lambda}^{LP})=\nabla^2 f(\mathbf{x}_k)+\sum_{i\in\mathcal{A}(\mathbf{x}_k)}\lambda_i^{LP}\nabla^2 c_i(\mathbf{x}_k)$$

where λ_i^{LP}, $i\in\mathcal{A}(\mathbf{x}_k)$ are the working set Lagrange multipliers found from $LP(\mathbf{x}_k,\rho)$.

As in most SQP codes that use a non-differentiable penalty function, our technique may also reject unit QP steps as the iterates approach the solution. This is commonly referred to as the Maratos effect. To help circumvent this problem, Second Order Correction (SOC) steps may be computed to keep the current point close to the feasible region. Hence in the filter algorithm we also sample SOC steps if the QP steps fail to be acceptable to the filter. The advantage of computing such a step is that it provides an extra opportunity for the algorithm to accept a step within the current trust region radius ρ, thus reducing the likelihood of having to reduce ρ. Moreover, small values of ρ can lead to slow convergence and the possibility of entering the restoration phase is much higher. In addition, computing an SOC step is also quite cheap to implement by

exploiting CPLEX callable library funtions. Conceptually, we do this by computing a step $\mathbf{v}$ by solving the following linear equations

$$\nabla c_i(\mathbf{x}_k)^T\mathbf{v} + c_i(\mathbf{x}_k + \mathbf{d}^{QP}) = 0 \quad i \in \mathcal{A}(\mathbf{x}_k)$$

and the SOC step $\mathbf{d}^{SOC}$ is then defined as

$$\mathbf{d}^{SOC} = \mathbf{d}^{QP} + \mathbf{v}.$$

If a sequence of $\mathbf{d}^{QP}$ and $\mathbf{d}^{SOC}$ steps are unsatisfactory then a step derived from $\mathbf{d}^{LP}$ is used. Therefore for any displacement $\mathbf{d}$, we denote

$$\Delta f = f(\mathbf{x}_k) - f(\mathbf{x}_k + \mathbf{d})$$

as the *actual reduction* in $f(\mathbf{x}_k)$, and

$$\Delta q = -\nabla f(\mathbf{x}_k)^T\mathbf{d} - \frac{1}{2}\mathbf{d}^T\mathbf{W}(\mathbf{x}_k, \boldsymbol{\lambda}^{LP})\mathbf{d}$$

as the *quadratic predicted reduction* in $f(\mathbf{x}_k)$. In the algorithm, we do not calculate Δq directly but a bound can be obtained by making use of the so-called *Cauchy step*, defined by

$$\mathbf{d}^C = \alpha_c \mathbf{d}^{LP}$$

where $\alpha_c \in [0,1]$ is chosen to maximize Δq for $\mathbf{d} = \alpha_c \mathbf{d}^{LP}$. By denoting Δq^C as the quadratic predicted reduction for the Cauchy step, if $\Delta l > 0$ then we can set

$$\Delta q^C \begin{cases} \geq \frac{1}{2}\Delta l & \text{if } \Delta l \geq b, \\ = \frac{1}{2}\alpha_c \Delta l & \text{if } \alpha_c = \Delta l / b < 1. \end{cases}$$

where $b = (\mathbf{d}^{LP})^T\mathbf{W}(\mathbf{x}_k, \boldsymbol{\lambda}^{LP})\mathbf{d}^{LP}$. For further information in obtaining this result, see Chin and Fletcher [2].

We have now discussed the main ingredients of the class of methods that we are considering and a description of the filter algorithm is given as follows.

The Main Filter Algorithm

Given $\mathbf{x}_0$, $\rho_{\text{ini}} \geq \rho_{\min} > 0$, set $\rho = \rho_{\text{ini}} > 0$, $\eta \in (0,1)$, $\eta_1 \in (0,1)$, $\gamma \in (0,1)$ and $\delta > 1$. Set $k := 0$ and

$$u = \begin{cases} \delta h(\mathbf{c}(\mathbf{x}_0)) & \text{if } h(\mathbf{c}(\mathbf{x}_0)) > 1, \\ \delta & \text{otherwise.} \end{cases}$$

where $h(\mathbf{c}(\mathbf{x})) = \sum_{i=1}^{m} \max\{0, c_i(\mathbf{x})\}$.

If $h(\mathbf{c}(\mathbf{x}_0)) > 0$ then put $(h(\mathbf{c}(\mathbf{x}_0)), f(\mathbf{x}_0))$ into the filter and set $\mathcal{F}^{(k)} := \{0\}$.

Step 1 Solve $LP(\mathbf{x}_k, \rho)$ subproblem to obtain $\mathbf{d}^{LP}$.

Step 2 **If** $LP(\mathbf{x}_k, \rho)$ subproblem does not have a feasible solution **Then**

- Goto Feasibility Restoration Phase.

Endif

Step 3 Solve $QP(\mathbf{x}_k, \infty)$ subproblem to obtain $\mathbf{d}^{QP}$.

Set $\mathbf{d}^{QP}$ as a feasible point with respect to all linearized constraints.

Set $i := 1$.

Step 4 **If** $i = 1$ **Then**

- Set $\mathbf{d} = \mathbf{d}^{QP}$.

Else If $i = 2$ **Then**

- Construct an SOC step, $\mathbf{d}^{SOC}$.
- Set $\mathbf{d} = \mathbf{d}^{SOC}$.

Else

- Set $\mathbf{d} = \alpha_c \mathbf{d}^{LP}$, the Cauchy step.

Endif

Step 5 **If** *convergence criterion* is met **Then**

- STOP.

