The PATH Solver: A Non-Monotone Stabilization Scheme for Mixed Complementarity Problems

Steven P. Dirkse* Michael C. Ferris*

September 1993

Abstract

The PATH solver is an implementation of a stabilized Newton method for the solution of the Mixed Complementarity Problem. The stabilization scheme employs a path-generation procedure which is used to construct a piecewise-linear path from the current point to the Newton point; a step length acceptance criterion and a non-monotone pathsearch are then used to choose the next iterate. The algorithm is shown to be globally convergent under assumptions which generalize those required to obtain similar results in the smooth case. Several implementation issues are discussed, and extensive computational results obtained from problems commonly found in the literature are given.

1 Introduction

Among the many algorithms for solving nonlinear systems of equations, Newton's method is perhaps the most well known; basic references for it and its many variants are found in [28, 7]. This powerful method has excellent local convergence properties, but may fail to be globally convergent. Several damping strategies which involve searching the Newton direction exist; these enlarge the domain of convergence for Newton's method. Computational experience with linesearch schemes has shown that convergence can be slowed considerably by the requirements these schemes impose. Recently, Grippo, Lampariello, and Lucidi [12, 13] and Ferris and Lucidi [10] have shown that the monotone descent requirement of the linesearch can be relaxed somewhat without sacrificing the global convergence properties. Other references on this subject are Pang, Han, and Rangaraj [31] and Rangaraj [35].

A key generalization of Newton's method is its application to systems of nonsmooth equations. Nonsmooth equations arise in a number of mathematical programming applications;

*Computer Sciences Department, University of Wisconsin, Madison, Wisconsin 53706

examples include Wilson's sequential quadratic programming (SQP) method [43, 11, 39], the generalized equations of Robinson [36, 37], and reformulations of variational inequalities (VI) and complementarity problems [39]. While these nonsmooth equations are not F(réchet)–differentiable, they are often B(ouligand)–differentiable[38]; Pang [29] presents a generalization of Newton's method based on this, while Harker and Xiao [19] describe the the results of their computational study of this method as applied to a number of complementarity problems. Other work in this area includes that of Han, Pang, and Rangaraj [14] and a recent article by Pang and Qi [32]. An extensive bibliography can be found in the survey article [18] by Harker and Pang.

The primary aims of this paper are to introduce and describe a stabilization scheme for Newton methods for nonsmooth equations, present a global convergence result for the resulting algorithm, and give the results of an extensive computational study of this algorithm. The Newton method makes use of the path-following technique of Ralph [34]. A novel method of pathsearch damping, similar to the watchdog techniques used in the smooth case, is proposed. Strong global convergence results for this technique are given. The computational results confirm that the method does represent an increase in robustness over the undamped Newton method.

The PATH solver is an implementation of the damped Newton method described in this paper. This solver makes use of the mixed complementarity problem (MCP) formulation defined below.

Definition 1 (MCP) *Given a function $f : \mathbb{R}^n \to \mathbb{R}^n$ and bounds $l, u \in \overline{\mathbb{R}}^n$,*

$$\text{find} \quad z \in \mathbb{R}^n, \;\; w, v \in \mathbb{R}^n_+$$

$$\text{s.t.} \quad \begin{aligned} f(z) &= w - v && \text{(1a)} \\ l \le z &\le u && \text{(1b)} \\ (z - l)^\top w &= 0 && \text{(1c)} \\ (u - z)^\top v &= 0, && \text{(1d)} \end{aligned}$$

where $\overline{\mathbb{R}} := \mathbb{R} \cup \{-\infty, +\infty\}$. The MCP is a very general problem and includes many standard problem instances corresponding to special choices of f, l, and u. To illustrate, we will show how both the zero-finding problem for a system of equations and the standard linear complementarity problem (LCP) can be cast as MCP's.

Example 2 *Zero-finding for systems of equations: find z s.t. $f(z) = 0$.*

Let the function f in our MCP be the given f. The variable z should be unbounded; (1b) leads us to set $l = -\infty$ and $u = +\infty$. Let z be a solution to our MCP; conditions (1c) and (1d) imply that $v = w = 0$, so that (1a) is reduced to $f(z) = 0$, which is exactly as desired.

Example 3 *The standard LCP: find $z \ge 0, Mz + q \ge 0$, s.t. $z^\top(Mz + q) = 0$.*

Let $f(z) := Mz+q$ in the MCP, and set $l=0$ and $u=+\infty$. Assume z solves our MCP. (1b) implies that $z \geq 0$. (1d) implies that $v=0$, which with (1a) implies that $f(z)=w\geq 0$. Since $w=f(z)$ and $l=0$, (1c) implies that $z^{\top}(Mz+q)=0$.

The generalization of the standard non-negatively constrained complementarity problem to the MCP is shown to be particularly useful in practice; models can be defined more succinctly and clearly and solved more robustly and efficiently using this formulation.

The paper is composed of 6 sections. In Section 2, we describe the non-monotone stabilization scheme for Newton's method as implemented in the PATH solver. In Section 3, we present a global convergence result for the PATH algorithm. Section 4 describes the implementation of the PATH solver and its linkage as a GAMS subsystem, while Section 5 gives our computational results (with comparison to other algorithms). Finally, Section 6 outlines our conclusions based on this work.

A word about notation is in order. The Euclidean unit ball is denoted by $\mathbb{B} := \{x \mid \|x\|_2 \leq 1\}$. The non-negative orthant in $\mathbb{R}^n$ is denoted by $\mathbb{R}^n_+$. The transpose of a matrix or vector A is denoted by $A^{\top}$. Assuming the set $C \subset \mathbb{R}^n$ is closed and convex, we denote the projection operator onto the set C as $\pi_C(\cdot)$; $\pi_C(x)$ is the unique point in C which minimizes the Euclidean norm $\|c-x\|_2$ for $c \in C$. The projection of a vector x onto $\mathbb{R}^n_+$ is denoted more simply by x_+.

We write $s \downarrow 0$ to mean $s \to 0, s > 0$. Given a scalar or vector function $h(s)$, we say $h(s) = o(s)$ (as $s \downarrow 0$) if $h(s)/\|s\| \to 0$ in norm as $s \downarrow 0$; similarly, $h(s) = O(s)$ if $h(s)/\|s\|$ is bounded as $s \downarrow 0$. Similar definitions hold for the cases where $s \to \infty$. A function f is Lipschitz of modulus $L \geq 0$ on a subset X_0 of $\mathbb{R}^n$ if $\|f(x)-f(y)\| \leq L\|x-y\| \ \forall x,y \in X_0$. A function f is Lipschitzianly invertible of modulus $L \geq 0$ if f is bijective and its inverse mapping is Lipschitz of modulus L.

2 The PATH solver

The PATH solver is an implementation of a stabilized Newton method for solving the MCP. The general form of the pathsearch technique is due to Ralph [34]. In this section, we describe this technique, along with several theoretical and computational enhancements. In order to do so, it will be convenient to express the MCP as the normal equation

$$f_B(x) = 0, \tag{NE}$$

where f_B is the normal map of Robinson [40] imposed on f by the *rectangular* set $B := \{z \mid l \leq z \leq u\}$:

$$f_B(x) := f(\pi_B(x)) + x - \pi_B(x).$$

The relationship between NE and MCP is made clear by the transformations $x = z - f(z)$ and $z = \pi_B(x)$; see [40]. Thus, we can view the MCP as the problem of finding a zero of an equation, albeit a potentially non-smooth one. This framework enables us to apply Newton-type techniques from equation solving to find solutions for the MCP.

In the classical Newton's method, the smooth function f is approximated at a point x^k via the linearization A_k defined by

$$A_k(x) := f(x^k) + \nabla f(x^k)(x - x^k). \tag{2}$$

This linearization A_k is said to be a *first order approximation* of f at x_k. The *Newton point* x_N^k is a zero of this approximation, that is, $A_k(x_N^k) = 0$. Assuming non-singularity of the Jacobian matrix, this zero is unique, and it is a conceptually simple task to find it; one merely solves the matrix equation $\nabla f(x^k)d^k = -f(x^k)$. The Newton point is defined by $x_N^k := x^k + d^k$, and the Newton direction by d^k. The next iterate in the Newton process is determined by a linesearch along this direction, that is,

$$x^{k+1} := x^k + \lambda^k d^k,$$

where λ^k satisfies some appropriate conditions. Thus, a linesearch-damped Newton method can be divided into three parts: linearization, direction-finding, and linesearching. Our presentation will be organized similarly; the PATH solver analogues of these three parts are approximation, path generation and pathsearch damping. The description of the non-monotone stabilization scheme which we have incorporated into the algorithm concludes this section.

2.1 Approximation

Due to the piecewise-linear nature of the projection operator $\pi_B(\cdot)$, it is in general impossible to approximate f_B with a linear function. Instead, a first-order approximation is used, which generalizes the familiar linearization used for smooth functions.

Definition 4 *Let $x^k \in \mathbb{R}^n$. A* first-order approximation *of f_B at x^k is a mapping $A_k : \mathbb{R}^n \mapsto \mathbb{R}^n$ such that*

$$\lim_{x \to x^k} \|f(x) - A_k(x)\| / \left\|x - x^k\right\| = 0.$$

This is expressed more compactly by saying $f(x) - A_k(x)$ is $o(x - x^k)$. A first-order approximation *of f_B on $X_0 \subset \mathbb{R}^n$ is a mapping $\mathcal{A}$ on X_0 such that for each $x \in X_0$, $\mathcal{A}(x)$ is a first-order approximation of f_B at x.*

Let $\mathcal{A}$ be a first-order approximation of f on X_0. $\mathcal{A}$ is a uniform *first-order approximation (with respect to X_0) if there exists $h : (0, \infty) \mapsto [0, \infty]$, with $h(s) = o(s)$, such that for any $x, y \in X_0$,*

$$\|\mathcal{A}(x)(y) - f(y)\| \leq h(\|x - y\|). \tag{3}$$

Note the fundamental difference between first-order approximations to a function at a point and on a set; the approximation on a set is an operator by which approximations at the points in that set can be obtained (e.g. for $x_k \in X_0$, $A_k := \mathcal{A}(x^k)$).

The non-differentiability of the normal map f_B is due to the piecewise-linear nature of the projection operator $\pi_B(\cdot)$. The standard first-order approximation of f_B at x^k is the point-based approximation of Robinson [39] obtained by linearizing f around $\pi_B(x^k)$ and leaving the projection operator alone. This yields

$$A_k(x) := M\pi_B(x) + q + x - \pi_B(x), \tag{4}$$

where

$$M := \nabla f(\pi_B(x^k)) \qquad \text{and} \qquad q := f(\pi_B(x^k)) - M\pi_B(x^k).$$

A Newton point x_N^k is a zero of the approximation A_k. This point may not be unique. However, we will continue to use the notation x_N^k to refer to the unique Newton point found by the path generation technique described in the next section. Much of the difficulty in computing a zero of A_k is caused by the projection operator $\pi_B(\cdot)$, which is nonsmooth. Our method for finding a zero depends on the notion of a path, which we now introduce.

