The Semismooth Algorithm for Large Scale Complementarity Problems

Todd S. Munson • Francisco Facchinei • Michael C. Ferris • Andreas Fischer • Christian Kanzow

Mathematics and Computer Sciences Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, USA

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Buonarroti 12, 00185 Roma, Italy

Computer Sciences Department, University of Wisconsin, 1210 West Dayton Street, Madison, WI 53706, USA

Department of Mathematics, University of Dortmund, 44221 Dortmund, Germany

Institute of Applied Mathematics, University of Hamburg, Bundesstrasse 55, 20146 Hamburg, Germany

tmunson@mcs.anl.gov • soler@dis.uniroma1.it • ferris@cs.wisc.edu • fischer@math.uni-dortmund.de • kanzow@math.uni-hamburg.de

---

Complementarity solvers are continually being challenged by modelers demanding improved reliability and scalability. Building upon a strong theoretical background, the semismooth algorithm has the potential to meet both of these requirements. We discuss relevant theory associated with the algorithm and then describe a sophisticated implementation in detail. Particular emphasis is given to the use of preconditioned iterative methods to solve the (nonsymmetric) systems of linear equations generated at each iteration and robust methods for dealing with singularity. Results on the MCPLIB test suite indicate that the code is reliable and efficient, and scales well to very large problems.

(*Artificial Intelligence; Simulation; Statistical Analysis; Analysis of Algorithms; Queues*)

---

1 Introduction

Operations researchers typically encounter complementarity conditions for the first time in the context of linear programs. In this instance, an optimal solution is characterized by the fact that either the dual variable (multiplier) is zero or the primal slack variable is zero. It is less well known that such (necessary) "optimality conditions" extend not only to the domain of nonlinear programming, but also appear quite naturally in a multiplicity of problems from

economics and engineering (**?** 14). To facilitate the development of such models by general operations researchers, extensions to the AMPL (16**?**) and GAMS (3**?** **?**) modeling languages have been developed for expressing complementarity conditions and a plethora of applications have used such systems to solve realistic problems. Some successes of this approach are documented in (**?** **?** **?**). Many of these solutions have been calculated using "successive linear complementarity" codes such as MILES (**?**) and PATH (9).

Due to the increased ability of complementarity algorithms at solving large, difficult problems, the modeling community has become more adventurous at generating even larger and "harder" models, some of which are poorly defined, suffer from conditioning or singularity problems, or contain "non-convexities". Any new algorithmic development should attempt to meet the expectations of the modeling community; the resulting code must terminate in all cases with appropriate solutions or error messages, and should reliably solve models from a broad range of application areas. It is with this in mind that we have developed a new implementation of a class of nonsmooth Newton methods.

Extensive theoretical research on the use of nonsmooth Newton methods for complementarity problems has been performed in the past few years with much emphasis on extending the domain of local convergence. One algorithmic approach for solving complementarity problems is to reformulate them as piecewise smooth systems of equations and applying an iterative linearization algorithm (22; 23). For example, the normal map reformulation (**?**) forms the basis for both the MILES and PATH codes. While it may be argued that piecewise linear maps are more effective at approximating piecewise smooth maps, generating the "Newton" direction involves the arduous task of solving a linear complementarity problem, typically with a derivative of the pivotal method due to **?**).

A seemingly more attractive approach is to use an algorithm based on solving a single system of linear equations to generate each "Newton" step. Recent theoretical work has outlined a host of methods with this property. Amongst these, the semismooth algorithm (6) appears to have some of the strongest associated theory. The aim of this paper is to develop a robust code based upon this semismooth algorithm for solving large complementarity problems.

We begin by briefly discussing the theoretical foundations of the semismooth algorithm. Many of the results contained in this paper are given without proof; instead, we provide references to the relevant literature. We then present the implementation details of the code. The main focus is on the numerical aspects of the code for solving the (nonsymmetric) systems of

linear equations using preconditioned iterative solvers, methods for dealing with singularity and ill-conditioning, and strategies to recover from finding non-optimal stationary points of the merit function. We test the code on the problems in the MCPLIB (8) test collection, and present results that indicate that the code is reliable and scalable. A comparison with PATH shows comparable robustness.