Endif

Step 6 Let $\widetilde{\mathbf{x}} = \mathbf{x}_k + \mathbf{d}$.

If $\{h(\mathbf{c}(\widetilde{\mathbf{x}})) \leq (1-\eta)h(\mathbf{c}(\mathbf{x}_i))$ **or** $f(\widetilde{\mathbf{x}}) \leq f(\mathbf{x}_i) - \gamma h(\mathbf{c}(\widetilde{\mathbf{x}}))\}$ for all $i \in \mathcal{F}^{(k)}$ **and** $h(\mathbf{c}(\widetilde{\mathbf{x}})) \leq u$ **Then**

- **if** $\Delta l > 0$ and $\Delta f < \eta_1 \Delta q^C$ **then**
  - Goto Step 8.
- **else**
  - Goto Step 7.
- **endif**

Else

- Goto Step 8.

Endif

Step 7 Set $\mathbf{d}_k = \mathbf{d}$, $\rho_k = \rho$, $\Delta l_k = \Delta l$, $\Delta q_k^C = \Delta q^C$ and $\Delta f_k = \Delta f$.

Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k$.

Set $\Delta = \begin{cases} \rho + \|\mathbf{d}\|_\infty & \text{if } \|\mathbf{d}\|_\infty \geq \rho, \\ \rho & \text{otherwise.} \end{cases}$

Set $\rho = \max\{\Delta, \rho_{\min}\}$.

If $h(\mathbf{c}(\mathbf{x}_{k+1})) > 0$ **Then**

- Put $(h(\mathbf{c}(\mathbf{x}_{k+1})), f(\mathbf{x}_{k+1}))$ into the filter. Set $k+1 \in \mathcal{F}^{(k+1)}$.
- Remove points from the filter that are dominated by $(h(\mathbf{c}(\mathbf{x}_{k+1})), f(\mathbf{x}_{k+1}))$.

Endif

Set $k := k+1$ and return to Step 1.

Step 8 Set $i := i + 1$.

If $i \leq 3$ **Then**

- Goto Step 4.

Endif

Step 9 Set $\rho := \frac{1}{2}\rho$ and return to Step 1.

2.1 Infeasible Trust Region LP Subproblem

As described in the filter algorithm, unless $h(\mathbf{c}(\mathbf{x})) = 0$, then by repeatedly reducing the trust region ρ in Step 9 then there is a strong possibility that the current LP subproblem could be incompatible. Less frequently however, is the possibility that the linearizations of the nonlinear constraints may themselves be inconsistent. Therefore provision for continuing the algorithm can be made by entering a *feasibility restoration phase* in which the intention is to get close to the boundary of the feasibility region by reducing the constraint infeasibilities. It is desirable that this phase should terminate by finding a new point $\mathbf{x}_k$ for some $k \geq K > 0$ such that the subproblem $LP(\mathbf{x}_k, \rho)$ is feasible.

In the paper by Fletcher and Leyffer [4], a technique is introduced to reduce the constraint infeasibilities by applying a trust region SQP method to the following problem

$$\begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} & \sum_{i\in J} c_i(\mathbf{x}) \\ \text{subject to} & c_i(\mathbf{x}) \leq 0 \ , \quad i \in J^{\perp} \end{array}$$

where $J^{\perp} \subset \{1, 2, \ldots, m\}$ such that

$$\{\mathbf{d} \ : \ \nabla c_i(\mathbf{x})^T\mathbf{d} + c_i(\mathbf{x}) \leq 0, \ i \in J^{\perp}\} \cap \{\mathbf{d} \ : \ \|\mathbf{d}\|_\infty \leq \rho\} \neq \emptyset$$

and $J = \{1, 2, \ldots, m\} \backslash J^{\perp}$. Acceptance of a step is then based on using a *restoration filter* or *phase I filter* in which the filter consists of pairs $(h_J, h_{J^{\perp}})$ such that

$$\begin{aligned} h_J &:= h(\mathbf{c}_J(\mathbf{x})) \\ &= \sum_{i\in J} \max\{0, c_i(\mathbf{x})\} \end{aligned}$$

and

$$\begin{aligned} h_{J^{\perp}} &:= h(\mathbf{c}_{J^{\perp}}(\mathbf{x})) \\ &= \sum_{i\in J^{\perp}} \max\{0, c_i(\mathbf{x})\}. \end{aligned}$$

Although the use of a filter allows greater flexibility in deciding the steps to be accepted, nevertheless the determination of the sets J and $J^{\perp}$ at every iteration might not be straightforward.

In our technique to reduce constraint infeasiblities the main strategy is to apply a trust region method to solve the following problem

$$F \left\{ \begin{array}{ll} \underset{\mathbf{x}\in\mathbb{R}^n}{\text{minimize}} & \sum_{i\in\mathcal{J}} c_i(\mathbf{x}) \\ \text{subject to} & c_i(\mathbf{x}) \leq 0 \ , \quad i \in \mathcal{J}^{\perp} \end{array} \right.$$

where $\mathcal{J} = \mathcal{J}(\mathbf{x}) = \{i \ : \ c_i(\mathbf{x}) > 0\}$ and $\mathcal{J}^{\perp} = \mathcal{J}^{\perp}(\mathbf{x}) = \{i \ : \ c_i(\mathbf{x}) \leq 0\}$. By solving Problem F, the concept of a restoration filter is therefore not needed as we can just use a simple reduction test to assess a step generated from a subproblem.