2.2 Path Generation

An essential part of the algorithm is the path constructed between the current point x^k and the Newton point x_N^k. This path generalizes the Newton direction d^k in the smooth case, and serves two purposes: it provides us with the Newton point, and it is the backbone of a pathsearch scheme which serves to damp our Newton method and improve on its convergence properties. This piecewise linear path is constructed using pivotal techniques; each pivot step results in a new linear piece of the path. In this section, we describe a parametric method used to construct the desired path from x^k to x_N^k. However, we first describe the equivalence between the approximation A_k of (4) and another system more amenable to pivotal techniques, and we review Lemke's method as a type of path construction technique.

Instead of attempting to find a zero of the approximation A_k directly, this approximation is cast as a linear MCP, and solved using a pivotal technique. This technique yields a path to the Newton point x_N^k; furthermore, there is a simple relationship between the variables used in the pivotal technique and those of (NE) which allows an easy transition from points $x \in \mathbb{R}^n$ to points $z \in B \subset \mathbb{R}^n$, and vice versa. This will be crucial in the pathsearch stage of the algorithm. Set

$$\begin{aligned} z &= \pi_B(x), \\ v &= (x - z)_+, \\ w &= (z - x)_+. \end{aligned} \tag{5}$$

Since v and w are the positive and negative parts of $x - z$, it follows that $v - w = x - z$ and

$$x = z - w + v,$$

where

$$w \geq 0, \qquad v \geq 0, \qquad w^\top v = 0 \tag{6a}$$
$$z \in B, \tag{6b}$$
$$z = \pi_B(z - w + v). \tag{6c}$$

Definition 5 *Let $B \subset \mathbb{R}^n$ be rectangular, and $x \in \mathbb{R}^n$.*

1. *The vectors z, w, and v defined by (5) are said to be the* components *of x.*
2. *Vectors z, w, and v satisfying (6) are said to* comprise *x; (z, w, v) is called a* triple.

It is clear that there is a 1-1 correspondence between triples (z, w, v) and the points $x \in \mathbb{R}^n$; a triple (z, w, v) comprises x precisely when z, w, and v are the components of x. Moreover, the vector x solves $f_B(x) = 0$ exactly when its components solve MCP, and vice versa. Indeed, the approximation A_k can be written using the components of x to obtain

$$\begin{aligned} 0 = A_k(x) &= M\pi_B(x) + q + x - \pi_B(x) \\ &= Mz + q - w + v, \end{aligned} \tag{7}$$

where (z, w, v) comprise x. It is in the form of (7) that the zero of the approximation A_k is computed. In the course of solving (7), valid triples (z, w, v) are maintained throughout; the vectors x comprised by these triples form a path.

We now define a path formally, using the definition from [34].

Definition 6 *A* path *in $\mathbb{R}^n$ is a continuous function $p : [0, T] \mapsto \mathbb{R}^n$, where $T \in [0, 1]$. The Newton path satisfies the following additional conditions:*

$$p^k(0) = x^k, \tag{8a}$$
$$A_k(p^k(t)) = (1 - t) f_B(x^k), \quad \forall\, t \in [0, T]. \tag{8b}$$

The path p^k may be denoted simply by p when the context makes the meaning clear. Note that (8b) implies that the norm of the approximation at points on p decreases linearly as a function of $1 - t$, and that the point $p(1)$ is a Newton point. To avoid ambiguity, we will assume that the notation x^k_N refers to this Newton point, which is unique, if it exists. Note also that (8a) requires that the path begin at the current point x^k. When the feasible set $B = \mathbb{R}^n_+$, the approximation (7) reduces to the linear complementarity problem (LCP), to which Lemke's method can be applied. We now consider Lemke's method as a path construction technique.

In Lemke's method [24, 6], an extra column (called a covering vector) is added to the matrix M, along with an artificial variable λ. Typically, the covering vector is taken to be the unit vector e (the vector whose components are all ones). This vector is introduced to achieve feasibility for an augmented system, while also maintaining complementarity in the original variables, that is, (7) is replaced by

$$\begin{bmatrix} M & -I & e \end{bmatrix} \begin{bmatrix} z \\ w \\ \lambda \end{bmatrix} = -q \tag{9}$$
$$z, w, \lambda \geq 0,$$

where $(z, w, v \equiv 0)$ comprise x. A ray start is performed, in which λ_0 is set to $\min\{\lambda \mid \lambda \geq 0,\ e\lambda + q \geq 0\}$. This ray start leads directly to an initial basic feasible solution (BFS) [5] of the system (9). Note that the variables z and w are feasible for the LCP only if $\lambda = 0$. In general, λ will be basic in the initial BFS, with value $\lambda_0 > 0$, as a result of the ray start. Thus, Lemke's method specifies pivoting rules which determine a sequence of entering and leaving variables and BFS which maintain the complementarity of z and w. The algorithm terminates successfully when a pivot results in λ leaving the basis at 0. At this point, the LCP has been solved; the original variables z and w are both complementary *and* feasible. The solution to this LCP is the Newton iterate, and a path parameterized by $t = 1 - \frac{\lambda}{\lambda_0}$ leading from the initial BFS to the Newton iterate has been constructed by the sequence of Lemke pivots. At every point in this path, z and w comprise a vector x. Unfortunately, the Lemke path is not quite what is needed, since in general it does not include the current point (z^k, w^k), violating condition (8a).

In the general case, the approximation (7) is expressed using the triple (z, w, v); any path p from x^k to x_N^k can be expressed as a triple by letting $(z(t), w(t), w(t))$ be the components of $p(t)$, that is,

$$p(t) := z(t) - w(t) + v(t),$$

for all $t \in [0, T]$. The requirements for a feasible path (8) require that

$$(1 - t) f_B(x^k) = A_k(p(t))$$

or applying (7) that

$$(1 - t)r = Mz(t) + q - w(t) + v(t) \tag{10}$$

where $r := f_B(x^k)$ is the "residual" vector at the start of the path. Clearly, setting $p(0) = x^k$ satisfies (10), while the triple $(z(1), w(1), v(1))$ comprises the Newton point x_N^k. The path from x^k to x_N^k is now determined by a sequence of pivots, which are analogous to the pivots used in Lemke's method above and which we now describe.

In the PATH solver we use r as the covering vector. Thus in the general case, (9) becomes:

$$\begin{aligned} \begin{bmatrix} M & -I & I & r \end{bmatrix} \begin{bmatrix} z \\ w \\ v \\ t \end{bmatrix} &= -q + r \\ l \leq z &\leq u \\ w, v &\geq 0 \\ 0 \leq t &\leq 1 \end{aligned} \tag{11}$$

The initial BFS is determined by the triple (z^k, w^k, v^k), where $t = 0$. If the triple is nondegenerate (i.e., for all $j \in 1, \ldots, n$, exactly one of z_j^k, w_j^k, and v_j^k is not at a bound), then the choice of basis corresponding to the triple is unique; the basis consists of columns corresponding to variables not at bound. The *first* entering variable is always t, which enters the basis at its lower bound 0, and forces a variable to leave the basis. The leaving variable, chosen by a ratio test, must be one of four types, and determines the choice of entering variable according to the following **pivot rules**:

- w_j: If w_j leaves the basis, the next entering variable will be z_j, which will enter at its lower bound l_j.
- v_j: If v_j leaves the basis, the next entering variable will be z_j, which will enter at its upper bound u_j.
- z_j: If z_j leaves the basis at lower bound, w_j enters at 0. If z_j leaves at upper bound, v_j enters at 0.
- t: If t leaves the basis at upper bound 1, the Newton point x_N^k has been computed and can be recovered from the basis.

The choice of entering variable drives a new pivot step. The path-construction algorithm continues taking pivot steps, using the pivot rules indicated above, until t leaves the basis at 1 (successful termination), t leaves at lower bound, or the ratio test results in no leaving variable (ray termination). Note that once t enters the basis, the lower bound of 0 for t may be relaxed or ignored. Relaxing this bound has proved useful in practice; some of the linearizations solved admit a Newton point only after a sequence of pivots in which t oscillates and takes values less than 0. Each pivot step described above results in a new (linear) piece of the path. Thus, it is possible for a path to have a very large number of pieces; consequently, it may be quite expensive to store. Since the techniques used for storing the path depend upon how the path is to be searched, we defer a discussion of path storage techniques to Section 2.3, where the pathsearch is considered.

It is possible that the basis corresponding to the triple (z^k, w^k, v^k) be rank deficient. In this case, it is impossible to construct the path from the current point to the Newton point by the path generation method described; instead, the PATH solver constructs a path from a new point to the Newton point. This new point is chosen so as to correspond to a basis; for all constrained variables, slack columns are made basic. If there are no free variables, this all-slack basis is guaranteed to be of full rank. Furthermore, is is possible, by a simple choice of the basic values for the slack variables w and v, to duplicate exactly the sequence of pivots performed by Lemke's complementary pivot algorithm. In the case where there are free variables, it may not be possible to choose a full-rank basis corresponding to any valid triple, since the columns corresponding to the free variables must always be in the basis; a sufficient condition is that the principal submatrix M_f corresponding to the free variables be of full rank. In this case, a basis can be obtained by choosing as many slack columns as possible. In [4], Cao and Ferris describe a scheme whereby the lineality of the feasible set B is factored out, and a new problem solved over a reduced space. The full rank condition on M_f is a necessary condition in that context.

Having generated the path, we now return to the nonlinear model and describe our globalization strategy, pathsearch damping.

2.3 Pathsearch Damping

In pathsearch-damped Newton's method for finding a zero of the function f_B, the path from x^k to x_N^k is searched for a point satisfying some descent condition. This condition is often a sufficient reduction in the norm of f_B or in some other *merit function*. These merit functions are nonnegative functions whose zeroes coincide exactly with those of f_B. Thus, while the path is computed in order to find a zero of A_k, the next iterate will be a point on this path yielding a suitable decrease in the merit function. In the smooth case, the Newton direction yields a zero of the approximation and also serves as a descent direction for the merit function [7]. The paths we construct have similar properties.