Before proceeding, we recall the definition of the mixed complementarity problem (MCP), also known as a box constrained variational inequality. Given lower bounds, $\ell_i \in \Re \cup \{-\infty\}$, and upper bounds, $u_i \in \Re \cup \{+\infty\}$, with $\ell_i < u_i$ for all $i \in I := \{1, \ldots, n\}$ and a continuously differentiable function,$F : \Re^n \to \Re^n$, we say that $x^* \in \Re^n \cap [\ell, u]$ solves MCP(F, ℓ, u) if and only if one of the following holds for all $i \in I$:

$$\begin{aligned} x_i^* = \ell_i \quad &\text{and} \quad F_i(x^*) \geq 0, \\ x_i^* \in (\ell_i, u_i) \quad &\text{and} \quad F_i(x^*) = 0, \\ x_i^* = u_i \quad &\text{and} \quad F_i(x^*) \leq 0. \end{aligned}$$

Two special cases of the general mixed complementarity problem are systems of nonlinear equations, $F(x) = 0$, obtained by taking $\ell_i = -\infty$ and $u_i = \infty$ for each i, and the (standard) nonlinear complementarity problem, where for each i, $\ell_i = 0$ and $u_i = \infty$.

As an example of a general mixed complementarity problem, consider the bound constrained optimization problem

$$\min f(x) \text{ subject to } \ell \leq x \leq u$$

where $f : \Re^n \to \Re$ is a twice continuously differentiable convex function. The first order optimality conditions of this problem are precisely MCP$(\nabla f, \ell, u)$. We can easily see the meaning of the complementarity conditions in a one dimensional setting. If we are at an unconstrained stationary point, then $\nabla f(x) = 0$. Otherwise, if x is at its lower bound, then the function must be increasing as x increases, so $\nabla f(x) \geq 0$. Further, if x is at its upper bound, then the function must be increasing as x decreases, so that $\nabla f(x) \leq 0$.

For fast convergence of iterative methods for smooth nonlinear equations, the invertiblity of the Jacobian at the limit point is typically assumed. A generalization of this concept to the nonsmooth case is that of *strong regularity* (31**?**). We say that a solution x^* of MCP(F, ℓ, u) is strongly regular if the submatrix $F'(x^*)_{\alpha\alpha}$ is nonsingular and the Schur-complement

$$F'(x^*)_{\alpha\cup\beta,\alpha\cup\beta}/F'(x^*)_{\alpha\alpha} := F'(x^*)_{\beta\beta} - F'(x^*)_{\beta\alpha}F'(x^*)_{\alpha\alpha}^{-1}F'(x^*)_{\alpha\beta}$$

has positive principal minors. Here, the index sets α and β are defined as

$$\alpha := \{i : \ell_i < x_i^* < u_i, F_i(x^*) = 0\}, \quad \beta := \{i : x_i^* \in \{\ell_i, u_i\}, F_i(x^*) = 0\}.$$

The definition of strong regularity may look odd at first sight, but it is well known that it is a natural extension of the concept of nonsingularity of the Jacobian in the case of systems of equations. In fact, it is not difficult to check that if $\ell_i = -\infty$ and $u_i = \infty$ for all i, so that MCP(F, ℓ, u) reduces to the system of equations $F(x) = 0$, strong regularity of a solution x^* corresponds precisely to the nonsingularity of $F'(x^*)$. We note that strong regularity is typically used in proofs of the local convergence of "Newton" type methods for complementarity problems.

In this paper we make the standard blanket assumption that F is continuously differentiable on $\Re^n$.

2 Mathematical Foundation

The algorithm discussed in this paper is based on a reformulation of the mixed complementarity problem as a semismooth system of equations. In this section we present some basic definitions related to semismoothness, followed by the reformulation used and a statement of the basic algorithm.

2.1 Semismooth Functions

Semismooth functions were introduced by **(author?)** (25) and have been subsequently extended to vector valued functions (29; 28). Unless otherwise noted, all of the definitions and results in this section are taken from these references. In order to define the semismooth property, let $G : \Re^n \to \Re^n$ be a locally Lipschitzian function and note that by Rademacher's theorem G is differentiable almost everywhere. If we indicate by D_G the set where G is differentiable, we can define the B-subdifferential of G at x as

$$\partial_B G(x) := \left\{ H \in \Re^{n \times n} : \exists \{x^k\}, x^k \in D_G, \text{ with } \lim_{x^k \to x} G'(x^k) = H \right\},$$

and the **(author?)** (5) subdifferential of G at x as

$$\partial G(x) := \text{co } \partial_B G(x),$$

where co C denotes the convex hull of a set C.