In analogy with the main SLPSQP filter algorithm, the feasibility restoration phase is also solved in two stages. At the k-th iteration, the LP subproblem at the point $\mathbf{x}_k$ and a current trust region radius ρ takes the form

$$\widetilde{LP}(\mathbf{x}_k, \rho) \left\{ \begin{array}{ll} \underset{\mathbf{d} \in \mathbb{R}^n}{\text{minimize}} & \sum_{i \in \mathcal{J}_k} \nabla c_i(\mathbf{x}_k)^T \mathbf{d} \\ \text{subject to} & \nabla c_i(\mathbf{x}_k)^T \mathbf{d} + c_i(\mathbf{x}_k) \leq 0 \ , i \in \mathcal{J}_k^{\perp} \\ & \|\mathbf{d}\|_{\infty} \leq \rho \end{array} \right.$$

where $\mathcal{J}_k = \{i : c_i(\mathbf{x}_k) > 0\}$ and $\mathcal{J}_k^{\perp} = \{i : c_i(\mathbf{x}_k) \leq 0\}$. As the above subproblem is always feasible, we denote the solution of $\widetilde{LP}(\mathbf{x}_k, \rho)$ as $\widehat{\mathbf{d}}^{LP}$ and the *linearized predicted reduction* at $\widehat{\mathbf{d}}^{LP}$ as

$$\Delta \widehat{l} \ = \ - \sum_{i \in \mathcal{J}_k} \nabla c_i(\mathbf{x}_k)^T \widehat{\mathbf{d}}^{LP}.$$

To compensate for the loss of second order information, the next stage is to solve a QP subproblem via an EQP subproblem of the form

$$\widetilde{QP}(\mathbf{x}_k, \infty) \left\{ \begin{array}{ll} \underset{\mathbf{d} \in \mathbb{R}^n}{\text{minimize}} & \sum_{i \in \mathcal{J}_k} \nabla c_i(\mathbf{x}_k)^T \mathbf{d} + \frac{1}{2} \mathbf{d}^T \mathbf{B}(\mathbf{x}_k, \boldsymbol{\mu}^{LP}) \mathbf{d} \\ \text{subject to} & \nabla c_i(\mathbf{x}_k)^T \mathbf{d} + c_i(\mathbf{x}_k) = 0 \ , i \in \widehat{\mathcal{A}}(\mathbf{x}_k) \end{array} \right.$$

and we let $\widehat{\mathbf{d}}^{QP}$ be defined as the solution of $\widetilde{QP}(\mathbf{x}_k, \infty)$. As in the main filter algorithm, we define the QP working set $\widehat{\mathcal{A}}(\mathbf{x}_k)$ by

$$\widehat{\mathcal{A}}(\mathbf{x}_k) = \{i \in \mathcal{J}_k^{\perp} : \nabla c_i(\mathbf{x}_k)^T \widehat{\mathbf{d}}^{LP} + c_i(\mathbf{x}_k) = 0\}.$$

In addition, the Hessian matrix $\mathbf{B}(\mathbf{x}_k, \boldsymbol{\mu}^{LP})$ used is the Hessian of the Lagrangian of Problem F given by

$$\mathbf{B}(\mathbf{x}_k, \boldsymbol{\mu}^{LP}) = \sum_{i \in \mathcal{J}_k} \nabla^2 c_i(\mathbf{x}_k) + \sum_{i \in \widehat{\mathcal{A}}(\mathbf{x}_k)} \mu_i^{LP} \nabla^2 c_i(\mathbf{x}_k)$$

where μ_i^{LP}, $i \in \widehat{\mathcal{A}}(\mathbf{x}_k)$ are the working set Lagrange multipliers determined from solving $\widetilde{LP}(\mathbf{x}_k, \rho)$.

As in the main filter algorithm, we also allow SOC steps to be sampled if the restoration phase QP step, $\widehat{\mathbf{d}}^{QP}$ is rejected by the algorithm. Hence, we define the restoration phase SOC step as

$$\widehat{\mathbf{d}}^{SOC} \ = \ \widehat{\mathbf{d}}^{QP} + \widehat{\mathbf{v}}$$

where $\widehat{\mathbf{v}}$ is found by solving a set of linear equations of the form

$$\nabla c_i(\mathbf{x}_k)^T \widehat{\mathbf{v}} + c_i(\mathbf{x}_k + \widehat{\mathbf{d}}^{QP}) \ = \ 0 \quad i \in \widehat{\mathcal{A}}(\mathbf{x}_k).$$

For any displacement $\widehat{\mathbf{d}}$, we denote the *actual reduction* in $h(\mathbf{c}(\mathbf{x}_k))$ as

$$\Delta h \;=\; h(\mathbf{c}(\mathbf{x}_k)) - h(\mathbf{c}(\mathbf{x}_k + \widehat{\mathbf{d}}))$$

and the *quadratic predicted reduction* as

$$\Delta\widehat{q} \;=\; -\sum_{i\in\mathcal{J}_k} \nabla c_i(\mathbf{x}_k)^T\widehat{\mathbf{d}} - \frac{1}{2}\widehat{\mathbf{d}}^T\mathbf{B}(\mathbf{x}_k, \boldsymbol{\mu}^{LP})\widehat{\mathbf{d}}.$$