Recall from Section 2.2, (8b) that the norm of the approximation A_k goes to zero linearly in $(1-t)$ on the path p. We use this and the approximation properties of A_k to show that the norm of f_B must decrease on the path near $t=0$. Let $\sigma \in (0,1)$; then

$$\begin{aligned} \|f_B(p(t))\| &= \|A_k(p(t)) + o(t)\| \\ &= (1-t)\,\|f_B(x_k)\| + o(t) \\ &\leq (1-\sigma t)\,\|f_B(x_k)\|, \end{aligned} \tag{12}$$

for some $\bar{t} \in (0,1)$ and all $t \in [0,\bar{t})$. Thus, the norm of f_B decreases on a section of the path near 0, so that p is a "descent path" for f_B. Note that the relaxation parameter $\sigma < 1$, so that the norm of f_B for acceptable points on p does not have to be as small as predicted by the approximation A_k; in practice, σ will be chosen to be close to 0, so that almost any decrease in $\|f_B\|$ will be sufficient for acceptance.

For solving $f(x) = 0$ (Example 2), one possible merit function $\Theta(\cdot)$ is

$$\Theta(x) := \frac{1}{2} f(x)^\top f(x),$$

the norm function of f. In this case the Newton direction $d = -\nabla f(x^k)^{-1} f(x^k)$ is a descent direction for Θ; note that

$$\begin{aligned}
\Theta'(x^k)\ d &= f(x^k)^\top \nabla f(x^k)\ d \\
&= f(x^k)^\top \nabla f(x^k)(-\nabla f(x^k)^{-1}) f(x^k) \\
&= -f(x^k)^\top f(x^k) \\
&< 0.
\end{aligned}$$

In order to find a value of t which satisfies (12), an Armijo search [1, 7] can be performed on the path p. In a typical implementation of this technique, a parameter $\gamma \in (0,1)$ is chosen, and the points $p(1),\ p(\gamma^1),\ p(\gamma^2),\ \ldots$ are tried, until a value of t is found for which

$$\|f_B(p(t))\| \leq (1 - \sigma t)\, \|f_B(x_k)\|\,.$$

In the smooth case, the path p consists of the line between x^k and x^k_N, so that both storing p and computing $p(t)$ are trivial tasks. This is not the case when p is piecewise linear; in this case, it is necessary to modify the standard linesearch techniques to accommodate the special form of the path. In [34], Ralph suggests two approaches to pathsearching. The first, called the *forward pathsearch*, checks that the descent condition (12) is satisfied as the path is being constructed. Assuming that each pivot step in the path generation algorithm results in an increase in the value of t from t_{old} to t_{new}, the forward pathsearch ensures that t_{new} satisfies (12). The path generation / pathsearch routine terminates when the Newton point is found, or when a pivot step results in an unacceptable value of t_{new}. In the latter case, the line segment between $p(t_{\text{old}})$ and $p(t_{\text{new}})$ is searched for an acceptable point, which becomes the next iterate x^{k+1}. The primary advantage of the forward pathsearch lies in its simplicity; the path is searched as it is constructed, so that it does not need to be stored.

As reported in [34], the chief drawback of the forward pathsearch lies in the fact that the search begins on the wrong end of the path. When the Newton point $p(1)$ is acceptable, all the function evaluations performed in checking that the descent condition is satisfied during path construction are essentially wasted. Also, the forward pathsearch is too restrictive in the sense that an acceptable Newton point may exist at the end of a path which has been terminated due to a failure to satisfy (12) at an intermediate point on the path. Since we wish to accept the Newton step as often as possible, it seems reasonable to check the Newton point first. Recognizing this, Ralph [34] suggests a *backward pathsearch*, in which the path is constructed without checking the descent condition. Instead, a list is made, with each element in the list containing the values of the variables at a breakpoint in the path. When the path has been fully constructed, the endpoint is checked. If this point satisfies (12), this point is accepted as the next iterate; if not, a recursive bisection search is carried out on the list. This bisection search checks (12) for a point near the center of the list; if this point is acceptable, the second half of the list is searched; if not, the first half is searched. In this way, the Newton point is checked first, when it is part of the path. Ralph demonstrates that this leads to fewer function evaluations. Unfortunately, the backward pathsearch also requires that the sequence of pivot steps be recorded. This may require a large amount of space, which cannot be estimated before path construction. The amount of storage required at each pivot step is $O(n)$, while the number of pivot steps may be exponential in n. This is a serious drawback for large-scale problems. Also, a bisection of the list may not lead to a bisection of the current section of the path to be searched; the change in t at each pivot varies widely and non-uniformly.

Motivated by the success of the backwards pathsearch in reducing the number of necessary function evaluations and in increasing the chance of accepting the Newton point, we have implemented a *backtracing* pathsearch. As in the backwards pathsearch, we construct the path without searching it; path generation is completed when the Newton point is found or when ray termination occurs, but not when t oscillates or when the descent criteria are violated. However, instead of saving all of the variable values at each pivot, only information about the entering variable is stored, on a stack which grows with the path. When path generation terminates at a point $p(T)$, the only information about the path that exists is the current basis and a record of the entering variables which led to this basis. At this point, the backtracing pathsearch traces the path in the reverse direction from that of its construction, using the information about the entering variables from the stack to "unpivot", i.e., to reconstruct the breakpoints of the path, along with their associated bases. Backtracing ends when an acceptable point is found. Backtracing is essentially as expensive as is constructing the path; however, this expense is only incurred when a backtrace is necessary. When the point $p(T)$ is acceptable, the information about the path (which can be saved quite cheaply) can be thrown away, without any real backtracing taking place. Also, in this case, at most one function evaluation will be performed.

The forward pathsearch requires little storage, at the cost of added computation (in the form of function evaluations) and reduced robustness. The backward pathsearch requires a minimal number of function evaluations and increases robustness, at the cost of a large storage requirement. The backtracing pathsearch possesses the advantages of both of the above methods, while its drawback (the computational cost of reconstructing parts of the path) is evidenced only when $p(T)$ is unacceptable and a non-trivial pathsearch must be performed. The non-monotone stabilization techniques discussed in the next section serve to reduce the number of pathsearches performed and make the backtracing pathsearch an even better choice.

2.4 Non-Monotone Stabilization

In a linesearch-damped Newton method, the line from x^k to x^k_N is searched for a point satisfying some descent condition, usually expressed in terms of a decrease in some merit function. Implementations of these methods invariably require a monotonic decrease in this merit function, although there is evidence which indicates that this requirement may impede or block convergence to the solution of the equation [12, 13, 10]. Various non-monotone stabilization (NMS) schemes for Newton's method have been proposed, each seeking to improve efficiency by relaxing the requirement of monotone descent. The PATH solver implements a scheme of this type, modified to incorporate a pathsearch rather than a linesearch.

The NMS scheme implemented makes use of a *watchdog* technique to reduce the number of pathsearches performed, and allows a non-monotonic decrease in the merit function associated with the points chosen as a result of these pathsearches. The number of pathsearches is reduced by taking a *d-step* in the majority of cases. A d-step is acceptable if the point returned by the path generation procedure is suitably close to the current point. The measure of closeness, Δ, decreases as the algorithm progresses. In order to monitor these steps, the non-monotone descent criteria for the merit function are checked at least once every $\bar{n}$ number of iterations. The current merit function value is compared with a *reference value* $\mathcal{R}$, which is computed from previous function values. Steps in which these checks on the current merit function value occur are called *m-steps*. The points at which these criteria are checked *and satisfied* are called *check* points. An m-step is also taken when a d-step is unacceptable, that is, when it is too large. A watchdog step occurs when descent criteria are violated; when this occurs, the algorithm returns to the most recent check point, re-generates the path from the check point (if necessary), and backtraces the path until the non-monotone descent criteria are satisfied.

For future reference we introduce a new index j which is set initially to $j = 0$ and incremented each time we define a new check point. If $\ell(j)$ is the index of the jth check point, then we indicate by $\{x^{\ell(j)}\}$ the sequence of check points (where the merit function has been evaluated) and by $\{\mathcal{R}_j\}$ the sequence of reference values associated with the check points. Each check point $x^{k+1} := p^k(t_k)$ is chosen so that the step length t_k satisfies equation (NmD) below, a generalization of a descent condition for the monotone linesearch: given a reference value $\mathcal{R} \geq \left\| f_B(x^k) \right\|$, the step length t_k satisfies

$$\left\| f_B(p^k(t)) \right\| \leq (1 - \sigma t)\, \mathcal{R}. \tag{NmD}$$

If T_k satisfies (NmD), then the pathsearch will choose the step length $t_k := T_k$. If not, we require that the step length be chosen to be large enough, in some sense. This is accomplished by making the technical assumption that t_k be at least τ times as large as the largest interval $[0, T]$ on which (NmD) is satisfied, for some $\tau \in (0, 1)$. Thus, we require that the step length t_k satisfy the following:

$$\begin{array}{c} \text{(NmD) holds for } T_k \text{ implies } t_k := T_k; \text{ otherwise,} \\ \exists\, \tau \in (0,1) \text{ s.t. } t_k \geq \tau \sup\{T \mid \text{(NmD) holds } \forall\, t \in [0, T]\}. \end{array} \tag{NmPs}$$

Note that the backtracing pathsearch described in Section 2.3 yields a value t_k which satisfies (NmPs). To see this, note that, given a linear segment of the path from $p(t_{\text{old}})$ to $p(t_{\text{new}})$ for which (NmD) holds at t_{old} but not at t_{new}, the segment can be searched using an Armijo technique for a point at which (NmPs) is satisfied, where τ depends on the pathsearch parameters used.

In order to complete the description of the algorithm we must specify the rule employed for updating $\mathcal{R}_j$, the reference value for the merit function. This is initially set to $\|f_B(x^0)\|$.

Whenever a point $x^{\ell(j)}$ is generated such that $\left\| f_B(x^{\ell(j)}) \right\| < \mathcal{R}_j$, the reference value is updated by taking into account the memory (that is, a fixed number $m(j) \leq \bar{m}$ of previous values) of the merit function. To be precise, we require the updating rule for $\mathcal{R}_{j+1}$ to satisfy the following condition.

Reference Updating Rule: Given $\bar{m} \geq 0$, let $m(j+1)$ be such that

$$m(j+1) \leq \min\left[m(j)+1, \bar{m}\right],$$

let

$$\mathcal{M}_{j+1} := \max_{0 \leq i \leq m(j+1)} \left\| f_B(x^{\ell(j+1-i)}) \right\|, \tag{13}$$

and choose the value $\mathcal{R}_{j+1}$ to satisfy

$$\left\| f_B(x^{\ell(j+1)}) \right\| \leq \mathcal{R}_{j+1} \leq \mathcal{M}_{j+1}. \tag{14}$$

These conditions on the reference values include several ways of determining the sequence $\{\mathcal{R}_j\}$ in an implementation of the algorithm. For example, any of the following updating rules can be used:

$$\mathcal{R}_{j+1} = \mathcal{M}_{j+1} = \max_{0 \leq i \leq m(j+1)} \left\| f_B(x^{\ell(j+1-i)}) \right\|, \tag{15}$$

$$\mathcal{R}_{j+1} = \max\left[\left\| f_B(x^{\ell(j+1)}) \right\|, \frac{1}{m(j+1)+1} \sum_{i=0}^{m(j+1)} \left\| f_B(x^{\ell(j+1-i)}) \right\| \right], \tag{16}$$

$$\mathcal{R}_{j+1} = \min\left[\mathcal{M}_{j+1}, \frac{1}{2}\left(\mathcal{R}_j + \left\| f_B(x^{\ell(j+1)}) \right\| \right) \right]. \tag{17}$$

We note that (15) is the easiest to satisfy and is used in the PATH solver, while (16) and (17) define conditions which guarantee "mean descent".