Definition 2.1 *Let $G : \Re^n \to \Re^n$ be locally Lipschitzian at $x \in \Re^n$. We say that G is* semismooth *at x if*

$$\lim_{\substack{H \in \partial G(x+tv') \\ v' \to v, t \downarrow 0}} Hv' \tag{1}$$

exists for all $v \in \Re^n$. G is termed a semismooth function *if G is semismooth at all $x \in \Re^n$.*

Semismooth functions lie between Lipschitz functions and continuously differentiable functions. It can be shown that if G is semismooth at x then it is also directionally differentiable there with the directional derivative in the direction v given by the limit in (1). The class of semismooth functions is rather broad. Examples include real valued functions that are continously differentiable or convex, and those functions that can be represented as the maximum or minimum of a finite number of semismooth functions. Note that $G : \Re^n \to \Re^n$ is semismooth at x if and only if all its component functions are semismooth at x. Furthermore, the composition of two semismooth functions is also semismooth.

A slightly stronger notion than semismoothness is strong semismoothness.

Definition 2.2 *Suppose that G is semismooth at x. We say that G is* strongly semismooth *at x if for any $H \in \partial G(x + d)$, and for any $d \to 0$,*

$$Hd - G'(x; d) = O(\|d\|^2).$$

Once again, G is strongly semismooth if and only if all its component functions are strongly semismooth. In turn, continuously differentiable functions with locally Lipschitz Jacobians, l_p norms, and the max and min of affine functions are all examples of strongly semismooth functions.

In the study of the local convergence of algorithms for solving semismooth systems of equations, the following regularity condition plays a role similar to that of the nonsingularity of the Jacobian in the study of algorithms for smooth systems of equations.

Definition 2.3 *We say that a semismooth function $G : \Re^n \to \Re^n$ is* BD-regular *at x if all the elements in $\partial_B G(x)$ are nonsingular.*

The importance of the above definitions lies in the fact that it is possible to extend many results related to Newton's method for smooth systems of equations to semismooth systems. In fact, a generalized Newton method for the solution of a semismooth system of equations, $G(x) = 0$, can be defined as

$$x^{k+1} = x^k - (H^k)^{-1} G(x^k), \qquad H^k \in \partial_B G(x^k); \tag{2}$$

where H^k can be any element in $\partial_B G(x^k)$. The following result then holds.

Theorem 2.4 *Suppose that x^* is a solution of the system $G(x) = 0$ and that G is semismooth and BD-regular at x^*. Then the iteration in (2) is well defined and convergent to x^* superlinearly in a neighborhood of x^*. If, in addition, G is directionally differentiable in a neighborhood of x^* and strongly semismooth at x^*, then the convergence rate of (2) is quadratic.*

This simple scheme forms the basis of the semismooth method.

2.2 Semismooth Algorithm

The notion of an NCP-function is used to reformulate the complementarity problem as a semismooth system of equations. A mapping $\phi : \Re^2 \to \Re$ is called an *NCP-function* if it satisfies the condition

$$\phi(a, b) = 0 \Longleftrightarrow a \geq 0, b \geq 0, ab = 0.$$

Two examples of such functions are the Fischer-Burmeister (15) function

$$\phi_{FB}(a, b) := \sqrt{a^2 + b^2} - a - b \tag{3}$$

and the penalized Fischer-Burmeister (4) function

$$\phi_{CCK}(a, b) := \lambda \left(\sqrt{a^2 + b^2} - a - b\right) - (1 - \lambda) \max\{0, a\} \max\{0, b\}, \tag{4}$$

where $\lambda \in (0, 1)$ is a given parameter. We can easily verify that the Fischer-Burmeister function is an NCP-function by performing some simple algebraic manipulations (after noting that $\phi_{FB}(a, b) = 0$ if and only if $\sqrt{a^2 + b^2} = a + b$). In the case of the penalized Fischer-Burmeister function we note that outside of the interior of the positive orthant it is the Fischer-Burmeister function (multiplied by λ). Thus, to show ϕ_{CCK} is an NCP-function, we only need to show that $\phi_{CCK}(a, b)$ cannot be zero in the interior of the first orthant. But since on the interior of the first orthant Φ_{FB} is negative, this easily follows from the definition. These two functions will play a central role in this paper.