In analogy with the filter algorithm, there is no need to calculate $\Delta\widehat{q}$ directly but a bound can also be obtained by using the *Cauchy step* defined by

$$\widehat{\mathbf{d}}^C \;=\; \hat{\alpha}_c\widehat{\mathbf{d}}^{LP}$$

where $\hat{\alpha}_c \in [0,1]$ is chosen to maximize $\Delta\widehat{q}$ for $\widehat{\mathbf{d}} = \hat{\alpha}_c\widehat{\mathbf{d}}^C$. By substituting Δl with $\Delta\widehat{l}$, the quadratic reduction for the Cauchy step, $\Delta\widehat{q}^C$ can then be written as

$$\Delta\widehat{q}^C \begin{cases} \geq \frac{1}{2}\Delta\widehat{l} & \text{if } \Delta\widehat{l} \geq \hat{b}, \\ = \frac{1}{2}\hat{\alpha}_c\Delta\widehat{l} & \text{if } \hat{\alpha}_c = \Delta\widehat{l}/\hat{b} < 1. \end{cases}$$

where $\hat{b} = (\widehat{\mathbf{d}}^{LP})^T\mathbf{B}(\mathbf{x}_k, \boldsymbol{\mu}^{LP})\widehat{\mathbf{d}}^{LP}$. Thus, our sufficient reduction test takes the form

$$\Delta h \geq \eta_2\Delta\widehat{q}^C$$

where $\eta_2 \in (0,1)$ and it follows from $\Delta\widehat{l} > 0$ that $\Delta\widehat{q}^C > 0$.

With all the required concepts and explanation we are now able to examine the feasibility restoration phase algorithm in greater detail by means of the following pseudo-code.

Feasibility Restoration Phase Algorithm

Given $\mathbf{x}_k$, ρ and $\eta_2 \in (0,1)$.

Step 1 Solve $\widetilde{LP}(\mathbf{x}_k, \rho)$ subproblem to obtain $\widehat{\mathbf{d}}^{LP}$.

Step 2 Solve $\widetilde{QP}(\mathbf{x}_k, \infty)$ subproblem to obtain $\widehat{\mathbf{d}}^{QP}$.

Set $\widehat{\mathbf{d}}^{QP}$ as a feasible point with respect to all linearized constraints in $\mathcal{J}_k^{\perp}$.

Set $i := 1$.

Step 3 **If** $i = 1$ **Then**

- Set $\widehat{\mathbf{d}} = \widehat{\mathbf{d}}^{QP}$.

Else If $i = 2$ **Then**

- Construct an SOC step, $\widehat{\mathbf{d}}^{SOC}$.
- Set $\widehat{\mathbf{d}} = \widehat{\mathbf{d}}^{SOC}$.

Else

- Set $\widehat{\mathbf{d}} = \hat{\alpha}_c\widehat{\mathbf{d}}^{LP}$.

Endif

Step 4 **If** *convergence criterion* is met **Then**

- STOP.

Endif

Step 5 **If** $\Delta h \geq \eta_2 \Delta \widehat{q}^C$ **Then**

- Goto Step 6.

Else

- Set $i := i + 1$.
- **if** $i \leq 3$ **then**
  - Goto Step 3.
- **else**
  - Goto Step 7.
- **endif**

Endif

Step 6 Set $\widehat{\mathbf{d}}_k = \widehat{\mathbf{d}}$, $\rho_k = \rho$, $\Delta h_k = \Delta h$, $\Delta \widehat{q}_k^C = \Delta \widehat{q}^C$ and $\Delta \widehat{l}_k = \Delta \widehat{l}$.

Set $\mathbf{x}_{k+1} = \mathbf{x}_k + \widehat{\mathbf{d}}_k$.

Set $\Delta = \begin{cases} \rho + \|\widehat{\mathbf{d}}\|_\infty & \text{if } \|\widehat{\mathbf{d}}\|_\infty \geq \rho, \\ \rho & \text{otherwise.} \end{cases}$

Set $\rho_f = \max\{\Delta, \rho_{\min}\}$.

Set $k := k + 1$.

If $(h(\mathbf{c}(\mathbf{x}_k)), f(\mathbf{x}_k))$ is acceptable into the filter **Then**

- **if** $h(\mathbf{c}(\mathbf{x}_k)) > 0$ **then**
  - Put $(h(\mathbf{c}(\mathbf{x}_k)), f(\mathbf{x}_k))$ into the filter.
  - Set $k \in \mathcal{F}^{(k)}$.
  - Remove points from the filter that are dominated by $(h(\mathbf{c}(\mathbf{x}_k)), f(\mathbf{x}_k))$.
- **endif**
- Set $\rho = \max\{\rho_{\text{ini}}, \rho_f\}$.
- Return to filter algorithm.

Else

- Set $\rho = \rho_f$ and return to Step 1.

Endif

Step 7 Set $\rho := \frac{1}{2}\rho$ and return to Step 1.

3 Implementation Details

Details regarding the implementation of the algorithm are listed below.

- The prototype code SLPSQP has been implemented in FORTRAN with double precision. In addition, the code is also interfaced with CPLEX version 4.0 and the test problems are solved on a SPARCstation Ultra 10 with 256 Mb memory under Solaris 7.

- For initial trust region radius we choose $\rho_{\text{ini}} = 5$. As for parameter values, we set $\delta = 10$, $\eta = 10^{-3}$, $\eta_1 = 10^{-2}$, $\eta_2 = 10^{-3}$, $\gamma = 10^{-3}$ and $\rho_{\min} = 10^{-4}$. In addition, we set the tolerance level $\epsilon_{\text{tol}} = 10^{-6}$ and the maximum iterations permitted is $k_{\max} = 1000$.

- As for convergence criteria, the KKT error, the constraint violation $h(\mathbf{c}(\mathbf{x}))$ and the ℓ_∞ norm of the step are computed. We terminate the iteration when the above conditions are satisfied to an accuracy of ϵ_{tol} (unless $k = k_{\max}$).

