Feasible Interior Methods Using Slacks for Nonlinear Optimization

Richard H. Byrd* Jorge Nocedal† Richard A. Waltz†

December 30, 2000

Abstract

A slack-based feasible interior point method is described which can be derived as a modification of infeasible methods. The modification is minor for most line search methods, but trust region methods require special attention. It is shown how the Cauchy point, which is often computed in trust region methods, must be modified so that the feasible method is effective for problems containing both equality and inequality constraints. The relationship between slack-based methods and traditional feasible methods is discussed. Numerical results showing the relative performance of feasible versus infeasible interior point methods are presented.

Key words: constrained optimization, interior point method, feasible method, large-scale optimization, nonlinear programming, primal-dual method, sequential quadratic programming, barrier method, trust region method.

*Computer Science Department, University of Colorado, Boulder, CO 80309. This author was supported by Air Force Office of Scientific Research grant F49620-00-1-0162, Army Research Office Grant DAAG55-98-1-0176, and NSF grant INT-9726199.

†Electrical and Computer Engineering Department, Northwestern University, Evanston IL 60208. These authors were supported by National Science Foundation grant CDA-9726385, and by Department of Energy grant DE-FG02-87ER25047-A004.

1 Introduction

In many applications, it is desirable for all of the iterates generated by an optimization algorithm to be feasible with respect to some or all of the inequality constraints. For example, the objective function may be defined only when some of the constraints are satisfied, making this feature absolutely necessary. In other instances one may want to terminate an algorithm before optimality has been reached and be assured that the current approximate solution is feasible.

Various feasible active set methods (see, e.g. [14]) have been developed by including deflections in the search directions which ensure that the total step points towards the interior of the feasible region. They typically require the solution of two or more linear systems of equations per iteration, although some recent approaches [15] aim at decreasing the cost per iteration.

Interior point approaches provide a natural framework for deriving feasible methods for nonlinear programming. The methods proposed in [1, 7, 12, 13] either start with a feasible point or apply a phase-one procedure to compute one, and then generate strictly feasible iterates. Most other implementations of interior methods for nonlinear programming are based, however, on infeasible algorithms [5, 10, 19, 22] which may enter and leave the feasible region during the course of the minimization.

In this paper we describe a framework for transforming slack-based infeasible methods into feasible methods. In this framework, feasible and infeasible interior algorithms can be considered as variants of the same basic method. Feasibility is controlled by whether or not one resets the slack variables after a trial step has been taken, and how these variables are reset. Using this flexibility one can choose to enforce feasibility with respect to some, all, or none of the inequality constraints depending on what is needed or desired. In addition, this flexibility provides a convenient testing environment for analyzing the effects of staying feasible in interior point methods.

The slack reset strategies may experience difficulties on problems with both equality and inequality constraints. The difficulties will not arise in most line search methods, but can occur in trust region methods. We describe a procedure for generating search directions which ensures that the feasible methods proposed here behave efficiently for problems containing both equality and inequality constraints.

The paper is organized as follows. We first outline, in section 2, the general formulation of an infeasible interior point algorithm for nonlinear programming. In section 3 we describe a strategy which transforms infeasible interior methods into methods that satisfy some or all of the inequality constraints by resetting slack variables. The relationship between these slack-based feasible methods and the classical methods of Fiacco and McCormick [11] is discussed. In section 4 we describe potential difficulties with our strategy when equality constraints are present, and provide guidelines as well as a concrete procedure for modifying the step-computation to deal with this complication. We briefly present an alternative slack-based feasible method which does not involve resetting slack variables in section 5. Numerical results comparing feasible and infeasible methods are presented in section 6. We conclude the paper with final remarks in section 7.

2 Infeasible Methods

The problem under consideration will be formulated as

$$\begin{aligned} \min_{x} \quad & f(x) && (2.1a)\\ \text{s.t.} \quad & h(x) = 0 && (2.1b)\\ & g(x) \geq 0, && (2.1c) \end{aligned}$$

where f, h and g are smooth functions of the variable $x \in I\!R^n$. Here f is a scalar-valued function, and h and g are vector-valued functions. By a feasible method for (2.1) we mean one in which the starting point and all subsequent iterates satisfy the inequality constraints $(2.1c)$.

Infeasible interior methods do not enforce satisfaction of the inequality constraints at each iteration. They typically make use of slack variables to transform (2.1) into the equivalent problem

$$\begin{aligned} \min_{x,s} \quad & f(x) && (2.2a)\\ \text{s.t.} \quad & h(x) = 0 && (2.2b)\\ & g(x) - s = 0 && (2.2c)\\ & s \geq 0. && (2.2d) \end{aligned}$$

We will consider interior methods that, at each iteration, apply a form of Newton's method to solve, to some degree of accuracy, the barrier problem

$$\begin{aligned} \min_{x,s} \quad & \psi(x, s; \mu) \equiv f(x) - \mu \sum_{i \in \mathcal{I}} \ln(s_i) && (2.3a)\\ \text{s.t.} \quad & h(x) = 0 && (2.3b)\\ & g(x) - s = 0 && (2.3c)\\ & s > 0, && (2.3d) \end{aligned}$$

where μ is a positive parameter and $\mathcal{I}$ is the set of indices corresponding to the inequality constraints. We will assume that the methods use a merit function of the form

$$\phi(x, s) = f(x) - \mu \sum_{i \in \mathcal{I}} \ln(s_i) + \chi(c(x, s)), \tag{2.4}$$

where χ is some measure of infeasibility, and

$$c(x, s) = \begin{bmatrix} h(x) \\ g(x) - s \end{bmatrix}. \tag{2.5}$$

The function χ can be chosen as a vector norm, or as some other function with the properties that $\chi(0) = 0$ and $\phi(x, s) \to \infty$ as $s \to 0$. We will also define ϕ to have the value ∞ for $s \leq 0$.

Let us consider the following very general type of iterative method for solving the barrier problem (2.3). This method will be applied until (2.3) is solved to some accuracy; then a new barrier parameter is chosen, and the method is applied again. Methods that change the barrier parameter at each iteration would then apply a single iteration of the following method to each barrier problem.

Algorithm 2.1 *Generic Algorithm (Infeasible method for problem* (2.3)*)*

An iterate x (possibly infeasible) and a slack vector $s > 0$ are given.
while *a stopping rule is not satisfied*
    Compute the step $d = (d_x, d_s)$.
    Define the trial point $x_T = x + d_x$; $s_T = s + d_s$.
    while *$\phi(x_T, s_T)$ is not sufficiently smaller than $\phi(x, s)$*
        Compute a shorter step d.
        Set $x_T = x + d_x$; $s_T = s + d_s$.
    end (while)
    Set $x_+ = x_T$; $s_+ = s_T$.
end (while)

In a trust region method, a shorter step would be obtained by decreasing the trust region radius and recomputing a step, whereas in a line search method, a backtracking line search would be employed. We assume that the step-generation procedure and the merit function ϕ are compatible in the sense that if $\|d\|$ is sufficiently small, the merit function will be decreased. No other assumptions will be made on d until we consider, in section 4, its effect when both equality and inequality constraints are present.

3 A Feasible Method with Slack Resetting

We now describe a way of transforming this infeasible Generic Algorithm into a feasible method while retaining the use of slack variables. For simplicity we will assume that the feasible method must satisfy all inequality constraints at every iteration. It is straightforward to extend the algorithm described below to the case when only some of the inequality constraints must be honored.

Assume the current iterate is feasible with respect to the inequality constraints. To ensure that the next iterate is also feasible, we introduce the following simple modification. After computing a step (d_x, d_s) we redefine the slacks as

$$s_{\mathrm{T}} \leftarrow g(x_{\mathrm{T}}), \tag{3.6}$$

and test whether the point $(x_{\mathrm{T}}, s_{\mathrm{T}})$ is acceptable for the merit function (2.4). If it is not, we reject the step and compute a new, shorter, trial step.

If the initial iterate x_0 does not satisfy all inequality constraints, we first apply the infeasible Generic Algorithm until all inequalities are greater than some threshold value.

At that point the algorithm switches to the feasible mode, and stays feasible for the rest of the optimization calculation. This algorithm (Feas-Reset) is summarized below. We let e denote the vector of ones, of appropriate dimension.

Algorithm 3.1 *Algorithm Feas-Reset*

An iterate x (possibly infeasible), a slack vector $s > 0$, and a positive threshold value τ are given.
if $g(x) < \tau e$ **then**
  Run infeasible Generic Algorithm until $g(x) \geq \tau e$.
  Set $s = g(x)$.
end (if)
while *a stopping test is not satisfied*
  Compute the step $d = (d_x, d_s)$ as in the Generic Algorithm.
  Define the trial point $x_{\scriptscriptstyle T} = x + d_x$; $\boxed{s_{\scriptscriptstyle T} = g(x_{\scriptscriptstyle T})}$.
  while *$\phi(x_{\scriptscriptstyle T}, s_{\scriptscriptstyle T})$ is not sufficiently smaller than $\phi(x, s)$*
    Compute a shorter step d.
    Set $x_{\scriptscriptstyle T} = x + d_x$; $\boxed{s_{\scriptscriptstyle T} = g(x_{\scriptscriptstyle T})}$.
  end (while)
  Set $x_+ = x_{\scriptscriptstyle T}$; $s_+ = s_{\scriptscriptstyle T}$.
end (while)

Due to the use of slacks, the feasible and infeasible modes require the same data structures and variables, and only differ in the two instructions enclosed in boxes. The test $g(x) \geq \tau e$, can be replaced by some other condition that does not treat each constraint equally and that takes into account the scale of the constraints.