We should stress at this point that the stabilization technique differs from standard linesearch techniques in two ways. Firstly, the acceptance criteria for the pathsearch are relaxed significantly by replacing the current merit function value by a reference value, typically taken to be the maximum over a fixed number of previous merit function values. Secondly, the pathsearch is skipped entirely when the Newton point is close to the current point (within the d-step tolerance Δ) and an m-step is not required.

The algorithm can be outlined as follows:

Algorithm PATH

1) [Initialization] Let x^0, $\bar{n} >= 1$, $\Delta = \bar{\Delta} > 0$, $\beta \in (0,1)$ be given:

   set $k = 0$, check_point $= 0$, j $= 0$, $\Delta_0 = \Delta$, $\mathcal{R}_0 = \|f_B(x^0)\|$.

2) If $f_B(x^k) = 0$, stop.

3) Using the approximation A_k, generate a path $p^k : [0, T_k] \mapsto \mathbb{R}^n, T_k \in (0, 1]$, satisfying (8).

4) If ($k <$ check_point $+ \bar{n}$) then

   d-step:
   if ($\left\|p^k(T_k) - p^k(0)\right\| < \Delta$), the step is small enough; accept it:
   set $x^{k+1} := p^k(T_k)$;
   set $\Delta = \Delta * \beta$;
   else the step is too large; go to m-step:

   else

   m-step:
   if ($\left\|f_B(p^k(T_k))\right\| \leq (1 - \sigma T_k)\mathcal{R}_j$), accept the step:
   set $x^{k+1} := p^k(T_k)$;
   else perform a watchdog step:
   set $k =$ check_point, $\Delta = \Delta_j$;
   if necessary, generate the path p^k from x^k to $p^k(T_k)$;
   backtrace p^k to find $t_k \in (0, T_k]$ satisfying (NmPs); set $x^{k+1} := p^k(t_k)$;
   increment j; update $\mathcal{R}_j$; set $\Delta_j = \Delta$; set check_point $= k + 1$.

5) Increment k, and go to Step 2.

In Step 3 above, it is assumed that T_k is as large as possible, i.e., that if $T_k < 1$, A_k is not continuously invertible near $p^k(T_k)$. For practical reasons and robustness, the PATH solver checks whether the function f_B is defined at a given point before accepting that point. This check, not described in the algorithm above, yields a function value at each point. When this function value is computed after a d-step and found to be lower than the reference value, the reference value and check point are updated. If the function at the prospective new iterate is undefined, a watchdog step is performed in the same manner as that performed for a failing m-step.

3 A Global Convergence Result

In this section, we present a global convergence result for the PATH solver. This result generalizes the work of Ralph [34] through the addition of the watchdog technique described earlier. Before doing so, we include, without proof, a path lifting result from Ralph [34] which guarantees the existence of the paths used in our algorithm.

Lemma 7 *Let $\Phi : X \mapsto Y$, $x \in X$ and $\Phi(x) \neq 0$. Suppose the restricted mapping $\bar{\Phi} := \Phi \mid_U : U \mapsto V$ is continuously invertible, where U and V are neighborhoods of x and $\Phi(x)$, respectively. If U is open, and $\epsilon > 0$ is such that $\Phi(x) + \epsilon\,\mathbb{B} \subset V$, then, for $0 \leq T \leq \min\{\frac{\epsilon}{\|\Phi(x)\|}, 1\}$, the unique path p of domain $[0, T]$ such that*

$$p(0) = x$$
$$\Phi(p(t)) = (1-t)\,\Phi(x) \quad \forall\, t \in [0, T]$$

is given by

$$p(t) = \bar{\Phi}^{-1}((1-t)\Phi(x)) \quad \forall\, t \in [0, T].$$

We now present our main result, which gives the convergence properties of the PATH algorithm. Note that (A3), the third assumption below, is a technical one which states that the domains of the paths used by the algorithm can be closed.

Theorem 8 *Let $f : \mathbb{R}^n \mapsto \mathbb{R}^n$ be continuous, $\alpha_0 > 0$ and $X_0 := \{x \in \mathbb{R}^n \mid \|f_B(x)\| \leq \alpha_0\}$. Let $\sigma, \beta \in (0, 1)$, $\bar{\Delta} > 0$, and $\bar{m}, \bar{n} \in \mathbb{N}$ be the parameters governing the pathsearch, and let $X_{\bar{\Delta}\bar{n}} := X_0 + \bar{\Delta}\bar{n}\,\mathbb{B}$. Suppose*

(A1) $\mathcal{A}$ is a uniform first-order approximation of f on X_0, i.e., (3) holds.

(A2) $\mathcal{A}(x)$ is uniformly Lipschitzianly invertible near each $x \in X_{\bar{\Delta}\bar{n}}$; i.e., for some δ, ϵ, and $L > 0$ and for each $x \in X_{\bar{\Delta}\bar{n}}$, there exist sets U_x and V_x containing $x + \delta\,\mathbb{B}$ and $f_B(x) + \epsilon\,\mathbb{B}$ respectively, such that $\mathcal{A}(x) \mid_{U_x} : U_x \mapsto V_x$ is Lipschitzianly invertible of modulus L.

(A3) For each $x \in X_{\bar{\Delta}\bar{n}}$ and $T \in (0, 1]$, if $p : [0, T) \mapsto \mathbb{R}^n$ is such that $p(0) = x$ and, for each $t \in [0, T)$, $\mathcal{A}(p(t)) = (1-t)f_B(x)$ and $\mathcal{A}(x)$ is continuously invertible near $p(t)$, then there exists $p(T) := \lim_{t \uparrow T} p(t)$ with $\mathcal{A}(x)(p(T)) = (1-T)f_B(x)$.

Then for any $x^0 \in X_0$, Algorithm PATH produces a sequence $\{x^k\}$ such that either $f_B(x^k) = 0$ for some $k \geq 0$ or the sequence $\{x^k\}$ converges to a zero x^* of f_B at a Q-superlinear rate.

Furthermore, the residuals $f_B(x^k)$ converge to zero, and the sequence of reference values $\{\mathcal{R}_j\}$ converges to zero at an R-linear rate. If for some $c > 0$ the approximation $\mathcal{A}$ satisfies $\|\mathcal{A}(x)(x^*) - f_B(x^*)\| \leq c\|x - x^*\|^2$ on some neighborhood of x^*, the sequence $\{x^k\}$ converges to x^* at a Q-quadratic rate.

Proof Assume that $f_B(x^k) \neq 0$ for each k. We note first that given $x^k \in X_{\bar{\Delta}\bar{n}}$, there exists a unique path $p^k : I \mapsto \mathbb{R}^n$ of largest domain I such that the following hold:

1. $p(0) = x^k$;
2. $I = [0, T]$ for some $T \in (0, 1]$;
3. $\forall\, t \in I, A_k(p(t)) = (1-t) f_B(x^k)$; and
4. $\forall\, t < T, A_k$ is continuously invertible near $p^k(t)$.

To see this, note that Ralph [34] has shown that the sets U_x and V_x in (A2) can be assumed to be open. Let $\hat{A}_k$ be the Lipschitzianly invertible mapping obtained by restricting A_k to the neighborhood U_{x^k} around x^k. (A2), Lemma 7, and (A3) imply the existence of the path described above, where the technical assumption (A3) is used to close the domain of the path.

Thus, the paths required by the algorithm are guaranteed to exist when $\{x^k\} \subset X_{\bar{\Delta}\bar{n}}$. Note that the Path algorithm will construct these paths, and that the backtracing pathsearch described in Section 2.3 will yield a point which satisfies (NmPs). (See note following definition of (NmPs)).

To see that the algorithm is well defined, we need only show that the sequence of iterates remains in $X_{\bar{\Delta}\bar{n}}$, where the pathsearch is well defined. To do this, we show that the algorithm can take only a limited number of bounded steps before the iterates are forced to return to X_0. It will be convenient to define the index

$$\jmath(k) := \max\,[j \mid \ell(j) \leq k].$$

Thus $\ell(\jmath(k))$ is the largest iteration index not exceeding k at which the merit function has been evaluated. For example,

$$\begin{array}{rcllllllllll}
k & = & 0 & 1 & 2 & \ldots & & 10 & 11 & \ldots & & 57 & \ldots & \ldots \\
j & = & 0 & & & & & 1 & & & & 2 & & \ldots \\
\ell(j) & = & 0 & & & & & 10 & & & & 57 & & \ldots \\
\jmath(k) & = & 0 & & & \ldots & 0 & 1 & & \ldots & 1 & 2 & 2 & \ldots
\end{array}$$

We use the notation

$$d^k := p^k(T_k) - p^k(0)$$

to denote the difference between the initial and terminal points of the path p^k. The d^k above should not be confused with the notation for the search direction d used in the smooth case; rather, $\left\|d^k\right\|$ is the size of a possible d-step.

If the point x^{k+1} is a check point, then it has been generated as the result of an m-step, so that

$$\left\|f_B(x^{k+1})\right\| < \mathcal{R}_{\jmath(k)} \leq \left\|f_B(x^0)\right\|$$

and $x^{k+1} \in X_0$.

If the point x^{k+1} is *not* a check point, then it has been generated as the result of a (bounded) d-step, so that x^{k+1} satisfies

$$x^{k+1} = x^{\ell(\jmath(k))} + \sum_{i=\ell(\jmath(k))}^{k} d^i,$$

where $\|d^i\| \leq \bar{\Delta}$ and $k - \ell(\jmath(k)) < \bar{n}$. Since $x^{\ell(\jmath(k))}$ is a check point, it must be in X_0, so that $x^{k+1} \in X_{\bar{\Delta}\bar{n}}$.

Thus, we have shown that the algorithm is well-defined and that every iterate $x^k \in X_{\bar{\Delta}\bar{n}}$. We now demonstrate the global convergence of our method. We first show that f_B converges to 0 (i.e $\lim_{k\to\infty} \left\| f_B(x^k) \right\| = 0$). This result is used to show convergence of the iterates, and to derive rates for their local convergence.

To show convergence of $\{f_B(x^k)\}$ to zero, the sequence $\{x^k\}$ can be split into two subsequences: $\{x^{\ell(k)}\}$, the points at which a reference value has been defined, and $\{x^{r(k)}\}$, the remainder of the points.