- The algorithm may also terminate in the feasibility restoration phase and this occurs when the restoration phase algorithm can no longer reduce the constraint infeasibility function. Thus, the algorithm stops if either one of these conditions is true:

  (a) the trust region radius $\rho < \epsilon_{\text{tol}}$,

  (b) the iteration $k = k_{\max}$,

  (c) the KKT error of Problem F and the ℓ_∞ of the step are less than ϵ_{tol}.

- At the initial stage, the code SLPSQP always starts by finding a point which satisfies only the linear constraints (including the simple bounds on the variables) of Problem P. If the linear constraints are found to be inconsistent, the program terminates with an indication that the problem constraints have no feasible solution.

- Provided $\|\mathbf{d}^{QP}\|_\infty > \rho$ or $\|\mathbf{d}^{SOC}\|_\infty > \rho$, we repeat the QP or SOC steps a finite number of times, before falling back on the Cauchy step. This is achieved by imposing a bound on the QP or SOC steps so that new trial steps of the form $\mathbf{d} = \dfrac{\rho}{\|\mathbf{d}^{QP}\|_\infty}\mathbf{d}^{QP}$ or $\mathbf{d} = \dfrac{\rho}{\|\mathbf{d}^{SOC}\|_\infty}\mathbf{d}^{SOC}$ are also tested where $\|\mathbf{d}\|_\infty = \rho$. By using such a strategy of sampling more steps, it is hoped that it reduces the likelihood of reducing the current trust region radius and also entering the feasibility restoration phase.

4 Numerical Performance of SLPSQP

In this section, we describe the numerical performance of our code on a selection of CUTE test problems. As the number of test problems chosen is quite substantial and to avoid the interruption of the flow of presentation, the complete numerical results are given in a separate report (see Chin [1]).

First of all, from the numerical results on QP problems given in Chin [1], a majority of QP problems solved required fewer than 20 iterations. Therefore we feel that our

algorithm which uses the EQP active set strategy is quite efficient, and the correct active set is obtained very quickly even though it is determined by solving LP subproblems. Next, to assess the effectiveness of the filter strategy we discuss the numerical results on 300 NLP problems. As regards on the performance of the algorithm on NLP problems, we feel that it is very encouraging as the filter code is able to solve 284 problems out of 300 test problems which translates to a rate of success of 94.67%. From the set of 300 test problems of varying sizes and complexities, SLPSQP failed to solve only 16 problems and the reasons for failure are given as follows:

- The trust region LP subproblem was found to be infeasible (but with $h(\mathbf{c}(\mathbf{x}_k)) < \epsilon_{\text{tol}}$) on 1 problem (SVANBERG ($n = 500$)).
- SLPSQP converged to a local minimum of Problem F in the restoration phase on 6 problems (DISCS, HIMMELBD, MINMAXBD, NYSTROM5, OPTCDEG3 and POLAK3);
- SLPSQP terminated with $\rho < \epsilon_{\text{tol}}$ on 9 problems (ARTIF ($n = 12$), ARTIF ($n = 1002$), CRESC50, DRUGDISE, HS101R, LEWISPOL, MANNE ($n = 1095$), POWELLSQ and ROSENMMX). Note that in this case the restoration phase converged to a non-KKT point of Problem F for these test problems.

Another question of some interest is the size of the filter needed to solve a particular test problem. In Figure 4.1, we present the distribution of filter sizes against the number of problems solved by SLPSQP.

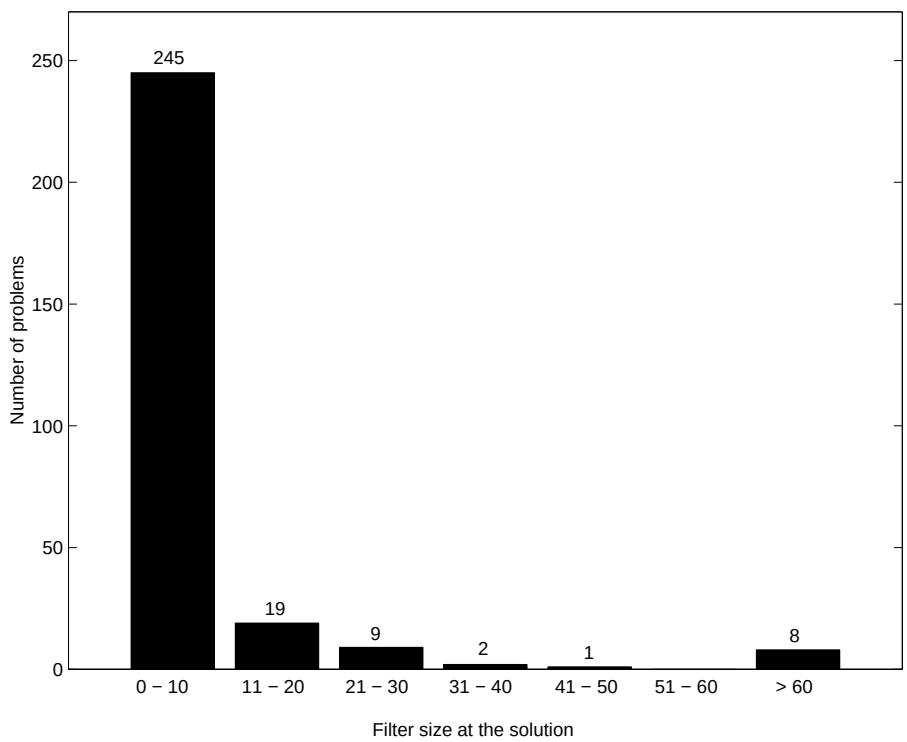

Figure 4.1: Distribution of filter sizes of successful run test problems

From Figure 4.1, we can see that a majority of the problems solved required filter size which is less than 10 and only a small proportion of test problems required filter size which is greater than 60.