Making the substitution (3.6) has the effect of replacing $\ln(s_i)$ with $\ln(g_i(x))$ in the merit function, which is the standard form of classical barrier functions [11]. If at a trial point we have that $g_i(x_{\scriptscriptstyle T}) \leq 0$ for some inequality constraint, the value of the merit function is $+\infty$, and we reject the trial point. Note that this approach will also reject steps $x + d_x$ that are too close to the boundary of the feasible region because such steps increase the barrier term $-\mu \sum_{i \in \mathcal{I}} \ln(s_i)$ in the merit function (2.4).

In section 6 we show that a trust region implementation of this feasible method is efficient in practice for problems with inequality constraints only, and also handles problems with equality constraints well provided a modification to the step computation is made.

3.1 Equivalence of Slack-based and Classical Feasible Methods

We now ask if Algorithm Feas-Reset, in its feasible mode, is identical to a classical barrier method without slacks of the type described in [11]. By a classical barrier method we mean

one in which Newton's method is applied to the problem:

$$\min_x \quad f(x) - \mu \sum_{i \in \mathcal{I}} \ln(g_i(x)) \tag{3.7a}$$

$$\text{s.t.} \quad h(x) = 0. \tag{3.7b}$$

At first it may appear that Algorithm Feas-Reset cannot be equivalent to this method because it uses slack variables in the step computation—even in feasible mode. It is easy to see, however, that for a class of interior methods, the reset (3.6) has the effect of eliminating the slacks in the step computation and working directly with problem (3.7).

To show this we first note that the KKT conditions of (3.7) are

$$\begin{aligned} \nabla f(x) - A_h \lambda_h - \mu A_g G(x)^{-1} e &= 0 \\ h(x) &= 0, \end{aligned}$$

where A_h^T and A_g^T denote the Jacobian matrices of h and g, respectively, λ_h is the vector of Lagrange multipliers for the equality constraints (3.7b), and $G(x)$ is a diagonal matrix whose diagonal is given by the components of $g(x)$. These conditions can be reformulated so as to be more benign for Newton's method: introducing the variable

$$\lambda_g = \mu G^{-1}(x),$$

we obtain

$$\begin{aligned} \nabla f(x) - A_h \lambda_h - A_g \lambda_g &= 0 \\ h(x) &= 0 \\ -\mu e + G(x)\lambda_g &= 0. \end{aligned} \tag{3.8}$$

Applying Newton's method (in the variables, x, λ_h, λ_g) to this system gives a primal-dual interior method for (3.7); see e.g. [8].

To study the relationship between this method and Algorithm Feas-Reset, let us consider a slack-based, feasible method for solving (2.3) that computes steps by applying Newton's method in the variables $x, s, \lambda_h, \lambda_g$ to the system

$$\begin{aligned} \nabla f(x) - A_h \lambda_h - A_g \lambda_g &= 0 \\ \lambda_i s_i &= \mu, \quad i \in \mathcal{I} \\ h(x) &= 0 \\ g(x) - s &= 0, \end{aligned}$$

which is equivalent to the KKT conditions for (2.3). This system is the basis for primal-dual infeasible algorithms; see e.g. [5, 19]. Application of Newton's method gives rise to the linear system

$$\begin{pmatrix} \nabla^2_{xx} L & 0 & A_h(x) & A_g(x) \\ 0 & \Lambda & 0 & -S \\ A_h(x)^T & 0 & 0 & 0 \\ A_g(x)^T & -I & 0 & 0 \end{pmatrix} \begin{pmatrix} d_x \\ d_s \\ -d_{\lambda_h} \\ -d_{\lambda_g} \end{pmatrix} = - \begin{pmatrix} \nabla_x L(x, \lambda) \\ -\mu e + S\lambda_g \\ h(x) \\ g(x) - s \end{pmatrix}, \tag{3.9}$$

where S and Λ denote diagonal matrices with s and λ_g on their respective diagonals, and L stands for the Lagrangian of (2.1):

$$L(x, \lambda_h, \lambda_g) = f(x) - \lambda_h^T h(x) - \lambda_g^T g(x). \tag{3.10}$$

Using the fact that $g(x) - s = 0$ due to the reset (3.6), we can eliminate d_s and s to obtain the equivalent linear system

$$\begin{pmatrix} \nabla^2_{xx} L & A_h(x) & A_g(x) \\ A_h(x)^T & 0 & 0 \\ \Lambda A_g(x)^T & 0 & -G(x) \end{pmatrix} \begin{pmatrix} d_x \\ -d_{\lambda_h} \\ -d_{\lambda_g} \end{pmatrix} = - \begin{pmatrix} \nabla_x L(x, \lambda) \\ h(x) \\ -\mu e + G(x)\lambda_g \end{pmatrix}. \tag{3.11}$$

This system is just Newton's method in the variables x, λ_h, λ_g applied to (3.8). Therefore, a primal-dual step for (3.7) is equivalent to a primal-dual step for (2.3), when the reset (3.6) is applied.

This equivalence does not always hold for interior methods [5, 9, 22] in which the step d does not always satisfy (3.9). However, as the iterates of those methods approach a solution, their steps approximate (3.9) with increasing accuracy, and their feasible version resembles a classical barrier method.

4 Effects of Equality Constraints

An iteration of Algorithm Feas-Reset (Algorithm 3.1) is successful if it results in a decrease in the merit function (2.4), which we restate for convenience,

$$\phi(x, s) = f(x) - \mu \sum_{i \in \mathcal{I}} \ln(s_i) + \chi(c(x, s)). \tag{4.12}$$

The computation of the step (d_x, d_s) is designed to cause such a decrease. The slack reset step $s_{\mathrm{T}} \leftarrow g(x_{\mathrm{T}})$ is, however, taken without regard to the merit function ϕ and can cause an increase in it. In particular, if d_x leads toward the boundary of an inequality constraint, the reset $s_{\mathrm{T}} \leftarrow g(x_{\mathrm{T}})$ in Algorithm Feas-Reset can cause the corresponding slack variable to take on a smaller value, increasing the term $-\mu \sum_{i \in \mathcal{I}} \ln(s_i)$ in the merit function. If this results in a total increase in the merit function, then a shorter step is computed. This behavior is not unexpected: it is the mechanism that prevents Algorithm Feas-Reset from generating steps that leave the feasible region or that get too close to its boundary. Indeed, for problems with inequality constraints only, this mechanism steers the iterates away from the boundary and results in an effective method.

It turns out, however, that when both equality and inequality constraints are present, it is harder to keep the iterates away from the boundary of the feasible region, and the attempt to stay feasible by means of slack resetting can actually cause the method to fail. To see what is the source of this problem, let us denote by δ_s the change in the slack due to the reset, i.e.,

$$s + d_s + \delta_s = s_{\mathrm{T}} = g(x + d_x). \tag{4.13}$$

As mentioned above, δ_s can cause an increase in the merit function even when the original step $d = (d_x, d_s)$ would have decreased it, and this is not necessarily undesirable. It is essential, however, that if a sequence of steps is rejected and the steps become increasingly small, any merit function increase due to the slack reset δ_s is eventually offset by the decrease in ϕ provided by the step d. This guarantees that, if the current iterate is not a stationary point, the algorithm will move away from this point. Unfortunately this may not be the case in methods that handle constraints by a trust region method. To illustrate this, we now present an example that occurs with a feasible version of the algorithm implemented in the NITRO package [4, 5].

Example 1. Consider the problem GAUSSELM from the CUTE collection [2]. This problem has 385 nonlinear equality constraints along with 750 linear inequality constraints and some bounds on the variables. We attempted to solve it using Algorithm Feas-Reset with steps d generated by NITRO, a trust region interior method. The merit function is given by (4.12) with $\chi(\cdot) = \|\cdot\|_2$. Some of the first 20 and last 3 iterations of the run are shown in Table 1. All the iterations of the run occur for a fixed barrier parameter value μ.

In Table 1, `Iter` refers to the iteration number, `Step` indicates whether or not the trial point was accepted or rejected, `Barr Obj` is the barrier objective value ψ defined in (2.3a), $\|c(x,s)\|_2 = \|(h(x), g(x) - s)\|_2$, `Delta` is the trust region radius, `Merit Red` is the reduction in the merit function ϕ obtained by the step, `Trial` indicates whether the trial point is feasible or not, and δ_s is the perturbation due to the slack reset (3.6). We note that the slack reset is not performed if the trial point is infeasible.

Even though all iterates are feasible with respect to the inequality constraints, we can see from Table 1 that there exist violated equality constraints since $\|c(x,s)\|$ is nonzero. From iteration 18 on, all the trial steps are feasible, but the merit function increases even as the trust region radius approaches zero. It is easy to see that this increase in the merit function is caused by the slack reset perturbation δ_s which is relatively large starting at iteration 5 and does not decrease quickly enough as the trust region approaches zero. Note that in the later iterations the length of δ_s is comparable to the trust region radius and, as a consequence, to the length of the step d. The algorithm eventually fails because the trust region radius becomes smaller than a preset tolerance.

This example indicates that the step produced by a trust region method, such as the one implemented in NITRO, is not appropriate for a slack reset in the presence of equality and inequality constraints. We will show, however, that by modifying the step computation slack resets can still be effective.

4.1 Step acceptance in Algorithm Feas-Reset

As we will show later on, it is notable that the difficulties in Algorithm Feas-Reset observed in Example 1 do not occur for problems with only inequality constraints. To propose remedies, we need to understand what is special about the handling of equality and inequality constraints in a trust region method.