If the sequence $\{x^{r(k)}\}$ is finite, then the algorithm will eventually take only m-steps. Once this point is reached, a pathsearch is performed at each iteration, and there can be no further watchdog steps taken. Ralph [34] shows that in this case, the residual norms $\left\| f_B(x^k) \right\|$ converge linearly to zero.

Assume then that $\{x^{r(k)}\}$ is an infinite sequence. We show first that for large enough k, $x^{r(k)} \in X_0$. Recall that $x^{r(k)}$ is the result of a d-step, so that $x^{r(k)} = x^{r(k)-1} + d^{r(k)-1}$, where $d^{r(k)-1}$ is bounded as follows:

$$\left\| d^{r(k)-1} \right\| \leq \bar{\Delta}\ \beta^k.$$

Thus, $\lim_{k\to\infty} \left\| d^{r(k)-1} \right\| = 0$. Choose K so that $\left\| d^{r(k)-1} \right\| \leq \hat{\xi}\ \forall\, k \geq K - \bar{n}$, where $\hat{\xi}$ is such that $h(\xi) \leq \dfrac{\xi}{L}\ \ \forall\, \xi \leq \hat{\xi}$. (Recall from Definition 4 that $h(s)$ is $o(s)$.) The Lipschitz invertibility of $\mathcal{A}$ yields

$$\|p(t) - p(0)\| = \left\| \hat{A}_k^{-1}((1-t)f_B(x^k)) - \hat{A}_k^{-1}(f_B(x^k)) \right\| \leq Lt \left\| f_B(x^k) \right\|, \tag{18}$$

so that for all $k \leq K - \bar{n}$, we have, by the uniformity of $\mathcal{A}$,

$$\begin{aligned}
\left\| f_B(p(T_{r(k)-1})) \right\| &\leq \left\| A_{r(k)-1}(p(T_{r(k)-1})) \right\| + h(\left\| d^{r(k)-1} \right\|) && \text{by (3)} \\
&\leq (1 - T_{r(k)-1}) \left\| f_B(x^{r(k)-1}) \right\| + \frac{1}{L} \left\| d^{r(k)-1} \right\| && \text{by (8b)} \\
&\leq (1 - T_{r(k)-1}) \left\| f_B(x^{r(k)-1}) \right\| + T_{r(k)-1} \left\| f_B(x^{r(k)-1}) \right\| && \text{by (18)} \\
&\leq \left\| f_B(x^{r(k)-1}) \right\|.
\end{aligned}$$

Since the PATH algorithm takes at most $\bar{n}$ d-steps before taking an m-step, and we have shown previously that all the points resulting from m-steps are in X_0, the above result shows that $x^{r(k)} \in X_0$ for $k \geq K$.

We now show that the sequence $\{\mathcal{R}_j\}$ converges linearly to 0. Recall from the algorithm description that the number of consecutive d-steps is bounded above by $\bar{n}$, after which an m-step must occur. Let $\{x^{s(k)}\}$ be the sequence of iterates which have occurred as the result of an m-step, but whose predecessors have occurred as the result of a d-step. The algorithm requires that $x^{s(k)} := p^{s(k)-1}(T_{s(k)-1})$ satisfies

$$\left\| f_B(x^{s(k)}) \right\| \leq (1 - \sigma T_{s(k)-1}) \mathcal{R}_{\jmath(s(k)-1)}. \tag{19}$$

Since $x^{s(k)-1}$ is the result of d-step, $\left\| f_B(x^{s(k)-1}) \right\| \leq \alpha_0$ for large enough k, where α_0 is an upper bound for $\|f_B\|$ on the level set X_0. Since by (A2), $A_{s(k)-1}$ is continuously invertible in an ϵ-neighborhood of $f_B(x^{s(k)-1})$, we can use Lemma 7 to show that

$$T_{s(k)-1} \geq \min\{\frac{\epsilon}{\alpha_0}, 1\},$$

thus bounding $(1 - \sigma T_{s(k)-1})$ away from 1. We now need only show that a result similar to (19) holds for $x^{\ell(k)}$ when $x^{\ell(k)}$ is the result of *consecutive* m-steps. This is precisely what Ralph ([34], Theorem 9) has shown in proving convergence for his algorithm; this and (19) imply that for large enough k,

$$\left\| f_B(x^{\ell(k)}) \right\| \leq (1 - \sigma \hat{T}) \mathcal{R}_{\jmath(\ell(k)-1)} \tag{20}$$

holds for some $\hat{T} \in (0, 1)$. Applying (20) and our rule (15) for updating the reference values, we have

$$\mathcal{R}_{\jmath(\ell(k+\bar{m}))} \leq (1 - \sigma \hat{T}) \mathcal{R}_{\jmath(\ell(k))},$$

which gives us R-linear convergence of the reference values at the rate of $(1 - \sigma \hat{T})^{\frac{1}{\bar{m}}}$. The entire sequence $\{f_B(x^k)\}$ converges to 0 as well, since for large enough k,

$$\mathcal{R}_{\jmath(\ell(k))} \geq \left\| f_B(x^i) \right\| \quad \text{for } i \geq \ell(k),$$

since all d-steps following iteration $\ell(k)$ result in a decrease in $\|f_B\|$, while all m-steps following $\ell(k)$ result in iterates x^i at which $\mathcal{R}_{\jmath(\ell(k))} > \|f_B(x^i)\|$.

Thus, we have established that $\{\left\| f_B(x^k) \right\|\}$ converges to 0. We show now that, after a certain point, the algorithm takes only Newton steps. Let $\gamma := \min\{\epsilon, \frac{\hat{\xi}}{L}\}$, where ϵ is defined in (A2) and $\hat{\xi}$ is such that $h(\xi) \leq \dfrac{\xi(1-\sigma)}{L}$ $\forall\, \xi \leq \hat{\xi}$. Let K be chosen so that $\left\| f_B(x^k) \right\| \leq \gamma$ for $k \geq K$. Then for $k \geq K$, the following hold:

$$\left\| f_B(x^k) \right\| \leq \epsilon, \tag{21}$$

$$\left\| f_B(x_N^k) \right\| \leq (1 - \sigma) \left\| f_B(x^k) \right\|, \tag{22}$$

where (21) follows directly from the choice of γ. To see (22), note that (21) and Lemma 7 imply that the PATH algorithm finds the Newton point $x_N^k := p^k(1)$, so that by the uniformity of $\mathcal{A}$,

$$\begin{aligned} \left\| f_B(x_N^k) \right\| &\leq \left\| A_k(x_N^k) \right\| + h(\left\| d^k \right\|) = h(\left\| d^k \right\|) \\ &\leq h(L \left\| f_B(x^k) \right\|) && \text{by (18)} \\ &\leq \frac{L \left\| f_B(x^k) \right\| (1-\sigma)}{L} && \text{by choice of } \gamma \\ &\leq (1-\sigma) \left\| f_B(x^k) \right\| . \end{aligned}$$

Hence by (22), $x^{k+1} = x_N^k$ for $k \geq K$.

We show now that the sequence $\{x^k\}$ is Cauchy. For $k \geq K$,

$$\begin{aligned} \left\| x^{k+1} - x^k \right\| &= \left\| \hat{A}_k^{-1}(0) - \hat{A}_k^{-1}(f_B(x^k)) \right\| \\ &\leq L \left\| f_B(x^k) \right\| \\ &\leq L(1-\sigma)^{k-K} \left\| f_B(x^K) \right\| , \end{aligned}$$

the last equality following from (22). Choosing $s \geq r \geq K$ implies that

$$\begin{aligned} \| x^s - x^r \| &\leq \sum_{k=r}^{\infty} \left\| x^{k+1} - x^k \right\| \\ &\leq \sum_{k=r}^{\infty} \frac{L \left\| f_B(x^K) \right\|}{(1-\sigma)^K} (1-\sigma)^k \\ &= \frac{L \left\| f_B(x^K) \right\|}{(1-\sigma)^K \sigma} (1-\sigma)^r \to 0 \quad \text{as } r \to \infty. \end{aligned}$$

This implies convergence of $\{x^k\}$. Let $x^* := \lim_{k \to \infty} x^k$. Since $\{\left\| f_B(x^k) \right\|\} \to 0$, the continuity of f_B implies $f_B(x^*) = 0$.

To see the Q-superlinear rate of convergence for the iterates, note that for some $\bar{K} \geq K$, $k \geq \bar{K}$ implies that $x^* \in x^k + \delta\, \mathbb{B}$ (i.e., x^* is in the range of the inverse of the linearization $\hat{A}_k^{-1}$), and the following applies:

$$\begin{aligned} \left\| x^{k+1} - x^* \right\| &= \left\| \hat{A}_k^{-1}(f_B(x^*)) - \hat{A}_k^{-1}(A_k(x^*)) \right\| \\ &\leq L \ \| f_B(x^*) - A_k(x^*) \| && (23) \\ &\leq L \ h(\left\| x^k - x^* \right\|), && (24) \end{aligned}$$

where the last inequality depends on the uniformity of $\mathcal{A}$. Since $h(s)$ is $o(s)$, inequality (24) shows convergence at a Q-superlinear rate.

If the approximation $\mathcal{A}$ also satisfies the inequality

$$\left\|\mathcal{A}(x^k)(x^*) - f_B(x^*)\right\| \leq c\left\|x^k - x^*\right\|^2 \tag{25}$$

for some $c > 0$ and on some neighborhood of x^*, then for large enough k, (23) and (25) together yield

$$\left\|x^{k+1} - x^*\right\| \leq cL\left\|x^k - x^*\right\|^2,$$

so that a quadratic rate of convergence is achieved. □

Note that although Theorem 8 deals with the normal map f_B, the result holds for more general nonsmooth mappings; the restriction to the normal map f_B is made only for the sake of consistency with the rest of the paper.

4 Implementation

The PATH solver was designed and implemented primarily to solve complementarity problems, using the pathsearch damped Newton method described above. The solver is written in the C language, and is designed to operate on its own or as a GAMS subsystem; the current implementation relies on GAMS/CPLIB (a software library developed in part by the authors [9]) to formulate the problems under consideration and provide function and derivative information. This interface with GAMS grants the user a convenient and flexible means of defining the problem to be solved, as well as ready access to a library of complementarity problems already formulated in the GAMS language [8]. Since large, sparse problems are frequently encountered, the PATH solver handles data in a sparse format. It also incorporates a DEVEX-type [20] scheme in the implementation of the ratio test, which increases numerical stability. Furthermore, it is possible, by making the proper choice of basis and basic values, to duplicate the sequence of pivots taken by Lemke's method, without making any changes to the parametric path generation code. This can be done automatically, by setting an option, and makes comparisons with Lemke's method a trivial task. These and other options can be set in an options file. In this section, use of the GAMS interface and options file are discussed further.