Next, we will analyze the performance of our algorithm further by comparing it with filterSQP and LANCELOT. The analysis shows that SLPSQP is also as reliable and efficient as filterSQP and LANCELOT. We do this by comparing the number of gradient evaluations and CPU time in seconds needed to solve a particular CUTE problem. For the default versions of filterSQP and LANCELOT, we note that

$$\rho_{\text{ini}} = \begin{cases} 10.0 & \text{for filterSQP,} \\ 1.0 & \text{for LANCELOT} \end{cases}$$

$$k_{\max} = \begin{cases} 1000 & \text{for filterSQP,} \\ 10000 & \text{for LANCELOT} \end{cases}$$

and

$$\epsilon_{\text{tol}} = \begin{cases} 10^{-6} & \text{for filterSQP,} \\ 10^{-5} & \text{for LANCELOT.} \end{cases}$$

As we will only discuss the statistics concerning the performance of the three codes in this section, the complete comparison results are given in Chin [1].

We first compare the reliability factor of the three codes. To simplify the presentation, if either SLPSQP, filterSQP or LANCELOT terminates before reaching a local solution, the following notations are used:

- E - An arithmetic error occurred causing the code to fail,
- H - The subproblem was found to be infeasible although $h(\mathbf{c}(\mathbf{x})) \leq \epsilon_{\text{tol}}$,
- I - Nonlinear constraints were found to be locally infeasible,
- M - The run was terminated after reaching the maximum number of iterations,
- R - Termination with $\rho < \epsilon_{\text{tol}}$.

By using the failure type notation, Table 4.1 gives a breakdown of the results of our runs.

Failure type	Number of problems		
	SLPSQP	filterSQP	LANCELOT
- E -	0	0	1
- H -	1	1	0
- I -	6	19	13
- M -	0	0	16
- R -	9	0	0
Total number of failures	16	20	30
Total number of successful runs	284	280	270

Table 4.1 Types of failure for SLPSQP, filterSQP and LANCELOT on 300 test problems

From Table 4.1, we can see that the codes SLPSQP and filterSQP which use the filter strategy report fewer failures than LANCELOT. Performance-wise, both SLPSQP and filterSQP have 94.67% and 93.33% success rate respectively as compared with LANCELOT which manages to solve 80% of the 300 test problems. Although more tests are needed to reach a more rigorous conclusion, the preliminary results do show that the filter concept together with the EQP active set strategy can be an attractive choice.

Another encouraging aspect of SLPSQP can also be seen by comparing the number of solved problems for a certain range of number of gradient evaluations (see Figures 4.2 and 4.3).

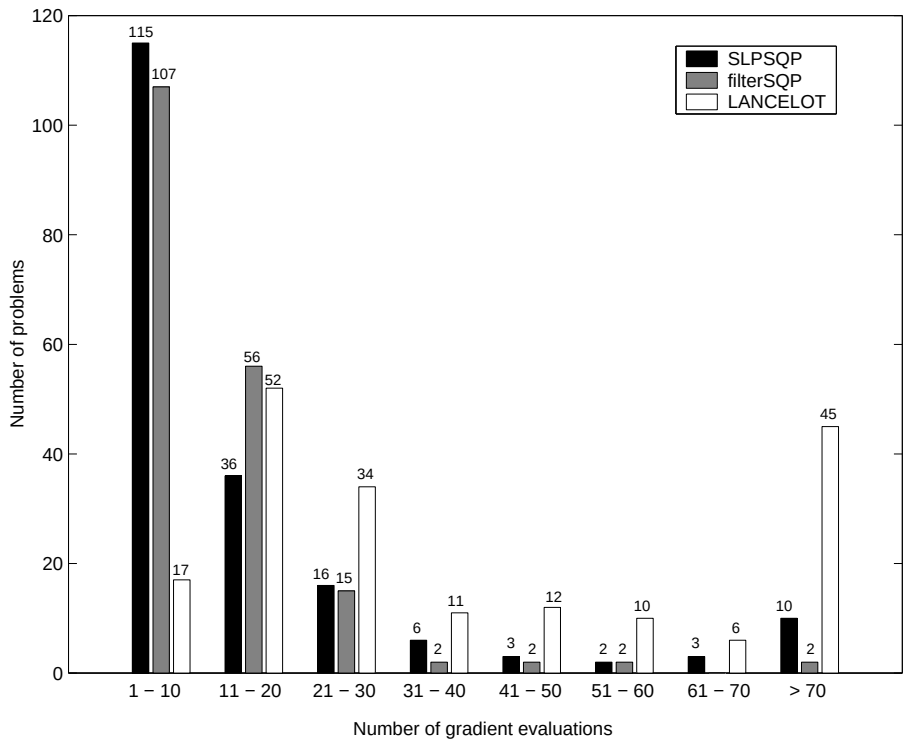

Figure 4.2: Comparing the number of gradient evaluations for small scale problems