Iter	Step	Barr Obj	$\|c(x,s)\|_2$	Delta	Merit Red	Trial	$\|\delta_s\|_2$
1	OK	4.51e+01	1.294e-03	1.000e+00	6.98e-01	feas	3.486e-15
2	OK	4.14e+01	3.014e-03	7.000e+00	3.69e+00	feas	4.253e-15
3	OK	3.28e+01	7.173e-02	4.900e+01	8.50e+00	feas	5.989e-15
4	OK	3.07e+01	9.705e-01	9.800e+01	1.28e+00	feas	5.224e-15
5	OK	2.82e+01	8.083e-01	9.800e+01	9.89e-01	feas	2.345e-01
6	rej	2.66e+01	3.285e-01	9.800e+01	9.89e-01	inf	---------
7	rej	2.49e+01	4.642e-01	3.434e+00	9.89e-01	inf	---------
...							
14	OK	2.82e+01	7.980e-01	2.683e-02	8.09e-02	feas	4.347e-02
15	rej	2.81e+01	7.769e-01	5.366e-02	8.09e-02	inf	---------
16	rej	2.82e+01	7.874e-01	2.683e-02	8.09e-02	inf	---------
17	rej	2.82e+01	7.927e-01	1.341e-02	8.09e-02	inf	---------
18	rej	2.84e+01	7.954e-01	6.707e-03	-1.34e-01	feas	1.115e-02
19	rej	2.83e+01	7.967e-01	3.354e-03	-3.82e-02	feas	5.579e-03
20	rej	2.82e+01	7.974e-01	1.677e-03	-1.54e-02	feas	2.789e-03
...							
55	rej	2.82e+01	7.980e-01	4.880e-14	-2.86e-13	feas	8.136e-14
56	rej	2.82e+01	7.980e-01	2.440e-14	-5.42e-14	feas	4.088e-14
57	rej	2.82e+01	7.980e-01	1.220e-14	-1.55e-13	feas	2.007e-14

Table 1: Algorithm Feas-Reset on problem GAUSSELM

Let us begin by considering, by way of contrast, line search interior methods. They typically demand [1, 7, 10, 13, 19] that the step satisfy the linearized constraints

$$c(x,s) + A(x)^T d = 0, \tag{4.14}$$

where $c(x,s)$ is defined by (2.5) and $A(x)^T$ denotes its Jacobian matrix, i.e.,

$$A(x) = \begin{bmatrix} A_h(x) & A_g(x) \\ 0 & -I \end{bmatrix}. \tag{4.15}$$

If x is feasible such that $g(x) - s = 0$, the second block of equations in (4.14) reads

$$A_g(x)^T d_x - d_s = 0. \tag{4.16}$$

We now show that, when all steps satisfy (4.16), the slack reset in Algorithm Feas-Reset will cause at most a small increase in the merit function ϕ which is offset by the decrease due to d—provided this step is of appropriate length. As a consequence, when (4.16) holds, the type of failure exhibited in Example 1 will not occur.

We first note that most constrained optimization methods generate a step d that is a direction of first order decrease of the merit function, or at the very least, is arbitrarily close to such a direction when $\|d\|$ is sufficiently small [8]. (This property holds at any iterate

that is not a stationary point of the problem, and where regularity holds.) This implies that at such a point there is a positive constant γ such that when d is sufficiently small, the decrease in the merit function satisfies

$$\phi(x,s) - \phi(x+d_x, s+d_s) \geq \gamma \|d\|. \tag{4.17}$$

The following result gives conditions under which the perturbation δ_s defined in (4.13) will be much smaller than d, ensuring that the decrease in the merit function due to the step d will be maintained.

Theorem 4.1 *Suppose Algorithm Feas-Reset is applied to the barrier problem* (2.3)*, and that the function* χ *in* (4.12) *is Lipschitz continuous. Let* (x,s) *be a feasible iterate with* $s > 0$*, and suppose that the step* $d = (d_x, d_s)$ *satisfies* (4.16)*. Then the perturbation in the slacks due to the reset* (3.6) *satisfies*

$$\|\delta_s\| = O(\|d_x\|)^2.$$

As a consequence, if (4.17) *holds, then for* d *sufficiently small the total step* $(d_x, d_s + \delta_s)$ *will reduce the merit function and the step will be accepted.*

Proof. The equality $g(x) - s = 0$, and (4.16) imply that

$$\|g(x+d_x) - (s+d_s)\| = O(\|d_x\|^2). \tag{4.18}$$

Combining this with (4.13) we obtain

$$\|\delta_s\| = O(\|d_x\|^2). \tag{4.19}$$

The effect of this reset on the merit function ϕ is also $O(\|\delta_s\|)$ since ϕ is Lipschitz continuous so that

$$\phi(x+d_x, s+d_s) - \phi(x+d_x, s+d_s+\delta_s) = O(\|\delta_s\|) = O(\|d_x\|^2).$$

Together with (4.17) this implies that for d sufficiently small, $\phi(x+d_x, s+d_s+\delta_s) < \phi(x,s)$.

□

Theorem 4.1 guarantees that the failure observed in Example 1 cannot occur with line search methods since their search directions typically satisfy (4.16) and (4.17). On the other hand, many trust region methods, including the algorithm in NITRO, do not always impose (4.16) and the guarantee of Theorem 4.1 is not available. We now discuss why (4.16) may not be satisfied by these methods, and how the difficulties illustrated in Example 1 can be remedied.

4.2 Behavior of feasible trust region methods

In a trust region method for solving the barrier problem (2.3), the step $d = (d_x, d_s)$ is not computed by solving the Newton system (3.9). Instead it is defined as the (approximate or exact) solution of a subproblem of the form

$$\min_d \quad \tfrac{1}{2} d^T W d + \nabla \psi(x, s; \mu)^T d \tag{4.20a}$$

$$\text{s.t.} \quad h(x) + A_h^T d = r_h \tag{4.20b}$$

$$g(x) - s + A_g^T d_x - d_s = r_g \tag{4.20c}$$

$$\|d\| \leq \Delta. \tag{4.20d}$$

Here W is the Hessian of the Lagrangian of the barrier problem (2.3), or a related matrix, and ψ is the barrier function. The vectors r_h and r_g are chosen with the dual objective of ensuring that the constraints (4.20b)-(4.20d) are compatible, and that the step makes sufficient progress towards achieving feasibility. The methods of Vardi [20], Celis, Dennis and Tapia [6], Powell and Yuan [18], Byrd and Omojokun [3, 16], and Yamashita and Yabe [21] can be described in this framework.

If there is a step d satisfying (4.20b)-(4.20d) for $r_h = r_g = 0$, then these trust region methods will usually define r_h and r_g to be zero. A situation where this always occurs is in a problem that contains *only inequality* constraints, and the current iterate is feasible. In this case $g(x) = s$, and thus (4.20c) can be satisfied with $r_g = 0$ by arbitrarily small steps d that lie inside the trust region. Thus if Algorithm Feas-Reset is used with such a method (4.16) will be satisfied. Therefore by Theorem 4.1 a failure like Example 1 will not occur for problems containing only inequality constraints.

It is often the case, however, that for general problems the constraints (4.20b)-(4.20d) are inconsistent and nonzero values of r_h and r_g must be chosen. Moreover, by the way these vectors are chosen in many standard trust region methods [3, 6, 16, 18] it is almost certain that if one of the vectors r_g, r_h is nonzero then *both* r_g and r_h will be nonzero. As a result the linear equation (4.16), which we have seen guarantees that the slack reset is not harmful, will not be satisfied. This is undesirable in the context of Algorithm Feas-Reset because the reset perturbation δ_s can be as large as the step d and cause a net increase in the merit function even as $\|d_x\| \to 0$. This can even occur when all constraints are linear.

The difficulty can be resolved by demanding that, whenever $g(x) - s = 0$, the vector r_g be chosen to be zero in (4.20c). The vector r_h should still ensure that (4.20b) and (4.20d) are compatible and that the step makes sufficient progress toward satisfying the equality constraints $h(x)$. Below we discuss how to satisfy these requirements using the algorithm in NITRO.

4.3 An Improved Normal Step for NITRO

In the algorithm implemented in NITRO, the vectors r_h and r_g in (4.20) are obtained by solving the auxiliary problem[1]

$$\min_{\eta} \quad \|A(x)^T\eta + c(x,s)\|_2^2 \tag{4.21a}$$

$$\text{s.t.} \quad \|\eta\|_{\text{TR}} \leq \Delta', \tag{4.21b}$$

where $\|\cdot\|_{\text{TR}}$ denotes a scaled trust region, and $\Delta' < \Delta$. Once the solution η is computed, we define

$$r_h = h(x) + A_h^T\eta_x, \qquad r_g = g(x) - s + A_g^T\eta_x - \eta_s.$$

The full step of the algorithm is given by

$$d = \eta + t, \tag{4.22}$$

where the "tangential component" t lies in the null space of $A(x)^T$ and attempts to move towards optimality, while the "normal component" η attempts to improve feasibility [5].

In the context of Algorithm Feas-Reset, the drawback of improving feasibility by solving (4.21) is that all the constraints are treated equally, so that if $g(x) - s = 0$ is satisfied at the current iterate x and the trust region is active, the effort to improve all the constraints is almost certain to result in a nonzero value of the linearization (4.16), and hence in a nonzero value r_g. To resolve this, instead of minimizing (4.21a) we follow the suggestion given above and impose the condition $r_g = 0$ when the iterate is feasible.