To reformulate the complementarity problem using these NCP-functions, we partition the index set $I = \{1, \ldots, n\}$ in the following way:

$$\begin{aligned} I_\ell &:= \{i \in I \mid -\infty < \ell_i < u_i = +\infty\}, \\ I_u &:= \{i \in I \mid -\infty = \ell_i < u_i < +\infty\}, \end{aligned}$$

$$
\begin{aligned}
I_{\ell u} &:= \{i \in I \mid -\infty < \ell_i < u_i < +\infty\}, \\
I_f &:= \{i \in I \mid -\infty = \ell_i < u_i = +\infty\}.
\end{aligned}
$$

That is, I_ℓ, I_u, $I_{\ell u}$ and I_f denote the set of indices $i \in I$ with finite lower bounds only, finite upper bounds only, finite lower and upper bounds and no finite bounds on the variable x_i, respectively. Hence, the subscripts in the above index sets indicate which bounds are finite, with the only exception of I_f which contains the free variables.

If ϕ_1, ϕ_2 are two (not necessarily different) NCP-functions that are nonpositive on the positive orthant and nonnegative elsewhere, we can extend an idea by **(author?)** (2) and define an operator $\Phi : \Re^n \to \Re^n$ component-wise as follows:

$$
\Phi_i(x) := \begin{cases}
\phi_1(x_i - \ell_i, F_i(x)) & \text{if } i \in I_\ell, \\
-\phi_1(u_i - x_i, -F_i(x)) & \text{if } i \in I_u, \\
\phi_2(x_i - \ell_i, \phi_1(u_i - x_i, -F_i(x))) & \text{if } i \in I_{\ell u}, \\
-F_i(x) & \text{if } i \in I_f.
\end{cases}
$$

Then we have that

$$
x^* \text{ solves } \mathrm{MCP}(F, \ell, u) \iff \Phi(x^*) = 0.
$$

Note that Φ is not differentiable in general. However, under suitable conditions, Φ is a semismooth function. A standard technique to solve the mixed complementarity problem is then to apply the semismooth Newton method outlined in the previous section to the system $\Phi(x) = 0$ and globalize it by considering the minimization problem

$$
\min \Psi(x),
$$

where Ψ is the natural merit function associated with the system $\Phi(x) = 0$:

$$
\Psi(x) := \frac{1}{2}\Phi(x)^T\Phi(x) = \frac{1}{2}\|\Phi(x)\|^2.
$$

Assuming that Ψ is continuously differentiable, we can follow the pattern from **(author?)** (6) and write down the basic semismooth solver for complementarity problems.

Algorithm 2.5 *(Basic Semismooth Method)*

(S.0) (Initialization)
Choose $x^0 \in \Re^n, \rho > 0, \beta \in (0,1), \sigma \in (0, 1/2), p > 2,$ *and set* $k := 0$.

(S.1) (Stopping Criterion)
If x^k *satisfies a suitable termination criterion: STOP.*

(S.2) (Search Direction Calculation)

Select an element $H_k \in \partial_B \Phi(x^k)$. *Find a solution* $d^k \in \Re^n$ *of the linear system*

$$H_k d = -\Phi(x^k). \tag{5}$$

If this system is not solvable or if the descent condition

$$\nabla \Psi(x^k)^T d^k \leq -\rho \|d^k\|^p \tag{6}$$

is not satisfied, set $d^k := -\nabla \Psi(x^k)$.

(S.3) (Line Search)

Compute $\bar{\ell}$ *as the smallest* ℓ *in* $\{0, 1, 2, \ldots\}$ *such that*

$$\Psi(x^k + \beta^{\bar{\ell}} d^k) \leq \Psi(x^k) + \sigma \beta^{\bar{\ell}} \nabla \Psi(x^k)^T d^k,$$

and set $t_k := \beta^{\bar{\ell}}$.