4.1 The GAMS Interface

GAMS [3] is an algebraic modeling system which allows models to be expressed in a clear, concise, and easily modified way. Furthermore, it allows the model to be expressed independently of any solution algorithm for it. By using GAMS/CPLIB, MCP's can be formulated and made available to solvers in a highly portable fashion, as described in [9] and illustrated below.

The key elements of the MCP are the function f and the bounds l and u; these determine the problem completely. Fortunately, specifying the bounds is a trivial task. In GAMS, each variable consists of three parts; the level value and explicit lower and upper bounds, which may be infinite. The bounds and initial level values can be assigned values in the GAMS code. Thus, any variables in the model are included with their bounds. The initial level values are also available to a solver, and may serve as an initial iterate, as is the case for the PATH solver.

The way in which the function f is defined is a bit more complicated. Instead of explicitly defining f, the GAMS user defines the complementarity relationship desired; the function f is defined by CPLIB in a manner consistent with this complementarity relationship. This is best illustrated by example. Consider the following quadratic program:

$$\begin{array}{ll} \underset{x}{\text{minimize}} & \frac{1}{2}x^\top Qx + c^\top x \\ \text{subject to} & Ax \leq b \end{array} \tag{QP}$$

Assuming Q is symmetric positive semi-definite, the KKT conditions

$$\begin{array}{rll} Qx + c + A^\top u = 0, & x \text{ free}, & \perp \\ b - Ax \geq 0, & u \geq 0, & \perp \end{array} \tag{KKT}$$

for (QP) are necessary and sufficient, and can be expressed as a MCP, in GAMS form.

Example 9 *KKT conditions in GAMS form:*

```
sets J  / 1 * 8 /   /* row indices for M /*
     I  / 1 * 5 /   /* constraints */
alias (JJ, J) ;
parameters Q(JJ,J),
           c(I),
           A(I,J),
           b(I) ;
$include 'QP.dat'  /* matrix data, etc.  */
variables x(J),  /* primal variables */
          u(I) ;   /* duals */
u.lo(I) = 0;  /* non-negative var */
equations delx(J),
          delu(I);
delx(J) ..  sum (JJ, Q(J,JJ)*x(JJ)) + c(J) + sum(I, u(I)*A(I,J)) =g= 0;
delu(I) ..  b(I) =g= sum(J, A(I,J)*x(J)) ;

model qp  / delx.x, delu.u / ;
option mcp = path ;
solve qp using mcp ;
```

The KKT conditions are expressed as a number of complementarity pairs; each pair consists of a function and an associated variable, along with their respective constraints and bounds. In the GAMS/CPLIB system, the functions are defined by the equation statement, and the variables are associated with the functions in the model statement. It is important to note that in the MCP model formulated by GAMS, constraints for each function are inferred from the variable bounds and complementarity relationship; the inequality used to define each function (the `=g=` in `delx(J)` and `delu(I)` above) is meaningful only as a consistency check.

4.2 Features and Options

As is the case with most implementations of solution algorithms, the PATH solver has evolved considerably during the course of its existence. The number of parameters over which the user has control has been increased, and features have been added, as described in this section. The PATH solver makes use of an options file to set parameter values, flags, and switches.

In its basic mode, the PATH solver performs as indicated in Section 2, finding Newton points of approximations via a path generation technique and choosing the successive iterates by means of a backtracing pathsearch. Most of the computational effort is expended in doing pivots during the path generation phase. A sparse basis updating routine is crucial in minimizing the amount of work done in this phase. The PATH solver uses a sparse matrix package to do the pivoting; the addition of a basis updating scheme is planned. It also incorporates a DEVEX-type [20] scheme in the implementation of the ratio test, which increases numerical stability.

Typically, the initial basis and the residual vector r in the path generation scheme are chosen so as to ensure that the path p begins at x^k. However, as described in Section 2.2, this first basis may be rank deficient; in that case, it may possible to start from a new point corresponding to a full-rank basis, one containing as many slack columns as possible. For the special case of an LCP this new point leads to a sequence of pivots which correspond exactly to those taken by Lemke's method. The user can specify whether to use Lemke's method for all subproblems, for none, or only for those in which the basis chosen initially is rank deficient.

Parameters, flags, and switches are reset using the options file, or, in some cases, by using GAMS `option` statements. The format of the PATH options file `path.opt` is similar to the MINOS options file format described in [27]; it consists of keywords and values. Blank lines are permitted. Any line starting with an asterisk ("*") is ignored (i.e., it is a comment). Each line can set at most one option. Keywords are uniquely determined by their first three characters; the remaining characters are ignored. Boolean values are determined by the first character in the value field; y/n/0/1 are acceptable. Relevant keywords and their meanings are summarized in Tables 1 and 2.

The performance of the non-monotone stabilization technique can be sensitive to the values of the parameters which control it; these parameters can be set via the options file. The user can also exercise considerable control over the amount and type of output produced by the PATH solver through the use of the verbosity flags (described in Table 2) and the *iterlog frequency* option.

Table 1: Options file algorithm parameters

keywords	value	description
lemke first	y/n	The **first** linearization will be solved "Lemke-style"
lemke option	required	**All** linearizations are solved "Lemke-style"
	allowed	When the initial basis is rank-deficient, a "Lemke-style" basis is tried.
	banned	The algorithm will never switch over to a "Lemke-style" basis.
major iterations	integer k	Maximum number of linearizations solved
minor iterations	k	Maximum number of pivots allowed per linearization
nms do	y/n	Use non-monotone stabilization techniques; if not, the point returned by the pathsearch is always accepted. Default: yes.
reference ifactor	real c	The reference value is initialized to $c \cdot \|f_B(x^0)\|$. Default: 5
reference min	c	The reference value is constrained to be no less than c. Default: 20
stepsize ifactor	c	The initial step size $\bar{\Delta}$ is set to $c \cdot \|x^0 - x^1_N\|$. Default: 20
stepsize imin	c	The initial step size in constrained to be no less than c. Default: 20
save best	y/n	Cycling may occur in the path generation code. When save best is on, a point corresponding to the best (highest) t-value is returned. Default: yes
tolerance factor	c	A tolerance (not necessarily a zero tolerance) supplied to the factorization routines. Default: .01
tol slack	c	During the pivoting, the bounds are relaxed by c in order to choose a leaving variable.
tol convergence	c	Convergence is declared when $\|f_B\| < c$. Default: 1.e-9

The GAMS `option` statement is used to set the `reslim` and `iterlim` options. The `iterlim` option sets the limit on the cumulative total of pivots performed during problem solution; the default is 1000. The `reslim` option sets the time limit (in seconds) for the solver; the default is 1000.

Table 2: Options file verbosity parameters

keywords	value	description
iterlog freq	k	Information line logged for every k'th major iteration
vfl backtrace	y/n	for each backtrace performed, log the steps taken
vfl initpoint	y/n	log z_0 and $f(z_0)$
vfl leaves	y/n	for each pivot, log leaving and entering variables
vfl mcp	y/n	for each major iteration, log the linear MCP to be solved
vfl nms	y/n	log actions of the non-monotone stabilization scheme
vfl ratiotest	y/n	for each pivot, the vectors involved in the ratio test are logged
vfl raybase	y/n	when ray termination occurs, log the base of the ray
vfl silent	y/n	keeps some warning/error messages from being logged
vfl solution	y/n	final problem solution is logged
vfl summary	y/n	at solution, summary solver statistics are logged
vfl tprint	y/n	at each pivot, the value of the parameter t is logged

5 Computational Results

This section contains computational results obtained from several complementarity problems considered in the literature. These problems, described in greater detail in [8], have been formulated in GAMS and are available from the authors. The following algorithms are compared:

- **PATH** The PATH solver described in this paper.
- **PATH^b** The PATH solver set to solve the first approximation from the Lemke-type basis described in Section 2.2.
- **J-N** The classic Josephy-Newton's method, as described in [21]. The results shown were obtained by running the PATH solver with the options file set to emulate Josephy-Newton's method.
- **B-DIFF** The B-differentiable equations approach of Harker and Xiao [19], in which each major iteration involves a linesearch of a direction determined by solving a system of equations.
- **NE/SQP** In [30], Pang and Gabriel describe a scheme in which the search direction is determined by solving a quadratic program; this direction is linesearched as well.

Unless otherwise noted, the results for the PATH solver were obtained using default values for all parameters; in no case was the code modified to improve performance for any particular problem. For B-DIFF and NE/SQP we include only results available in the literature [19, 30].

The results presented in this section indicate that the PATH solver improves on the classical Josephy-Newton's method in both speed and robustness. When both algorithms work, they will usually take the same number of major iterations; however, the PATH solver makes use of the current basis to take a "warm start" on each major iteration, thus reducing the number of pivots required. For the larger problems, this has a noticeable effect on the solution times. The nonmonotone stabilization techniques employed are of crucial importance in making the PATH solver robust (see Sections 5.1, 5.3, 5.4, 5.7 and 5.8). Without the use of these techniques, the PATH solver would behave about as well as Josephy-Newton's method; with them, it solves the problems considered from a larger set of initial points, as the theory behind the PATH algorithm would lead us to expect. Also, the PATH solver makes use of the MCP formulation; this greater generality enables problems to be solved more quickly and robustly as well (see Tables 8, 9, 11 and 12). The task of formulating the models and performing the computational experiments was eased greatly by the use of the GAMS environment, which made the formulations easier to write, read, and modify, as well as freeing us from the task of writing gradient evaluation routines. Unfortunately, the PATH solver, like the other algorithms considered in this section, is hampered by the presence of rank deficiency at a solution point. This has been a challenging problem in the past, and is the subject of current research in the area.

5.1 Walrasian Equilibrium Models

In [25], Mathiesen gives a simple example of a 4-variable Walrasian equilibrium model. In this model market, there are three commodities (with their corresponding price variables) and one production process or sector (with its corresponding level of activity). The consumers sell their initial endowment of goods to the producer, and buy back the goods produced. An *equilibrium* is a set of prices and a level of production which satisfy a number of equilibrium conditions on the supplies, demands, prices, and activity levels. The optimal prices are determined only in a relative sense; if the price vector p^* is optimal, then so is the price vector λp^*, for $\lambda > 0$. The price system can be normalized by fixing a numeraire price or by fixing the sum of the prices; because of its greater generality, we have taken the latter approach.

The parameters for Mathiesen's model are the consumer's initial commodity endowment vector b and the budget share parameters $(\alpha, 1-\alpha, 0)$ which determine which production goods the consumer purchases. In all the numerical tests, the initial endowment b_1 is 0; the values used for b_2, b_3, and α are indicated below. Note that for the run with b_3 set to 0.5, the PATH solver finds a solution point different from that found by NE/SQP.