Thus we wish to compute a step that significantly reduces $\|A_h(x)^T d + h(x)\|$ while lying inside the trust region and satisfying the linearization (4.16) of the inequality constraints. To describe a practical way to do this in the context of NITRO, we must consider in detail two aspects of the normal step computation. The first observation is that the norm in (4.21b) is defined as

$$\|\eta\|_{\text{TR}} = \|(\eta_x, \eta_s)\|_{\text{TR}} \equiv \|(\eta_x, S^{-1}\eta_s)\|_2,$$

where, as before, S is a diagonal matrix with the current values of the slack variables on the diagonal. This elliptical trust region is transformed into a spherical trust region by introducing the variable

$$\bar{\eta} = (\eta_x, \bar{\eta}_s) \equiv (\eta_x, S^{-1}\eta_s).$$

The linear term in (4.21a) then becomes

$$A(x)^T\eta = \bar{A}(x)^T\bar{\eta} \equiv \begin{bmatrix} A_h(x)^T & 0 \\ A_g(x)^T & -S \end{bmatrix} \begin{bmatrix} \eta_x \\ \bar{\eta}_s \end{bmatrix}.$$

The second observation is that the approximate solution of (4.21a) is computed in NITRO by means of the dogleg method [17]: it is a linear combination of a Newton step $\bar{\eta}^N$ and a Cauchy step $\bar{\eta}^C$. The Newton step is the solution of $\bar{A}(x)^T\bar{\eta}^N = -c(x,s)$ in the

[1] In addition to (4.21b) a bound on slack steps must be imposed, but this is done after the solution of (4.21).

range of $\bar{A}(x)$, and the Cauchy step $\bar{\eta}^C$ is a multiple of the direction of steepest descent on $\|\bar{A}(x)^T\bar{\eta} + c(x,s)\|_2^2$ in the scaled variables. When $g(x) - s = 0$, the x component of the Cauchy step is given by

$$\bar{\eta}_x^C = -\alpha A_h(x)h(x), \tag{4.23}$$

where α is chosen to minimize $\|A(x)^T\bar{\eta}^C + c(x,s)\|_2^2$.

By construction, the Newton step $\bar{\eta}^N$ satisfies (4.16), but the Cauchy step $\bar{\eta}^C$ does not. Thus when $\bar{\eta} = \bar{\eta}^N$, the normal step has the desired properties; otherwise we must develop a modified Cauchy step with the following three properties: (i) it should satisfy (4.16),

$$A_g(x)^T\eta_x^C - \eta_s^C = A_g(x)^T\eta_x^C - S\bar{\eta}_s^C = 0; \tag{4.24}$$

(ii) like the standard Cauchy step, the modified step should lie in the range of $\bar{A}(x)$; (iii) since it must make progress on the linearized equality constraints we require that the modified step act on the linearized equality constraints like the standard Cauchy step (4.23), so that

$$A_h(x)^T\eta_x^C = -A_h(x)^TA_h(x)h(x). \tag{4.25}$$

Together these three desired properties of the modified Cauchy step amount to requiring that $\bar{\eta}^C = (\eta_x^C, \bar{\eta}_s^C)$ solve the equations

$$\begin{aligned}
\eta_x^C + A_h(x)p_1 + A_g(x)p_2 &= 0 && \text{(4.26a)}\\
\bar{\eta}_s^C - Sp_2 &= 0 && \text{(4.26b)}\\
A_h(x)^T\eta_x^C &= -A_h(x)^TA_h(x)h(x) && \text{(4.26c)}\\
A_g(x)^T\eta_x^C - S\bar{\eta}_s^C &= 0, && \text{(4.26d)}
\end{aligned}$$

where p_1 and p_2 are auxiliary vectors. This can be written in matrix form as

$$\begin{bmatrix} I & 0 & A_h(x) & A_g(x) \\ 0 & I & 0 & -S \\ A_h(x)^T & 0 & 0 & 0 \\ A_g(x)^T & -S & 0 & 0 \end{bmatrix} \begin{bmatrix} \eta_x^C \\ \bar{\eta}_s^C \\ p_1 \\ p_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -A_h(x)^TA_h(x)h(x) \\ 0 \end{bmatrix}. \tag{4.27}$$

In summary, the change needed in the step computation of NITRO occurs only in the normal component η of the step (4.22). This normal component is formed by a Newton step $\bar{\eta}^N$, which needs no modification, and a Cauchy step $\bar{\eta}^C$, which is now computed via (4.27).

The step defined by (4.27) is not the only modification that is effective in the context of Algorithm Feas-Reset. For example, computing the steepest descent direction of $\|A_h(x)^T\eta_x + h(x)\|_2^2$ in the subspace of vectors satisfying (4.24) would yield an effective step, although it might be computationally expensive. A major advantage of using the solution to (4.27) is that the matrix on the left hand side of (4.27) is already factored in NITRO to compute the Newton direction $\bar{\eta}^N$, Lagrange multiplier estimates, and other quantities. Thus the extra cost of computing the proposed Cauchy step is quite moderate, amounting to the cost of one backsolve using the factors of the coefficient matrix in (4.27).

Example 2. We consider again the behavior of Algorithm Feas-Reset on problem GAUSSELM, this time using the modified normal step for NITRO described above. The optimal solution is reached in 102 iterations without any difficulties. For the sake of space only the first 15 iterations (all using the same barrier parameter value) are shown in Table 2. We note that the perturbation caused by the slack reset, δ_s, is always of the order of unit roundoff, and therefore does not prevent a decrease in the merit function.

We also tested the modified normal step by comparing the performance of Algorithm Feas-Reset implemented in the NITRO package with and without the modified step. Our test set had 153 problems, all containing equality constraints, and of those the two versions of NITRO performed differently on 45 problems. Of these 45 problems the new step resulted in failure on 3 problems, as compared to 17 failures for the unmodified step. On the 25 problems solved by both, NITRO with the new Cauchy step required significantly fewer iterations.

Iter	Step	Barr Obj	$\|c(x,s)\|_2$	Delta	Merit Red	Trial	$\|\delta_s\|_2$
1	OK	4.51e+01	1.294e-03	1.000e+00	6.98e-01	feas	3.486e-15
2	OK	4.14e+01	3.014e-03	7.000e+00	3.69e+00	feas	4.253e-15
3	OK	3.28e+01	7.173e-02	4.900e+01	8.50e+00	feas	5.989e-15
4	OK	3.04e+01	7.269e-01	9.800e+01	1.31e+00	feas	1.164e-14
5	OK	2.58e+01	5.199e-01	9.800e+01	4.86e+00	feas	4.063e-15
6	OK	1.20e+01	4.832e-01	9.800e+01	1.34e+01	feas	4.783e-15
7	OK	-2.77e-01	1.216e-01	9.800e+01	1.33e+01	feas	5.038e-15
8	OK	-1.03e+01	3.787e-01	9.800e+01	9.65e+00	feas	5.968e-15
9	OK	-1.62e+01	1.416e-01	9.800e+01	6.17e+00	feas	5.933e-15
10	OK	-1.82e+01	3.236e-02	9.800e+01	2.37e+00	feas	6.203e-15
11	OK	-1.98e+01	2.976e-01	9.800e+01	9.05e-01	feas	6.670e-15
12	OK	-2.52e+01	4.853e-02	9.800e+01	6.09e+00	feas	7.339e-15
13	OK	-2.98e+01	3.315e-02	9.800e+01	4.54e+00	feas	7.767e-15
14	OK	-3.44e+01	4.580e-02	9.800e+01	4.68e+00	feas	7.536e-15
15	OK	-3.82e+01	5.248e-02	9.800e+01	3.82e+00	feas	7.449e-15

Table 2: Algorithm Feas-Reset using the modified normal step on problem GAUSSELM

5 A Feasible Method without Slack Resetting

Although it was fairly simple to derive Algorithm Feas-Reset from a generic algorithm, an even more straightforward modification is to not reset slack variables, but to simply reject any trial iterate x_{T} that is infeasible. In a trust region method we would reduce the trust region and compute a new step; in a line search method we would simply backtrack. A step of this algorithm (Feas-NoReset) may be specified as follows.

Algorithm 5.1 *Algorithm Feas-NoReset*

An iterate x (possibly infeasible), a slack vector $s > 0$, and a positive threshold value τ are given.
if $g(x) < \tau e$ **then**
  Run infeasible Generic Algorithm until $g(x) \geq \tau e$.
end (if)
while *a stopping test is not satisfied*
  Compute the step (d_x, d_s) as in the Generic Algorithm.
  Define the trial point $x_T = x + d_x$; $s_T = s + d_s$.
  while $g(x_T) \leq 0$ *or* $\phi(x_T, s_T)$ *is not sufficiently smaller than* $\phi(x, s)$
    Compute a shorter step d.
    Set $x_T = x + d_x$; $s_T = s + d_s$.
  end (while)
  Set $x_+ = x_T$; $s_+ = s_T$.
end (while)

This algorithm is therefore identical to Feas-Reset method except for the fact that the slack variables are never redefined.

For different reasons, the presence of equality constraints can also cause Algorithm Feas-NoReset to be very inefficient or even fail in some cases. Suppose for example that the current iterate lies at the boundary of the feasible region defined by the inequality constraints and that the problem contains also equality constraints. Then it is possible for the direction generated by an interior point method to point towards the outside of the feasible region. In this case a line search method would fail when the trial steplength approached zero. If the current iterate lies near the boundary of the feasible region, the algorithm would generate very small steps.

A trust region approach can also fail. We illustrate this by means of the problem HS71 which has 4 variables (similar difficulties were observed in several larger problems from the CUTE collection.) The problem is,

$$\begin{aligned} \min_x \quad & x_1 x_4 (x_1 + x_2 + x_3) + x_3 \\ & x_1^2 + x_2^2 + x_3^2 + x_4^2 - 40 = 0 \\ & x_1 x_2 x_3 x_4 - 25 \geq 0 \\ & 1 \leq x_i \leq 5, \quad i = 1, \cdots, 4. \end{aligned}$$

The starting point is $x_0 = (1, 5, 5, 1)$, which is at the boundary of all five of the inequalities. The solution is, to 4 digits, $(1.000, 4.743, 3.821, 1.379)$.