(S.4) (Update)

Set $x^{k+1} := x^k + t_k d^k, k \leftarrow k+1$, *and go to (S.1).*

This algorithm looks very much like a simple globalization scheme for the solution of a smooth system of equations by Newton's method, the only difference being that we only assume that Φ is semismooth, and that the matrix H_k in (5) is an element of the generalized Jacobian instead of the (potentially unavailable) Jacobian. The key assumption we had to make in order to achieve this simplicity is the continuous differentiability of Ψ. In view of the nonsmoothness of Φ, this would appear to be a strong requirement, but Proposition 2.6 establishes the continuous differentiability of Ψ for the cases of interest.

Algorithm 2.5 actually represents a whole class of methods since it depends heavily on the definition of Φ that, in turn, is completely determined by the choice of the two NCP-functions ϕ_1 and ϕ_2. Usually ϕ_1 plays the central role in the definition of Φ. For example, if there is no variable with finite lower and upper bounds (as is the case for the standard nonlinear complementarity problem), then ϕ_2 is not used in the definition of Φ.

For the purpose of this paper, we are particularly interested in the following two choices of Φ. We define

$$\Phi_{FB} := \Phi \quad \text{if} \quad \phi_1 = \phi_2 = \phi_{FB},$$

and

$$\Phi_{CCK} := \Phi \quad \text{if} \quad \phi_1 = \phi_{CCK},\ \phi_2 = \phi_{FB}.$$

In the latter case, the reason for not using $\phi_2 = \phi_{CCK}$ is technical and is related to simplifying an overestimate of the generalized Jacobian $\partial\Phi(x)$.

For the standard nonlinear complementarity problem, the operator Φ_{CCK} has stronger properties than Φ_{FB}, both from a theoretical and a numerical point of view (4). Hence Φ_{CCK} will be used by default in our implementation of Algorithm 2.5. However, in some situations, it is also helpful to have some alternative operators like Φ_{FB}. For example, our implementation uses Φ_{FB} to perform restarts.

We now summarize some of the properties of Φ_{FB} and Φ_{CCK} as well as of their corresponding merit functions

$$\Psi_{FB}(x) := \frac{1}{2}\Phi_{FB}(x)^T\Phi_{FB}(x) \quad \text{and} \quad \Psi_{CCK}(x) := \frac{1}{2}\Phi_{CCK}(x)^T\Phi_{CCK}(x).$$

The proofs of the results can be found in **(author?)** (10) for the case of $\Phi = \Phi_{FB}$. Since the proofs for $\Phi = \Phi_{CCK}$ are very similar (although quite technical and lengthy), they are omitted here.

Proposition 2.6 *Let $F : \Re^n \to \Re^n$ be continuously differentiable, Φ belong to $\{\Phi_{FB}, \Phi_{CCK}\}$, and Ψ be the corresponding merit function. Then the following hold:*

1. *Φ is semismooth and if in addition F' is locally Lipschitzian, then Φ is strongly semismooth.*
2. *Ψ is continuously differentiable on $\Re^n$.*
3. *If x^* is a strongly regular solution of MCP, then x^* is a BD-regular solution of $\Phi(x) = 0$.*

The previous result allows us to state convergence properties of Algorithm 2.5; the proof is analogous to those given in **(author?)** (6) for Φ_{FB} and the standard nonlinear complementarity problem.

Theorem 2.7 *Suppose that $F : \Re^n \to \Re^n$ is continuously differentiable, $\Phi \in \{\Phi_{FB}, \Phi_{CCK}\}$, and that $\{x^k\}$ is a sequence generated by Algorithm 2.5. Then any accumulation point of this sequence is a stationary point of Ψ. Moreover, if one of these accumulation points, say x^*, is a BD-regular solution of $\Phi(x) = 0$, then the following statements hold:*

(a) The search direction d^k is eventually given by the Newton equation (5) and the full stepsize $t_k = 1$ is eventually accepted in Step (S.3).

(b) The entire sequence $\{x^k\}$ converges to x^*.

(c) The rate of convergence is Q-superlinear.

(d) If in addition F' is locally Lipschitzian, then the rate of convergence is Q-quadratic.

3 The Linear System

The key advantage of the semismooth algorithm over MILES and PATH (9) is that the former only solves a single linear system per iteration while the latter algorithms use pivotal based codes to solve linear complementarity problems. The pivotal based codes rely upon the availability of a direct factorization and (sparse) rank-1 updates that limit their applicability to medium sized or large, structured problems. The semismooth algorithm spends a majority of the total time solving the Newton systems, $H_k d = -\Phi(x^k)$. The semismooth method does not require a direct factorization and rank-1 updates and can therefore benefit from the use of iterative methods. Effective mechanisms for solving this system using either iterative methods or a (sparse) direct method are indispensable and have great impact upon the success of the algorithm.