Two other examples of Walrasian equilibrium problems are due to Scarf [41]. In the first model, there are six commodities and eight production sectors; thus, the NCP formulation for this problem contains 14 variables. The second problem is one with 40 variables, 14 corresponding to commodity prices and 26 to production activity levels. The B-DIFF results were obtained using a proximal point modification to that algorithm.

Table 3: Mathiesen's Walrasian Problem

$\alpha = .75, b = (0, 1, .5)$				
algorithm	start point	major (minor)	func / grad	time
PATH	(1, 1, 1, 1)	5 (11)	12 / 6	.09
J-N	" "	failed		(*)
NE/SQP	" "	11		

$\alpha = .75, b = (0, 1, 2)$				
algorithm	start point	major (minor)	func / grad	time
PATH	(1, 1, 1, 1)	5 (5)	6 / 6	.06
J-N	" "	5 (20)	6 / 6	.09
NE/SQP	" "	3		

$\alpha = .9, b = (0, 5, 3)$				
algorithm	start point	major (minor)	func / grad	time
PATH	(2.5, 5.5, 1.5, 4.5)	6 (6)	7 / 7	.07
J-N	" "	4 (16)	6 / 24	.06
B-DIFF	" "	5		
PATH	(3.5, 4.5, 0.5, 4.0)	4 (4)	5 / 5	.04
J-N	" "	4 (16)	5 / 5	.05
B-DIFF	" "	4		
PATH	(2.5, 1.5, 1.5, 3.5)	10 (24)	25 / 11	.19
J-N	" "	failed		(*)
B-DIFF	" "	6		
PATH	(3.5, 6.5, 0.5, 5.5)	5 (5)	6 / 6	.05
J-N	" "	5 (20)	6 / 6	.05
B-DIFF	" "	5		

Table 4: Scarf's Walrasian Problems

algorithm	start point	major (minor)	func / grad	time
PATH	(p_1, y_1)	4 (9)	5 / 5	.29
J-N	" "	4 (71)	5 / 5	.48
B-DIFF	" "	5		
NE/SQP	" "	6		
PATH	(p_2, y_2)	3 (8)	4 / 4	.24
J-N	" "	3 (52)	4 / 4	.36
B-DIFF	" "	5		
PATH	(p_3, y_3)	4 (10)	5 / 5	.31
J-N	" "	4 (77)	5 / 5	.47
NE/SQP	" "	28	/	
$p_1 = (.2, .2, .2, .1, .1, .2),\ y_1 = (.5, 0, 4, 0, 0, 0, .4, 0)$				
$p_2 = (.1, .1, .1, .1, .1, .2),\ y_2 = y_1$				
$p_3 = (1, 1, 1, 1, 1, 1),\ y_3 = (.4, 1, 0, 3, 0, 0, 1, 0)$				

algorithm	start point	major (minor)	func / grad	time
PATH	(p_1, y_1)	3 (36)	4 / 4	1.47
J-N	" "	3 (102)	4 / 4	3.41
NE/SQP	$(p_1, ???)$	3		
$p_1 = (1, \ldots, 1),\ y_1 = (1, \ldots, 1)$				

5.2 A Nash Equilibrium

In [26], a 10-variable Nash equilibrium problem is described; solution techniques for this problem are discussed in [17]. All the methods performed well on this problem; however, since all variables are positive at the solution, the warm start taken by the PATH solver makes it particularly effective.

Table 5: A Nash Problem

algorithm	start point	major (minor)	func / grad	time
PATH	(1, 1, . . . , 1)	6 (6)	7 / 7	.52
J-N	" "	6 (66)	7 / 7	.52
B-DIFF	" "	13		
PATH	(10, 10, . . . , 10)	6 (6)	7 / 7	.52
J-N	" "	6 (66)	7 / 7	.52
B-DIFF	" "	10		

5.3 Small Quadratic NCP's (Kojima-Shindo)

The PATH solver has been tested on two very similar 4-variable NCP's, due to Kojima [22] and Kojima and Shindo [23], whose functions are quadratic. The problem from [22] was also used in numerical tests done by Josephy in [21], and is referred to as the Josephy problem here and in [19].

When previous versions of the PATH solver which utilized a forward pathsearch and did not include the watchdog techniques of the current version were tested on these problems, they tended to perform poorly from some of the start points considered. The stabilization techniques have yielded the PATH solver results shown in Tables 6 and 7. These results were achieved without the heuristic used in [19]: this heuristic has the effect of moving to a basis for which both B-DIFF and the PATH solver have little difficulty.

5.4 A Non-Linear Program from Powell

In [33], Powell gives a nonlinear program containing 5 variables and three equality constraints. This problem can be expressed using inequalities by rewriting the 3 equality constraints as 6 inequalities. If the unbounded variables for this problem are split into their positive and negative parts, then the KKT conditions for this modified problem can be expressed as an NCP in 16 variables. However, if the MCP formulation is used, it is unnecessary to modify the constraints or the variables; the MCP contains only 8 variables, all of which are free, so that we have the special case of a system of non-linear equations.

Table 6: Josephy's Problem

algorithm	start point	major (minor)	func / grad	time
Path	(0, 0, 0, 0)	6 (7)	7 / 7	.08
Path^b	" "	6 (7)	7 / 7	.08
J-N	" "	failed		(*)
B-DIFF	" "	7		
Path	(1, 1, 1, 1)	8 (14)	14 / 9	.12
Path^b	" "	4 (6)	5 / 5	.05
J-N	" "	4 (12)	5 / 5	.06
B-DIFF	" "	10		
Path	(100, 100, 100, 100)	21 (30)	22 / 22	.24
Path^b	" "	10 (12)	11 / 11	.10
J-N	" "	10 (30)	11 / 11	.11
B-DIFF	" "	15		
Path	(1, 0, 1, 0)	24 (49)	25 / 25	.27
Path^b	" "	3 (5)	4 / 4	.05
J-N	" "	3 (9)	4 / 4	.06
B-DIFF	" "	7		
Path	(1, 0, 0, 0)	3 (4)	4 / 4	.04
Path^b	" "	3 (5)	4 / 4	.04
J-N	" "	3 (9)	4 / 4	.05
B-DIFF	" "	7		
Path	(0, 1, 1, 0)	17 (44)	24 / 18	.21
Path^b	" "	4 (6)	5 / 5	.06
J-N	" "	4 (12)	5 / 5	.05
B-DIFF	" "	9		

Table 7: Kojima / Shindo's Problem

algorithm	start point	major (minor)	func / grad	time
PATH	(0, 0, 0, 0)	5 (6)	6 / 6	.06
PATH^b	" "	5 (6)	6 / 6	.07
J-N	" "	failed		(*)
NE/SQP	" "	7		
PATH	(1, 1, 1, 1)	4 (6)	5 / 5	.04
PATH^b	" "	4 (6)	5 / 5	.06
J-N	" "	4 (12)	5 / 5	.06
NE/SQP	" "	7		

More importantly, the smaller problem is not as sensitive to the choice of starting point, as evidenced by the data in Table 8. The starting points given in Table 8 and 9 are the initial values for the free variable x in the original minimization problem; all constraint multipliers were initialized to zero.

Table 8: Powell's NLP, NCP Formulation

algorithm	start point	major (minor)	func / grad	time
PATH	(-2, 2, 2, -1, -1)	9 (16)	10 / 10	.62
J-N		8 (122)	9 / 9	.84
PATH (*)	(-3, 3, 3, -1, -1)	21 (101)	35 / 22	2.20
J-N (*)		5 (29)	6 / 6	.26
PATH	(-5, 5, 5, -1, -1)	10 (17)	11 / 11	.68
J-N (*)		6 (34)	7 / 7	.30

Table 9: Powell's NLP, MCP Formulation

algorithm	start point	major (minor)	func / grad	time
PATH	(-2, 2, 2, -1, -1)	8 (8)	9 / 9	.19
PATH	(-3, 3, 3, -1, -1)	11 (11)	12 / 12	.27
PATH	(-5, 5, 5, -1, -1)	9 (9)	10 / 10	.22

In Table 8, the "(*)" indicates where an algorithm terminated in a solution to the NCP which is known to be non-optimal for the source NLP. While the PATH solver did find a false optimum when running on the NCP, the objective value was better than that found by Josephy-Newton's method when used with the same starting point. Note also that the PATH solver terminated at the global optimum when run on the MCP model. While Harker and Xiao [19] report computational experience using this problem, we have not included their results here. We believe that the complementarity formulation they employed does not accurately reflect Powell's constraints, and that it is unlikely that their solutions correspond to feasible points for the original problem.

5.5 The Hansen–Koopmans Problem

In [15], Hansen and Koopmans consider the problem of determining an invariant capital stock. A *capital stock* is an investment of capital into various production processes for a given period of time; this investment strategy determines what investments are possible during the next time period and the utility earned during the current time period. The total utility is a discounted sum of the utilities from each time period. An *invariant* capital stock is an initial investment of capital which has the property that the discounted sum of the utilities is maximumized when the initial capital stock remains unchanged during each of the following time periods.

The PATH solver was tested on a 14-variable example taken from [15]; all the data in Table 10 were obtained using a discount factor of $\alpha = 0.7$ and the starting points indicated. In addition, we have used a two-stage approach to test the PATH solver and the Josephy-Newton method. First, the problem was solved with the lower bounds on six of the variables set to $\epsilon =$ 1e-05 instead of 0 to prevent function evaluation errors; the solution to this problem was then used as an initial iterate for the original problem. This refinement step was done at a cost of one major and one minor iteration; the solution to the modified problem was close to that for the original.

Table 10: Hansen / Koopmans' Problem

algorithm	start point	major (minor)	func / grad	time
PATH	(0.3, ... , 0.3, 0, 0, 0, 0)	16 (35)	18 / 18	.54
J-N	" "	16 (175)	18 / 18	.79
NE/SQP	" "	10		
PATH	(0.5, ... , 0.5, 0, 0, 0, 0)	16 (33)	18 / 18	.54
J-N	" "	16 (175)	18 / 18	.79
NE/SQP	" "	11		

5.6 A Traffic Assignment Model

In [2], Bertsekas and Gafni describe a traffic assignment problem and its formulation as a NCP in 15 variables. However, this problem can be formulated more concisely as a MCP, using only 5 variables. The problem is one of finding an equilibrium traffic flow between five locations, subject to certain demand constraints. In Tables 11 and 12, we give results comparing solver performance for both of these formulations. Notice that the results in Table 12 are uniformly better than those of Table 11.