We attempted to solve the problem using Algorithm Feas-NoReset (Algorithm 5.1), where the step d was generated with the NITRO package. From this starting point, the algorithm was unable to generate an acceptable iterate; all the trial iterates violated the equality constraint. As a result the trust region was reduced to a very small value and the

algorithm terminated. These difficulties can be attributed to the presence of the equality constraint: there is nothing to tell the algorithm that by trying to satisfy this constraint it will immediately leave the feasible region defined by the inequalities.

One might think that this problem at the boundary could be avoided if we continue using the infeasible method until the iteration is well inside the interior of the feasible region. We find that strategy to be unsatisfactory, however, because if the algorithm fails at the boundary, it is likely to perform poorly near the boundary, and unlike Algorithm Feas-Reset, the barrier term in this method does not directly force the x values away from the boundary.

We tested this conjecture by slowly moving the starting point for problem HS71 towards the interior of the feasible region. We found that the method failed in a similar fashion with the starting point $x_0 = (1.01, 4.99, 4.99, 1.01)$ which is close to the boundary but significantly feasible such that $g(x_0) \geq 0.01$ is satisfied. The algorithm took a few steps towards the boundary and then was unable to generate an acceptable step. If the initial point was chosen further inside the feasible region the iteration was no longer trapped by the boundary and the problem was solved easily. However, these tests confirmed that given a point reasonably near the boundary, the algorithm may in fact converge to a non-optimal point on the boundary of the feasible region defined by the inequality constraints.

In analogy with Algorithm Feas-Reset, one could also try to improve upon this method by modifying the step computation. However, since this method is inherently different, a new type of step modification is in order. We have not explored this because Algorithm Feas-Reset, with its step modification, has proven to be a robust and efficient feasible method, but it is plausible that a careful modification of Algorithm Feas-NoReset could be just as effective.

6 Numerical Tests

To our knowledge there have been no studies comparing the performance of feasible versus infeasible interior point methods for nonlinear optimization. The algorithmic framework described in this paper is convenient for doing such tests as it allows us to easily switch between a feasible method and an infeasible method while keeping all other algorithmic features the same. One might expect that the flexibility provided by an infeasible method would be advantageous in certain circumstances allowing the iterates to take a more direct path to the solution through infeasible intermediate steps. However, by the same token, it is also conceivable that this additional freedom may allow for poor steps not permitted by feasible methods; this may make an infeasible code less efficient at times. To our surprise, the tests we report below do not indicate a clear superiority of either method and the potential advantages of each approach do not appear to be a major factor.

To compare the relative merits of these approaches we tested both feasible and infeasible versions of NITRO on a set of 216 test problems from the CUTE collection. The problems, and some of their properties, are listed in the Appendix. The infeasible version follows the Generic Algorithm (Algorithm 2.1) with steps generated by NITRO. The feasible version is based on Algorithm Feas-Reset (Algorithm 3.1) with steps determined by NITRO modified

to incorporate the new Cauchy step given by (4.27); the threshold value τ is chosen as $\tau = 10^{-4}$.

All the results were performed on a Sun Ultra 5/10 workstation with 384 MB of memory, in double precision Fortran. The termination test for the runs was the default NITRO stopping test with a tolerance of 10^{-6}; this test is based on a normalized KKT condition.

The results are summarized in Figure 1, and are reported in detail in the Appendix. All the figures given in this section plot the ratio

$$\log_2 \frac{(\text{fevals Alg 1})}{(\text{fevals Alg 2})}, \tag{6.28}$$

where fevals denotes the number of function evaluations required to meet the stopping test, and Alg 1 and Alg 2 denote the two algorithms tested. In the case of Figure 1, Alg 1 stands for the infeasible algorithm, and Alg 2 for the feasible algorithm. In all the figures, the problems are arranged along the horizontal axis in decreasing order of the absolute value of (6.28). We report results only for those problems for which both methods converged to the same solution.

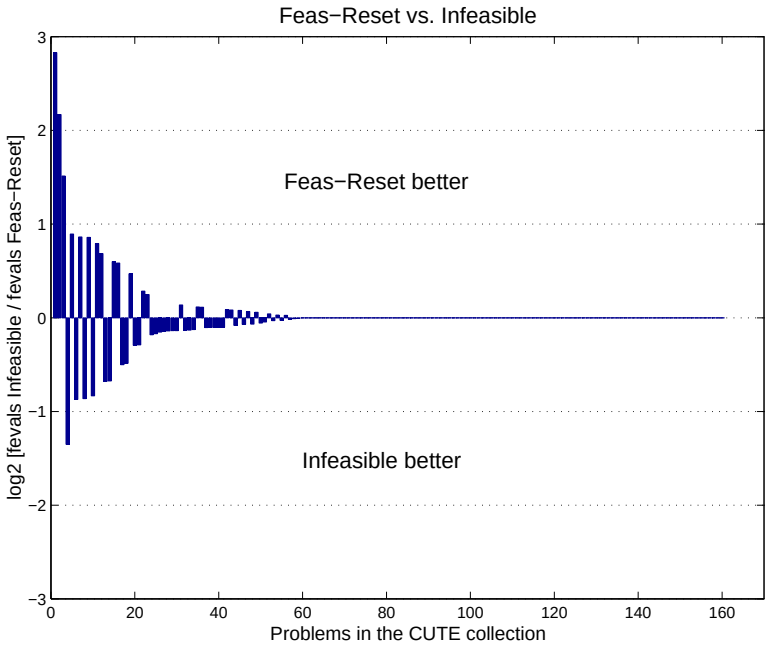

Figure 1: Comparison of infeasible and feasible versions of NITRO on 160 problems in which both algorithms found the same solution. A bar on the upper half means that the feasible algorithm required fewer function evaluations. The logarithmic scale (6.28) was used.

The feasible algorithm solved 181 problems and the infeasible algorithm 179; they mainly failed on the same problems. From this fact and Figure 1 we remark that for most of these problems, there is little difference between the two approaches. We observed that on 57

of the 216 problems (roughly 25%) Algorithm Feas-Reset never entered feasible mode. On these problems, therefore, the feasible and infeasible methods are identical. Moreover on a number of other problems there is little or no difference either because Algorithm Feas-Reset only entered feasible mode near the end of the run or the problems were solved too quickly or easily for there to be much difference. On three problems, the feasible algorithm was dramatically better than the infeasible method, but an examination of the runs does not suggest that this is due to the properties of the feasible iteration.

We also compared the two feasible methods discussed in this paper. In Figure 2 we summarize the performance of Algorithm Feas-Reset (Algorithm 3.1) and Algorithm Feas-NoReset (Algorithm 5.1) on the same set of CUTE problems as in the previous experiment; detailed results are given in the Appendix. The step d was computed in both cases using the NITRO package. Even though for many problems the two methods performed similarly, there is a significant number of problems in which Algorithm Feas-Reset was more efficient. This is not surprising since, in our view, the mechanism of keeping iterates away from the boundary is more effective in Algorithm Feas-Reset.

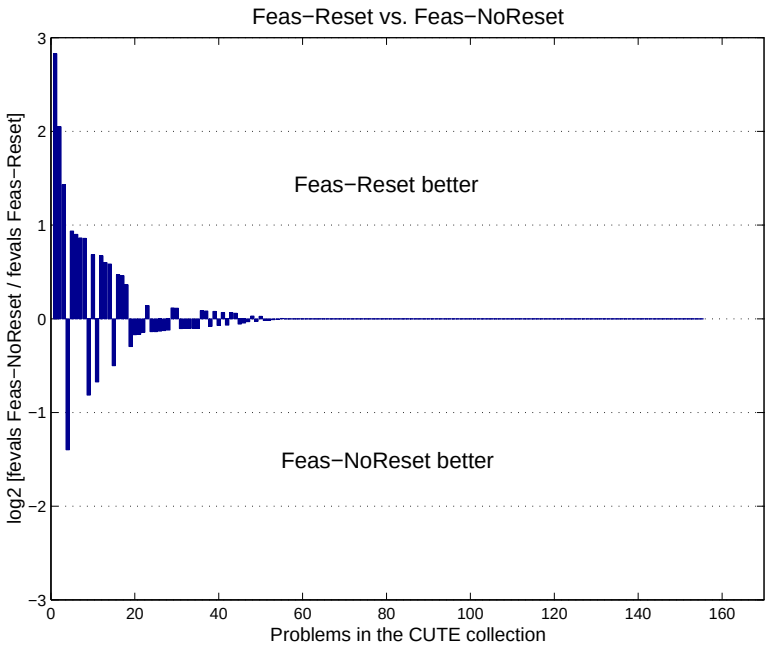

Figure 2: Comparison of two feasible methods, Feas-Reset and Feas-NoReset, on a set of problems from the CUTE collection.

We should point out that there is a feature of the NITRO step computation that contributes to the lack of disparity between the feasible and infeasible methods compared in Figure 1. Although the standard version of NITRO [5] is an infeasible algorithm, it makes use of a slack adjustment to accelerate convergence. NITRO adjusts the slack variables at the end of each iteration (regardless of whether the iterate is feasible or not) according to

the rule

$$s_{\mathrm{T}} \leftarrow \max(g(x_{\mathrm{T}}), s + d_s), \tag{6.29}$$

where the max function is applied component-wise; see step 5 of Algorithm I in [4]).

If one of the components of g is non-positive, this rule will not modify its associated slack variable. But suppose that one of the components of g, say g_j, satisfies $g_j(x_{\mathrm{T}}) > 0$ and that $(s + d_s)_j < g_j(x_{\mathrm{T}})$. Then the rule (6.29) increases the corresponding slack so that $(g(x_{\mathrm{T}}) - s_{\mathrm{T}})_j = 0$, while at the same time achieving a smaller barrier term value $-\mu \sum_{i \in \mathcal{I}} \ln(s_i)$ in the merit function. Note also that if one of the inequality constraints satisfies $g_j(x_{\mathrm{T}}) > 0$ and $(s + d_s)_j > g_j(x_{\mathrm{T}})$, then the value of the slack will not be changed by (6.29), even though by decreasing it we would obtain $(g(x_{\mathrm{T}}) - s_{\mathrm{T}})_j = 0$. We choose not to adjust the slacks in this case because this could cause an increase in the merit function.