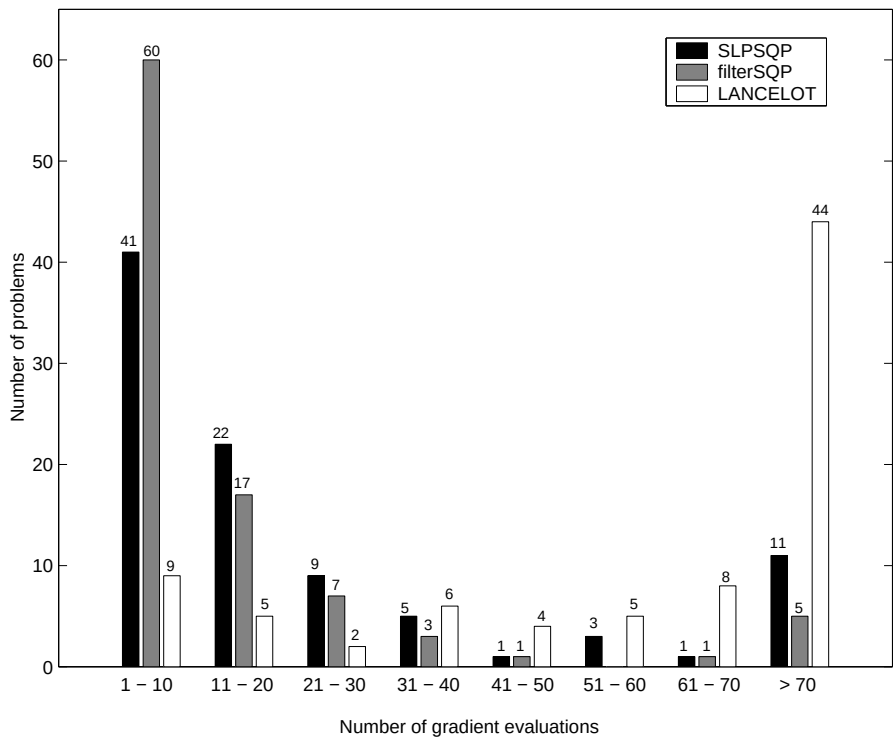

Figure 4.3: Comparing the number of gradient evaluations for medium and large scale problems

From Figures 4.2 and 4.3, we can see a majority of the problems solved by SLPSQP required less than 10 gradient evaluations while only a small proportion of the problems solved by LANCELOT belong to that category. However, filterSQP solves more problems with fewer gradient evaluations than SLPSQP in the case of medium and large scale problems and is by far the most efficient. Overall, the method we propose is comparable to both filterSQP and LANCELOT which indicates that SLPSQP does not lose out much in terms of solving LP rather than QP subproblems.

We now turn our attention to the comparison of CPU times where Figures 4.4 and 4.5 give a description of the number of problems solved for a certain range of CPU times.