Table 11: Traffic Equilibrium, NCP Formulation

algorithm	start point	major (minor)	func / grad	time
PATH	(0, 0, ... , 0)	4 (17)	5 / 5	.21
J-N	" "	4 (55)	5 / 5	.22
B-DIFF	" "	4		
PATH	(0.5, 0.5, ... , 0.5)	4 (6)	5 / 5	.21
J-N	" "	4 (60)	5 / 5	.27
B-DIFF	" "	5		
PATH	(100, 100, ... , 100)	12 (28)	13 / 13	.57
J-N	" "	12 (150)	13 / 13	.68

Table 12: Traffic Equilibrium, MCP Formulation

algorithm	start point	major (minor)	func / grad	time
PATH	(0, ... , 0)	3 (6)	4 / 4	.13
PATH	demand / 0.5	3 (5)	4 / 4	.13
PATH	demand	4 (11)	5 / 5	.18
PATH	(100, 100, ... , 100)	4 (11)	5 / 5	.18

5.7 Spatial Price Equilibriums

In [42], Tobin gives an example of a multi-commodity market modeled as a network. A type of Nash equilibrium point for this market called a *spatial price equilibrium* (SPE) is sought. At this equilibrium point, the market is in a steady state; no player will have a reason to change prices or supply and demand levels. In this example, the prices are variables; the supplies and demands are a function of the prices. The equilibrium conditions can be formulated as an NCP in 42 variables. Interestingly, some of the linear subproblems which arise when solving this model result in a cyclic pattern of bases. In dealing with this, the stabilization techniques (particularly the savebest option) of the PATH solver are useful; the minor iterations limit was set to 50 for these runs.

Table 13: Tobin's SPE Problem

algorithm	start point	major (minor)	func / grad	time
PATH	(0, 0, ... , 0)	9 (43)	10 / 10	.67
PATH^b	" "	9 (43)	10 / 10	.7
J-N	" "	9 (174)	10 / 10	1.50
B-DIFF	" "	16		
NE/SQP	" "	20		
PATH	(1, 1, ... , 1)	17 (413)	18 / 18	13.14
PATH^b	" "	9 (33)	10 / 10	.56
J-N	" "	9 (170)	10 / 10	1.41
B-DIFF	" "	15		
NE/SQP	" "	20		

In another example of spatial competition, Harker [16] gives an example of an oligopolistic market. In this market, the supplies and demands are the variables; these supplies and demands determine the prices. As in the previous example, the conditions for a spatial price equilibrium lead to an NCP, in 27 variables.

Table 14: Harker's SPE Problem

algorithm	start point	major (minor)	func / grad	time
PATH	(0, 0, ... , 0)	7 (34)	8 / 8	.31
J-N	" "	7 (185)	8 / 8	1.28
B-DIFF	" "	10		
PATH	(1, 1, ... , 1)	5 (7)	6 / 6	.17
J-N	" "	5 (134)	6 / 6	.97
B-DIFF	" "	10		

5.8 A Cycling Example

In [34], Ralph gives an example of a NCP for which undamped Newton methods cycle; the pathsearch damping does not suffer from this problem, however. The NCP is based on the familiar example used to motivate linesearch damping for Newton methods for systems of equations, where $f(x) = \tan^{-1}(x)$. While Josephy-Newton's method cycles between major iterations for this problem, the PATH solver solves the problem in 33 major iterations. The convergence in this case is achieved by setting the acceptable step length reduction factor $\beta = 0.99$. The number of major iterations could be reduced by choosing parameter values which result in more restrictive step length acceptance criteria, and an earlier application of the pathsearch.

6 Conclusion

In this paper, we have introduced a novel nonmonotone stabilization technique for Newton's method as applied to nonsmooth equations and particularly to the normal map f_B. Under certain assumptions regarding the approximations to f_B, we have shown in Section 3 that the PATH algorithm constructs a sequence of iterates which converges superlinearly or quadratically to a solution of the normal equation, thus demonstrating the theoretical effectiveness of our method.

The computational results presented in Section 5 demonstrate the effectiveness of the PATH solver in solving complementarity problems. In particular, the stabilization techniques are shown to be effective in increasing the robustness of our method and eliminating the divergence or cycling which might otherwise occur in some of the problems considered. Also, the warm start makes use of the current solution estimate to decrease the number of pivot steps necessary to solve the linear subproblems of the major iterations.

The GAMS model library we have formed has helped greatly in testing the PATH solver. In the future, this library will be used to test solution techniques aiming to deal more effectively with the rank-deficient case.

The authors wish to thank Danny Ralph for helpful comments regarding this paper.

References

[1] L. Armijo. Minimization of functions having Lipschitz-continuous first partial derivatives. *Pacific Journal on Mathematics*, 16:1–3, 1966.

[2] D. P. Bertsekas and E. M. Gafni. Projection methods for variational inequalities with application to the traffic assignment problem. *Mathematical Programming Study*, 17:139–159, 1982.

[3] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, South San Francisco, CA, 1988.

[4] M. Cao and M. C. Ferris. A pivotal method for affine variational inequalities. Technical Report 1114, Computer Sciences Department, University of Wisconsin, Madison Wisconsin 53706, October 1992.

[5] V. Chvatal. *Linear Programming.* W. H. Freeman and Company, New York, 1983.

[6] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and its Applications*, 1:103–125, 1968.

[7] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice–Hall Series in Computational Mathematics. Prentice–Hall, Inc, Englewood Cliffs, New Jersey, 1983.

[8] S. P. Dirkse and M. C. Ferris. Solving complementarity problems via GAMS: A model library. Manuscript, 1993.

[9] S. P. Dirkse, M. C. Ferris, P. V. Preckel, and T. Rutherford. The GAMS callable program library for variational and complementarity solvers. Manuscript.

[10] M. C. Ferris and S. Lucidi. Globally convergent methods for nonlinear equations. Technical Report 1030, Computer Sciences Department, University of Wisconsin, 1991.

[11] R. Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, 1987.

[12] L. Grippo, F. Lampariello, and S. Lucidi. A nonmonotone line search technique for Newton's method. *SIAM Journal of Numerical Analysis*, 23:707–716, 1986.

[13] L. Grippo, F. Lampariello, and S. Lucidi. A class of nonmonotone stabilization methods in unconstrained optimization. *Numerische Mathematik*, 59:779–805, 1991.

[14] S.-P. Han, J.-S. Pang, and N. Rangaraj. Globally convergent Newton methods for nonsmooth equations. *Mathematics of Operations Research*, 17(3):586–607, August 1992.

[15] T. Hansen and T. C. Koopmans. On the definition and computation of a capital stock invariant under optimization. *Journal of Economic Theory*, 5:487–523, 1972.

[16] P. T. Harker. Alternative models of spatial competition. *Operations Research*, 34:410–425, 1986.

[17] P. T. Harker. Accelerating the convergence of the diagonalization and projection algorithms for finite-dimensional variational inequalities. *Mathematical Programming*, 41:29–59, 1988.

[18] P. T. Harker and J.-S. Pang. Finite-dimensional variational inequality and nonlinear complementarity problems: A survey of theory, algorithms, and applications. *Mathematical Programming*, 48(2):161–220, September 1990.

[19] P. T. Harker and B. Xiao. Newton's method for the nonlinear complementarity problem: A B–differentiable equation approach. *Mathematical Programming*, 48:339–357, October 1990.

[20] P. M. J. Harris. Pivot selection methods of the Devex LP code. *Mathematical Programming*, 5:1–28, 1973.

[21] N. H. Josephy. Newton's method for generalized equations. Technical Summary Report 1965, Mathematics Research Center, University of Wisconsin–Madison, Madison, Wisconsin, 1979.

[22] M. Kojima. Computational methods for solving the nonlinear complementarity problem. *Keio Engineering Reports*, 27:1–41, 1974.

[23] M. Kojima and S. Shindo. Extensions of Newton and quasi-Newton methods to systems of PC^1 equations. *Journal of Operations Research Society of Japan*, 29:352–374, 1986.

[24] C. E. Lemke. Bimatrix equilibrium points and mathematical programming. *Management Science*, 11:681–689, 1965.

[25] L. Mathiesen. An algorithm based on a sequence of linear complementarity problems applied to a Walrasian equilibrium model: An example. *Mathematical Programming*, 37:1–18, 1987.

[26] F. H. Murphy, H. D. Sherali, and A. L. Soyster. A mathematical programming approach for determining oligopolistic market equilibrium. *Mathematical Programming*, 24:92–106, 1982.

[27] B. A. Murtagh and M. A. Saunders. MINOS 5.0 user's guide. Technical Report SOL 83-20, Department of Operations Research, Stanford University, CA, 1983.

[28] J. M. Ortega and W. C. Rheinbolt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, 1970.

[29] J.-S. Pang. Newton's method for B–differentiable equations. *Mathematics of Operations Research*, 15(2):311–341, May 1990.

[30] J.-S. Pang and S. A. Gabriel. NE/SQP: A robust algorithm for the nonlinear complementarity problem. *Mathematical Programming*, 1993. to appear.

[31] J.-S. Pang, S.-P. Han, and N. Rangaraj. Minimization of locally lipschitzian functions. *SIAM Journal on Optimization*, 1(1):57–82, February 1991.

[32] J.-S. Pang and L. Qi. Nonsmooth equations: Motivation and algorithms. *SIAM Journal on Optimization*, 3(3):443–465, August 1993.

[33] M. Powell. A method for nonlinear constraints in minimization problems. In R. Fletcher, editor, *Optimization*, pages 283–298. Academic Press, London, 1969.

[34] D. Ralph. Global convergence of damped Newton's method for nonsmooth equations, via the path search. *Mathematics of Operations Research*, 1993. to appear.

[35] N. Rangaraj. *Nonsmooth Optimization: Algorithms and Applications.* PhD thesis, Department of Mathematical Sciences, The Johns Hopkins University, Baltimore, MD, 1990.

[36] S. M. Robinson. Generalized equations and their solution: Part I: Basic theory. *Mathematical Programming Study*, 10:128–141, 1979.

[37] S. M. Robinson. Strongly regular generalized equations. *Mathematics of Operations Research*, 5(1):43–62, 1980.

[38] S. M. Robinson. Local structure of feasible sets in nonlinear programming, Part III: Stability and sensitivity. *Mathematical Programming Study*, 30:45–66, 1987.

[39] S. M. Robinson. Newton's method for a class of nonsmooth equations. Manuscript, Department of Industrial Engineering, University of Wisconsin–Madison, August 1988.

[40] S. M. Robinson. Normal maps induced by linear transformations. *Mathematics of Operations Research*, 17(3):691–714, August 1992.

[41] H. E. Scarf. *The Computation of Economic Equilibria.* Yale University Press, New Haven, Conneticut, 1973.

[42] R. L. Tobin. A variable dimension solution approach for the general spatial equilibrium problem. *Mathematical Programming*, 40:33–51, 1988.

[43] R. B. Wilson. *A Simplicial Algorithm for Concave Programming.* PhD thesis, Graduate School of Business Administration, Harvard University, Boston, MA, 1963.