This slack adjustment cannot cause the difficulties discussed in section 4.1 because it never gives rise to an increase in the merit function, and there is no need for modifying the step when equalities are present. The rule (6.29) has been analyzed in the context of the NITRO algorithm in [4], and in practice has proven to be superior to using no slack adjustment scheme. In the numerical tests described above, the slack adjustment rule (6.29) is applied at every iteration of the infeasible method and of Algorithm Feas-NoReset. In Algorithm Feas-Reset the slack adjustment is applied only before this method enters feasible mode (at which point it switches to the feasible slack reset scheme). Hence, all methods benefit from the slack adjustment (6.29) where possible. We should stress that using this adjustment dampens the disparity between the three methods tested, since the slack adjustment is identical to the feasible reset when $g(x_{\mathrm{T}}) > s + d_s$. However, we use it in our comparison taking the view that all methods should be tested using their best implementation within the NITRO framework.

7 Final Remarks

We have shown how infeasible slack-based interior methods can be transformed into feasible methods by using slack resets. Even though the modification is straightforward in most line search methods, care must be taken in trust region methods because, for problems containing both equality and inequality constraints, the slack reset can be harmful. We have given guidelines for the step computation so as to maintain feasibility with respect to inequality constraints, while improving the equality constraints. A specific normal step computation for the algorithm in NITRO was developed and shown to be effective in practice.

The formulation (2.2) of a nonlinear program made it simple to transform infeasible interior methods into feasible methods. We note, however, that other formulations do not allow one to apply the procedure used to generate Algorithm Feas-Reset. If the nonlinear program is given in the form

$$\min_x f(x) \quad \text{s.t.} \quad c(x) = 0, \quad x \geq 0,$$

there is no distinction between the components of x that are slacks and those that are variables in the original problem.

We should also point out that in all cases of slack resetting, the reset (3.6) should take place before the evaluation of the merit function. If it is done after evaluating the merit function performance can be significantly degraded since the slack reset effectively changes the iterate, but the merit function does not have the opportunity to measure its quality.

The ideas presented in this paper are applicable to the case when only some of the inequality constraints must be honored. The slack reset in Algorithm Feas-Rest would only be applied to those inequality constraints that must be satisfied. Similarly, to obtain the improved normal component discussed in section 4, we must maintain linear feasibility for the inequality constraints that must be honored.

We conclude by pointing out that the merit function considered in this paper has the form (2.4), but this choice is not of particular importance. Many other choices of merit functions would be allowed in Algorithm Feas-Reset.

8 Acknowledgments

We would like to thank Guanghui Liu who performed some initial experiments with the feasible methods described in this paper. We also thank Marcelo Marazzi for valuable suggestions during this research, and Jose Luis Morales for suggesting the method for displaying the results in the figures.

9 Appendix

Problem	n	m	# of function evaluations			final function value		
			Infeasible	Feasible		Infeasible	Feasible	
				Reset	NoReset		Reset	NoReset
AGG	163	488	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
AIRPORT	84	42	12	12	12	4.80e+04	4.80e+04	4.80e+04
AUG2DCQP	850	400	27	27	27	3.24e+03	3.24e+03	3.24e+03
AUG2DQP	850	400	25	25	25	2.20e+03	2.20e+03	2.20e+03
AUG3DCQP	156	27	20	20	20	3.93e+01	3.93e+01	3.93e+01
AUG3DQP	156	27	20	20	20	4.18e+00	4.18e+00	4.18e+00
BATCH	48	73	385	385	385	2.61e+05	2.61e+05	2.61e+05
BLOCKQP1	205	101	14	15	14	2.49e+00	2.49e+00	2.49e+00
BLOCKQP2	205	101	12	12	12	-9.61e+01	-9.61e+01	-9.61e+01
BLOCKQP3	205	101	17	17	17	2.48e+00	2.48e+00	2.48e+00
BLOCKQP4	205	101	22	22	22	-4.81e+01	-4.81e+01	-4.81e+01
BLOCKQP5	205	101	15	15	15	2.48e+00	2.48e+00	2.48e+00
BLOWEYA	202	102	11	11	11	-4.56e-01	-4.56e-01	-4.56e-01
BLOWEYB	202	102	11	11	11	-3.05e-01	-3.05e-01	-3.05e-01
BLOWEYC	202	102	11	12	11	-3.05e-01	-3.05e-01	-3.05e-01
BRAINPC0	6907	6900	**(2)**	**(2)**	**(3)**	**(2)**	**(2)**	**(3)**
BRAINPC1	6907	6900	97	34	**(3)**	4.00e-04	4.27e-04	**(3)**
BRAINPC2	13807	13800	**(2)**	**(2)**	**(3)**	**(2)**	**(2)**	**(3)**
BRAINPC3	6907	6900	**(2)**	116	**(3)**	**(2)**	3.65e-01	**(3)**
BRAINPC4	6907	6900	**(2)**	**(2)**	**(3)**	**(2)**	**(2)**	**(3)**
BRAINPC5	6907	6900	**(2)**	**(2)**	**(3)**	**(2)**	**(2)**	**(3)**

continued on next page

continued from previous page

Problem	n	m	# of function evaluations			final function value		
			Infeasible	Feasible		Infeasible	Feasible	
				Reset	NoReset		Reset	NoReset
BRAINPC6	6907	6900	200	366	**(3)**	3.46e-01	3.47e-01	**(3)**
BRAINPC7	6907	6900	**(2)**	**(2)**	**(3)**	**(2)**	**(2)**	**(3)**
BRAINPC8	6907	6900	**(2)**	**(2)**	**(2)**	**(2)**	**(2)**	**(2)**
BRAINPC9	6907	6900	**(2)**	314	**(3)**	**(2)**	3.51e-01	**(3)**
C-RELOAD	342	284	60	54	44	-1.03e+00	-1.03e+00	-1.03e+00
CLNLBEAM	303	200	21	25	21	3.50e+02	4.13e+02	3.50e+02
CORKSCRW	906	700	146	146	146	4.44e+01	4.44e+01	4.44e+01
COSHFUN	61	20	**(1)**	957	**(1)**	**(1)**	-7.99e-01	**(1)**
CRESC100	6	200	**(1)**	671	**(1)**	**(1)**	5.69e-01	**(1)**
CRESC132	6	2654	**(4)**	**(4)**	**(4)**	**(4)**	**(4)**	**(4)**
CRESC50	6	100	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**
CVXQP1	1000	500	11	11	11	1.09e+06	1.09e+06	1.09e+06
CVXQP2	1000	250	13	13	13	8.20e+05	8.20e+05	8.20e+05
CVXQP3	1000	750	13	12	13	1.36e+06	1.36e+06	1.36e+06
DALLASL	906	667	100	100	100	-2.00e+05	-2.00e+05	-2.00e+05
DALLASM	196	151	49	49	49	-4.53e+04	-4.53e+04	-4.53e+04
DECONVC	61	1	70	99	70	1.11e-05	9.67e-07	1.11e-05
DISCS	36	66	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
DITTERT	601	521	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**
DIXCHLNV	100	50	19	19	19	1.44e-19	1.52e-19	1.44e-19
DNIEPER	61	24	12	12	**(3)**	1.87e+04	1.87e+04	**(3)**
DRUGDIS	604	400	48	54	48	4.27e+00	4.26e+00	4.27e+00
DRUGDISE	603	500	**(1)**	**(4)**	**(1)**	**(1)**	**(4)**	**(1)**
DUAL1	85	1	16	16	16	3.50e-02	3.50e-02	3.50e-02
DUAL2	96	1	14	15	14	3.37e-02	3.37e-02	3.37e-02
DUAL3	111	1	16	16	16	1.36e-01	1.36e-01	1.36e-01
DUAL4	75	1	13	13	13	7.46e-01	7.46e-01	7.46e-01
DUALC1	9	215	66	66	66	6.16e+03	6.16e+03	6.16e+03
DUALC2	7	229	42	42	42	3.58e+03	3.58e+03	3.58e+03
DUALC5	8	278	21	21	21	4.27e+02	4.27e+02	4.27e+02
DUALC8	8	503	32	32	32	1.83e+04	1.83e+04	1.83e+04
EG3	101	200	31	38	**(3)**	1.28e-01	1.28e-01	**(3)**
EIGMAXB	101	101	36	52	36	-7.79e-02	-4.72e-02	-7.79e-02
EIGMINA	101	101	25	25	25	1.00e+00	1.00e+00	1.00e+00
EIGMINB	101	101	36	52	36	7.79e-02	4.72e-02	7.79e-02
ELATTAR	7	102	59	65	58	1.43e-01	1.43e-01	1.43e-01
EXPFITB	5	102	43	43	43	5.02e-03	5.02e-03	5.02e-03
EXPFITC	5	502	188	178	188	2.33e-02	2.33e-02	2.33e-02
FEEDLOC	90	259	126	126	126	1.93e-08	1.93e-08	1.93e-08
GAUSSELM	819	1926	257	244	115	-1.30e+01	-1.35e+01	-1.28e+01
GILBERT	1000	1	38	42	38	4.82e+02	4.82e+02	4.82e+02
GMNCASE1	175	300	9	9	9	2.67e-01	2.67e-01	2.67e-01
GMNCASE2	175	1050	10	10	10	-9.94e-01	-9.94e-01	-9.94e-01
GMNCASE3	175	1050	9	9	9	1.53e+00	1.53e+00	1.53e+00
GMNCASE4	175	350	27	27	27	5.95e+03	5.95e+03	5.95e+03
GOFFIN	51	50	46	47	46	1.28e-05	1.28e-05	1.28e-05
GOULDQP2	699	349	2	2	2	2.53e-04	2.53e-04	2.53e-04
GOULDQP3	699	349	15	15	15	2.03e+00	2.03e+00	2.03e+00