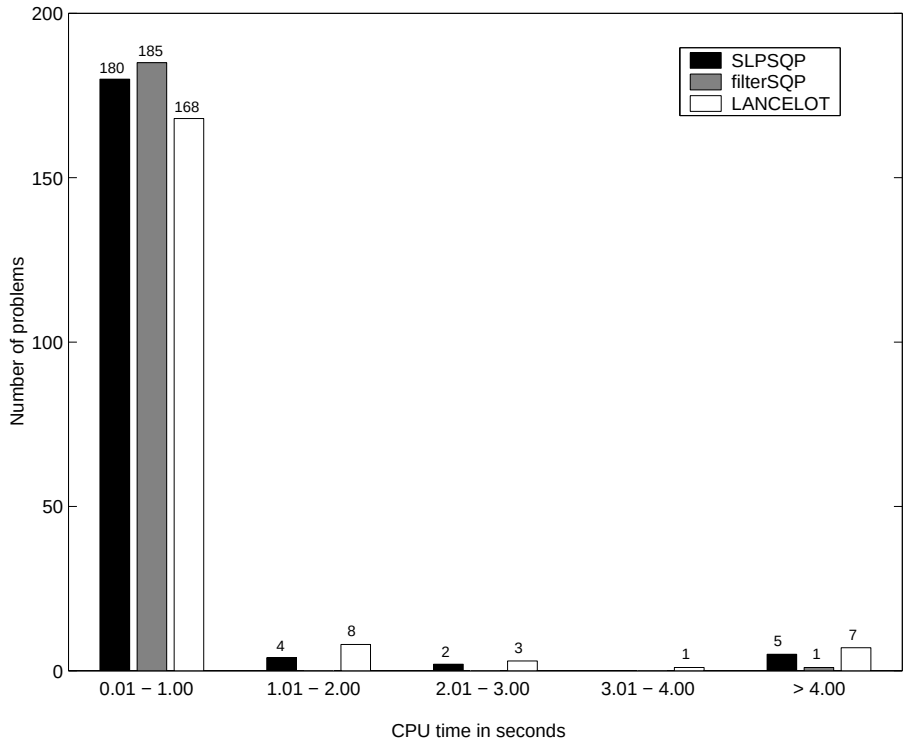

Figure 4.4: Comparing the CPU times for small scale problems

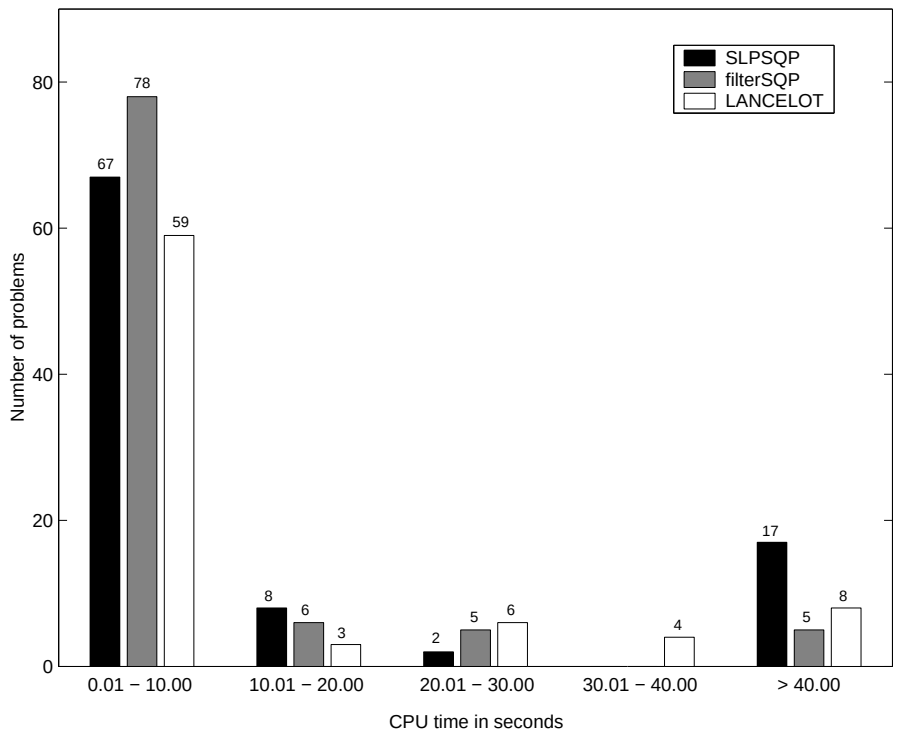

Figure 4.5: Comparing the CPU times for medium and large scale problems

From Figure 4.4, we can see a majority of small scale problems solved by either SLPSQP, filterSQP or LANCELOT require a computation time less than 1 second. The same efficiency is also observed in the performance of SLPSQP for medium and large scale problems where a majority of test problems are solved within 10 seconds of computational time. However, the proportion of medium and large scale test problems solved by SLPSQP which required a CPU time of greater than 40 seconds is much higher as compared with filterSQP and LANCELOT. One possible reason for the relative slowness of SLPSQP is that for some highly nonlinear problems, a poor approximation of the QP working set determined by solving LP subproblems at some stage of the iteration is obtained. In addition, some problems like HANGING, OPTMASS ($n = 610$) and OPTMASS ($n = 1210$) have a large null-space at its solution and this is unfavourable to our QP solver as it needs more computation time to factorize a dense reduced Hessian matrix of large dimension at every inner iteration. Since only a small proportion of test problems solved by SLPSQP have larger CPU times as compared with the total number of problems solved, we feel from an efficiency point of view that SLPSQP is as efficient as either filterSQP or LANCELOT if the evaluations of the problem functions are not expensive or if the size of the null-space is not large.

5 Conclusions

From what we have discussed concerning SLPSQP in relation with filterSQP and LANCELOT, we can finally draw the following conclusions:

- From the numerical tests, the code SLPSQP is relatively suitable for solving nonlinear optimization problems, including many large scale problems. In terms of number of successful run problems, both SLPSQP and filterSQP outperform LANCELOT and we believe this is due to the filter strategy which is more flexible in accepting iterates, and to the used of restoration phase to treat infeasible subproblems. In addition, both filter codes are also variants of SQP methods and this can be partly explained by the fact that both SLPSQP and filterSQP solve almost the same number of test problems.

- Another encouraging aspect is that from the test results, SLPSQP manages to solve a number of test problems previously not solved by the default version of filterSQP (e.g ARGAUSS, DISC2, DRCAVTY3, EIGENB, EIGMINA, EIGMINC, FLETCHER, HADAMARD, HS63, HS90, HS92, HS93, LOOTSMA, PFIT4 and S316-322). On the other hand, filterSQP also manages to solve a few problems where SLPSQP failed (e.g ARTIF ($n = 12$), ARTIF ($n = 1002$), CRESC50, DISCS, HS101R, MANNE ($n = 1095$), MINMAXBD, OPTCDEG3, POLAK3, ROSENMMX and SVANBERG ($n = 500$).

- The storage needed for the filter in SLPSQP is quite low with 276 out of 284 successfully run problems requiring a filter size less than 60.

- From Figures 4.2 and 4.3, we see that a majority of the test problems of either small or large dimension are solved by SLPSQP with fewer than 10 gradient evaluations. Although there are some problems which need more computation time to solve, nevertheless the number of such problems is small compared with the total number

of problems being solved. In terms of efficiency and reliability, the code SLPSQP in our view is suitable for large scale optimization and is well suited to provide the basis of a commercial NLP code.

6 References

[1] Chin, C.M. (2001), *Numerical results of* SLPSQP, filterSQP *and* LANCELOT *on selected CUTE test problems*, Numerical Analysis Report NA/203, University of Dundee, Dundee, Scotland.

[2] Chin, C.M. and Fletcher, R. (2001), *On the global convergence of an SLP-filter algorithm that takes EQP steps*, Numerical Analysis Report NA/199, University of Dundee, Dundee, Scotland.

[3] Conn, A.R., Gould, N.I.M. and Toint, Ph.L. (1992), LANCELOT: a Fortran Package for Large-Scale Nonlinear Optimization (Release A), Volume 17 of Springer Series in Computational Mathematics, Springer-Verlag, Heidelberg, Berlin and New York.

[4] Fletcher, R. and Leyffer, S. (1998), *Nonlinear programming without a penalty function*, Numerical Analysis Report NA/171, University of Dundee, Dundee, Scotland

[5] Fletcher, R. and Leyffer, S. (1999), *User manual for* filterSQP, Numerical Analysis Report NA/181, University of Dundee, Dundee, Scotland.