continued on next page

continued from previous page

Problem	n	m	# of function evaluations			final function value		
			Infeasible	Feasible		Infeasible	Feasible	
				Reset	NoReset		Reset	NoReset
GPP	250	498	15	21	27	1.44e+04	1.44e+04	1.44e+04
GRIDNETA	924	484	22	21	22	1.35e+02	1.35e+02	1.35e+02
GRIDNETC	924	484	19	19	19	7.64e+01	7.64e+01	7.64e+01
GRIDNETD	924	484	20	21	20	1.48e+02	1.48e+02	1.48e+02
GRIDNETF	924	484	20	22	20	8.79e+01	8.79e+01	8.79e+01
GRIDNETG	924	484	21	22	21	1.48e+02	1.48e+02	1.48e+02
GRIDNETI	924	484	28	28	28	8.79e+01	8.79e+01	8.79e+01
GROUPING	100	125	**(2)**	**(2)**	**(2)**	**(2)**	**(2)**	**(2)**
HADAMARD	401	1010	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**
HAGER4	201	100	17	17	17	2.80e+00	2.80e+00	2.80e+00
HAIFAL	343	8958	**(2)**	**(2)**	**(2)**	**(2)**	**(2)**	**(2)**
HAIFAM	99	150	262	141	270	-4.50e+01	-4.50e+01	-4.50e+01
HANGING	300	180	41	45	45	-6.20e+02	-6.20e+02	-6.20e+02
HELSBY	1408	1399	2	2	2	5.77e+01	5.77e+01	5.77e+01
HET-Z	2	1002	25	26	25	1.00e+00	1.00e+00	1.00e+00
HIMMELBI	100	12	54	55	54	-1.74e+03	-1.74e+03	-1.74e+03
HUES-MOD	1000	2	**(1)**	408	**(1)**	**(1)**	3.48e+07	**(1)**
HUESTIS	1000	2	**(1)**	403	**(1)**	**(1)**	3.48e+10	**(1)**
HVYCRASH	404	300	38	32	44	-2.19e-01	-2.18e-01	-2.19e-01
HYDROELL	1009	1008	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
HYDROELM	505	504	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
HYDROELS	169	168	493	504	499	-3.56e+06	-3.57e+06	-3.56e+06
KISSING	211	903	190	105	208	8.43e-01	8.46e-01	8.43e-01
KSIP	20	1001	34	34	34	5.76e-01	5.76e-01	5.76e-01
LAKES	90	78	**(3)**	38	**(3)**	**(3)**	3.51e+05	**(3)**
LEAKNET	156	153	2	2	2	1.53e+01	1.53e+01	1.53e+01
LHAIFAM	99	150	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**
LINSPANH	97	33	10	10	10	-7.70e+01	-7.70e+01	-7.70e+01
LISWET1	403	400	8	8	8	1.04e+00	1.04e+00	1.04e+00
LISWET10	403	400	8	8	8	1.06e+00	1.06e+00	1.06e+00
LISWET11	403	400	7	7	7	1.04e+00	1.04e+00	1.04e+00
LISWET12	403	400	9	9	9	3.06e+01	3.06e+01	3.06e+01
LISWET2	403	400	8	8	8	1.05e+00	1.05e+00	1.05e+00
LISWET3	403	400	8	8	8	1.06e+00	1.06e+00	1.06e+00
LISWET4	403	400	8	8	8	1.05e+00	1.05e+00	1.05e+00
LISWET5	403	400	8	8	8	1.05e+00	1.05e+00	1.05e+00
LISWET6	403	400	8	8	8	1.05e+00	1.05e+00	1.05e+00
LISWET7	403	400	8	8	8	1.05e+00	1.05e+00	1.05e+00
LISWET8	403	400	8	8	8	1.22e+00	1.22e+00	1.22e+00
LISWET9	403	400	9	9	9	2.83e+01	2.83e+01	2.83e+01
LUBRIF	751	500	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
MADSSCHJ	201	398	55	**(1)**	78	-4.99e+03	**(1)**	-4.99e+03
MANNE	300	200	64	9	64	-9.74e-01	-9.74e-01	-9.74e-01
MINC44	583	519	39	32	51	9.95e-04	1.00e-03	9.95e-04
MINPERM	583	520	90	91	90	1.19e-03	9.37e-04	1.19e-03
MODEL	1542	38	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
MOSARQP1	900	30	11	11	11	-3.80e+02	-3.80e+02	-3.80e+02
MOSARQP2	900	30	10	10	10	-5.10e+02	-5.10e+02	-5.10e+02

continued on next page

continued from previous page

Problem	n	m	# of function evaluations			final function value		
			Infeasible	Feasible		Infeasible	Feasible	
				Reset	NoReset		Reset	NoReset
MRIBASIS	36	55	50	55	50	1.82e+01	1.82e+01	1.82e+01
NCVXQP1	1000	500	55	54	55	-7.16e+07	-7.16e+07	-7.16e+07
NCVXQP2	1000	500	50	49	50	-5.78e+07	-5.78e+07	-5.78e+07
NCVXQP3	1000	500	67	72	67	-3.08e+07	-3.08e+07	-3.08e+07
NCVXQP4	1000	250	50	48	50	-9.40e+07	-9.40e+07	-9.40e+07
NCVXQP5	1000	250	54	51	54	-6.63e+07	-6.63e+07	-6.63e+07
NCVXQP6	1000	250	90	87	90	-3.46e+07	-3.47e+07	-3.46e+07
NCVXQP7	1000	750	40	37	40	-4.34e+07	-4.34e+07	-4.34e+07
NCVXQP8	1000	750	55	59	55	-3.05e+07	-3.05e+07	-3.05e+07
NCVXQP9	1000	750	68	**(3)**	68	-2.15e+07	**(3)**	-2.15e+07
NGONE	100	1273	314	**(1)**	217	-6.43e-01	**(1)**	-6.41e-01
OET1	3	1002	36	24	36	5.38e-01	5.38e-01	5.38e-01
OET2	3	1002	252	56	151	8.72e-02	8.72e-02	8.72e-02
OET3	4	1002	23	23	23	4.51e-03	4.51e-03	4.51e-03
OET4	4	1002	51	49	56	4.30e-03	8.56e-01	4.30e-03
OET5	5	1002	122	311	118	2.66e-03	2.66e-03	2.67e-03
OET6	5	1002	82	146	83	8.72e-02	8.72e-02	8.72e-02
OET7	7	1002	100	**(1)**	627	2.09e-03	**(1)**	2.09e-03
OPTCDEG2	302	200	30	34	**(3)**	2.37e+02	2.37e+02	**(3)**
OPTCDEG3	302	200	22	40	**(3)**	4.76e+01	4.76e+01	**(3)**
OPTMASS	610	505	26	15	28	-1.26e-01	-1.26e-01	-1.26e-01
ORTHREGE	999	662	168	268	168	1.32e+02	1.32e+02	1.32e+02
ORTHREGF	680	225	37	23	37	9.19e+00	9.19e+00	9.19e+00
POWELL20	1000	1000	596	596	596	5.21e+07	5.21e+07	5.21e+07
PRIMAL1	325	85	22	22	22	-3.50e-02	-3.50e-02	-3.50e-02
PRIMAL2	649	96	21	21	21	-3.37e-02	-3.37e-02	-3.37e-02
PRIMAL3	745	111	21	21	21	-1.36e-01	-1.36e-01	-1.36e-01
PRIMAL4	1489	75	22	22	22	-7.46e-01	-7.46e-01	-7.46e-01
PRIMALC1	230	9	17	17	17	-6.15e+03	-6.15e+03	-6.15e+03
PRIMALC2	231	7	14	14	14	-3.52e+03	-3.52e+03	-3.52e+03
PRIMALC5	287	8	19	19	19	-4.27e+02	-4.27e+02	-4.27e+02
PRIMALC8	520	8	27	29	27	-1.83e+04	-1.83e+04	-1.83e+04
PRODPL0	60	29	38	38	38	5.88e+01	5.88e+01	5.88e+01
PRODPL1	60	29	31	31	31	3.57e+01	3.57e+01	3.57e+01
PT	2	501	26	26	26	1.78e-01	1.78e-01	1.78e-01
QPCBLEND	83	74	9	9	9	-7.52e-03	-7.52e-03	-7.52e-03
QPCBOEI1	384	351	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**
QPCBOEI2	143	166	233	233	233	8.17e+06	8.17e+06	8.17e+06
QPCSTAIR	467	356	269	269	269	6.20e+06	6.20e+06	6.20e+06
QPNBLEND	83	74	12	12	12	-9.12e-03	-9.12e-03	-9.12e-03
QPNBOEI1	384	351	155	155	155	6.75e+06	6.75e+06	6.75e+06
QPNBOEI2	143	166	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**
QPNSTAIR	467	356	328	328	328	5.15e+06	5.15e+06	5.15e+06
READING1	202	100	52	55	52	-1.60e-01	-1.60e-01	-1.60e-01
READING2	303	200	12	12	12	-1.25e-02	-1.25e-02	-1.25e-02
READING3	202	101	35	35	35	-1.53e-01	-1.53e-01	-1.53e-01
READING4	501	500	140	77	140	-2.89e-01	-2.89e-01	-2.89e-01
READING5	501	500	6	6	6	-2.30e-10	-2.40e-10	-2.30e-10

continued on next page

continued from previous page

Problem	n	m	# of function evaluations			final function value		
			Infeasible	Feasible		Infeasible	Feasible	
				Reset	NoReset		Reset	NoReset
READING6	102	50	20	20	20	-1.45e+02	-1.45e+02	-1.45e+02
READING7	1002	500	41	41	41	-1.24e+03	-1.24e+03	-1.24e+03
READING9	402	200	15	15	15	-4.38e-02	-4.38e-02	-4.38e-02
ROTDISC	905	1081	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
SAWPATH	583	774	15	24	**(2)**	7.50e+02	7.50e+02	**(2)**
SINROSNB	1000	999	**(1)**	90	**(3)**	**(1)**	1.41e+00	**(3)**
SIPOW1	2	2000	17	16	17	-1.00e+00	-1.00e+00	-1.00e+00
SIPOW1M	2	2000	17	17	17	-1.00e+00	-1.00e+00	-1.00e+00
SIPOW2	2	2000	29	29	29	-1.00e+00	-1.00e+00	-1.00e+00
SIPOW2M	2	2000	34	34	34	-1.00e+00	-1.00e+00	-1.00e+00
SIPOW3	4	2000	25	25	25	5.35e-01	5.35e-01	5.35e-01
SIPOW4	4	2000	29	29	29	2.72e-01	2.72e-01	2.72e-01
SMBANK	117	64	23	**(3)**	23	-7.13e+06	**(3)**	-7.13e+06
SMMPSF	720	263	365	365	365	1.03e+06	1.03e+06	1.03e+06
SOSQP1	200	101	21	23	21	9.06e-05	7.96e-05	9.06e-05
SOSQP2	200	101	21	21	87	-4.87e+01	-4.87e+01	-4.87e+01
SPANHYD	97	33	18	18	18	2.40e+02	2.40e+02	2.40e+02
SREADIN3	202	101	30	30	30	-1.53e-01	-1.53e-01	-1.53e-01
SSEBLIN	194	72	212	213	212	1.62e+07	1.62e+07	1.62e+07
SSEBNLN	194	96	74	74	74	1.89e+07	1.89e+07	1.89e+07
SSNLBEAM	303	200	23	23	23	3.43e+02	3.43e+02	3.43e+02
STATIC3	434	96	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
STCQP1	257	128	11	11	11	4.04e+03	4.04e+03	4.04e+03
STCQP2	257	128	11	11	11	1.43e+03	1.43e+03	1.43e+03
STEENBRA	432	108	147	148	147	1.70e+04	1.70e+04	1.70e+04
STEENBRB	468	108	154	252	**(3)**	9.19e+03	9.08e+03	**(3)**
STEENBRC	540	126	527	489	**(3)**	2.76e+04	2.75e+04	**(3)**
STEENBRD	468	108	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**
STEENBRE	540	126	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**	**(3)**
STEENBRF	468	108	171	296	**(3)**	9.13e+03	8.99e+03	**(3)**
STEENBRG	540	126	390	616	**(3)**	2.75e+04	2.74e+04	**(3)**
STNQP1	257	128	10	10	10	-4.47e+03	-4.47e+03	-4.47e+03
STNQP2	257	128	11	11	11	-7.23e+03	-7.23e+03	-7.23e+03
SVANBERG	1000	1000	18	22	23	1.67e+03	1.67e+03	1.67e+03
SWOPF	83	92	18	13	18	6.79e-02	6.79e-02	6.79e-02
TFI1	3	101	45	50	46	5.33e+00	5.33e+00	5.33e+00
TFI2	3	101	12	12	12	1.14e+00	1.14e+00	1.14e+00
TFI3	3	101	25	25	25	4.30e+00	4.30e+00	4.30e+00
TRAINF	808	402	355	345	346	3.08e+00	3.08e+00	3.08e+00
TRAINH	808	402	485	441	486	1.23e+01	1.23e+01	1.23e+01
TRIMLOSS	142	75	78	43	78	9.06e+00	9.06e+00	9.06e+00
UBH1	909	600	852	**(1)**	**(3)**	1.12e+00	**(1)**	**(3)**
UBH5	110	70	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
YAO	202	200	9	9	9	2.02e+01	2.02e+01	2.02e+01
YORKNET	312	256	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**
ZAMB2	1326	480	56	37	56	-4.14e+00	-4.14e+00	-4.14e+00
ZAMB2-10	270	96	40	40	40	-1.58e+00	-1.58e+00	-1.58e+00
ZAMB2-11	270	96	30	30	30	-1.12e+00	-1.12e+00	-1.12e+00

continued on next page

continued from previous page								
Problem	n	m	# of function evaluations			final function value		
			Infeasible	Feasible		Infeasible	Feasible	
				Reset	NoReset		Reset	NoReset
ZAMB2-8	138	48	28	28	28	-1.53e-01	-1.53e-01	-1.53e-01
ZAMB2-9	138	48	32	33	32	-3.55e-01	-3.55e-01	-3.55e-01
ZIGZAG	604	500	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**	**(1)**

Table 3: Comparison of Infeasible and Feasible versions of NITRO in terms of function evaluations: **(1)** Maximum number of iterations (1000) reached; **(2)** Maximum allowable CPU time (60 minutes) reached; **(3)** Terminate because trust region radius, $\Delta \leq 10 \times \epsilon_{\text{mach}}$, where ϵ_{mach} is unit roundoff error; **(4)** Other abnormal termination.

References

[1] P. Armand, J.-Ch. Gilbert, and S. Jan-Jégou. A feasible BFGS interior point algorithm for solving strongly convex minimization problems. *SIAM Journal on Optimization*, 11:199–222, 2000.

[2] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software*, 21(1):123–160, 1995.

[3] R. H. Byrd. Robust trust region methods for constrained optimization. Third SIAM Conference on Optimization, Houston, Texas, May 1987.

[4] R. H. Byrd, J. Ch. Gilbert, and J. Nocedal. A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89(1):149–185, 2000.

[5] R. H. Byrd, M. E. Hribar, and J. Nocedal. An interior point algorithm for large scale nonlinear programming. *SIAM Journal on Optimization*, 9(4):877–900, 2000.

[6] M. R. Celis, J. E. Dennis, and R. A. Tapia. A trust region strategy for nonlinear equality constrained optimization. In P. T. Boggs, R. H. Byrd, and R. B. Schnabel, editors, *Numerical Optimization 1984*, pages 71–82, Philadelphia, USA, 1985. SIAM.

[7] A. R. Conn, N. I. M. Gould, D. Orban, and Ph. L. Toint. A primal-dual trust-region algorithm for non-convex nonlinear programming. *Mathematical Programming*, 87(2):215–249, 2000.

[8] A. R. Conn, N. I. M. Gould, and Ph. Toint. *Trust-region methods*. MPS-SIAM Series on Optimization. SIAM publications, Philadelphia, PA, USA, 2000.

[9] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A primal-dual algorithm for minimizing a nonconvex function subject to bound and linear equality constraints. In G. Di Pillo and F. Giannessi, editors, *Nonlinear Optimization and Related Topics*, pages 15–50, Dordrecht, The Netherlands, 1999. Kluwer Academic Publishers.

[10] A. S. El-Bakry, R. A. Tapia, T. Tsuchiya, and Y. Zhang. On the formulation and theory of the Newton interior-point method for nonlinear programming. *Journal of Optimization Theory and Applications*, 89(3):507–541, June 1996.

[11] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. J. Wiley and Sons, Chichester, England, 1968. Reprinted as *Classics in Applied Mathematics 4*, SIAM, Philadelphia, USA, 1990.

[12] A. Forsgren and P. E. Gill. Primal-dual interior methods for nonconvex nonlinear programming. *SIAM Journal on Optimization*, 8(4):1132–1152, 1998.

[13] D. M. Gay, M. L. Overton, and M. H. Wright. A primal-dual interior method for nonconvex nonlinear programming. In Y. Yuan, editor, *Advances in Nonlinear Programming*, pages 31–56, Dordrecht, The Netherlands, 1998. Kluwer Academic Publishers.

[14] C. T. Lawrence and A. L. Tits. Nonlinear equality constraints in feasible sequential quadratic programming. *Optimization Methods and Software*, 6(4):265–282, 1996.

[15] C. T. Lawrence and A. L. Tits. A computationally efficient feasible sequential quadratic programming algorithm. Technical report, Institute for Systems Research Technical Report TR 98-46, University of Maryland, College Park, 1998.

[16] E. O. Omojokun. *Trust region algorithms for optimization with nonlinear equality and inequality constraints*. PhD thesis, University of Colorado, Boulder, Colorado, USA, 1989.

[17] M. J. D. Powell. A hybrid method for nonlinear equations. In P. Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*, pages 87–114, London, 1970. Gordon and Breach.

[18] M. J. D. Powell and Y. Yuan. A trust region algorithm for equality constrained optimization. *Mathematical Programming*, 49(2):189–213, 1990.

[19] R. J. Vanderbei and D. F. Shanno. An interior point algorithm for nonconvex nonlinear programming. *Computational Optimization and Applications*, 13:231–252, 1999.

[20] A. Vardi. A trust region algorithm for equality constrained minimization: convergence properties and implementation. *SIAM Journal on Numerical Analysis*, 22(3):575–591, 1985.

[21] H. Yamashita and H. Yabe. Superlinear and quadratic convergence of some primal-dual interior point methods for constrained optimization. *Mathematical Programming, Series A*, 75(3):377–397, 1996.

[22] H. Yamashita, H. Yabe, and T. Tanabe. A globally and superlineraly convergent primal-dual point trust region method for large scale constrained optimization. Technical report, Mathematical Systems, Inc., Sinjuku-ku, Tokyo, Japan, 1997.