We begin by discussing the issues involved and options available when using direct methods to calculate the Newton direction. The main difficulty encountered is singularity in the Newton system. Information on detecting singularity and using that knowledge to construct a useful direction even in this case is presented. The effects of the techniques considered on the singular models in the MCPLIB test set are given and our choice of strategy for solving the linear system with direct methods is provided.

We then switch to an investigation of iterative methods for finding the Newton direction as these will enable the algorithm to solve very large problems. Unfortunately, in general H_k is neither symmetric nor positive definite; this restricts the choice of iterative method. One of the keys to achieving good performance from an iterative method is to select an appropriate preconditioner. The strategy implemented uses an incomplete LU factorization as a preconditioner. We present three of the iterative methods considered and evaluate their performance on the subset of the MCPLIB test set where the calculated preconditioner is not a complete LU factorization.

3.1 Direct Methods

When using a direct method to solve the Newton system, the software needs to have routines to factor and solve, and should be able to detect singularity problems. For reasonably sized problems we use the LUSOL (18) sparse factorization routines contained in the MINOS (26) nonlinear programming solver to factor H_k and solve for the Newton direction. The authors of this package have investigated the effects of modifying tolerances in the factorization on general linear systems and have suggested defaults that we have adopted for all our results.

The major difficulty with the direction finding problem is dealing with those instances where the Newton system either does not have a solution or has an infinite number of solutions. These singularity problems frequently occur in real world applications. While testing our semismooth code on the MCPLIB problem set, LUSOL reported that H_k was singular for 1929 of the 16525 attempted factorizations with at least one singular system encountered in 27 of the 88 models. However, the theoretical algorithm only provides a crude mechanism in this case, i.e. the use of a gradient step, while other approaches may be more effective. Clearly, any practical implementation of the semismooth algorithm must include appropriate procedures to deal with singularity.

We first investigate the applicability of scaling in conjunction with direct methods to avoid ill-conditioned systems. We then look at techniques to determine a useful direction when the model is singular, including using gradient steps, diagonal perturbations of H_k, and finding a least squares solution to the linear system. Empirical evidence is provided upon which we evaluate each of the methods.

3.1.1 Scaling

LUSOL can detect when a matrix is singular or nearly singular. In this subsection, we study the effects of scaling the linear problems in an effort to improve the conditioning of the matrices that we request to factor. Our goal is to see if we can reduce the number of occurrences where the factorization package determines that the matrix is singular. By using scaling we hope to improve the overall reliability of the code on ill-conditioned problems.

Two different scaling schemes were tested on the problems in the MCPLIB test set along with the default of no scaling. The first technique is the diagonal scaling used in the PATH solver. In this case, we define a row scaling by looking at elements of the diagonal of H_k that are large. Formally, we define a diagonal matrix R such that if $|(H_k)_{i,i}| > 100$ then

Table 1: The effects of scaling on the linear system solution

Scaling	Singular matrices detected	Failures	Time (secs)
none	1929	29	15,891
diagonal	1899	29	16,403
matrix	1371	31	15,240

$R_{i,i} = \frac{10}{|(H_k)_{i,i}|}$ and $R_{i,i} = 1$ otherwise. We then try to factor the scaled matrix RH_k.

Alternatively, we define a matrix scaling using the following diagonal matrices $R, C \in \Re^{n \times n}$ with diagonal entries:

$$R_{i,i} = \frac{1}{\max\left\{\sqrt{\Phi_i(x^k)^2 + \sum_j (H_k)_{i,j}^2}, 10^{-10}\right\}}, \tag{7}$$

$$C_{j,j} = \frac{1}{\max\left\{\sqrt{\sum_i (RH_k)_{i,j}^2}, 10^{-10}\right\}}. \tag{8}$$

We then solve the linear system $RH_kCC^{-1}d = -R\Phi(x^k)$ by defining $\tilde{d} := C^{-1}d$, solving the system $RH_kC\tilde{d} = -R\Phi(x^k)$, and recovering the Newton direction as $d = C\tilde{d}$. This procedure scales the rows and then the columns so that each has a two norm of 1. The constants in the max operator are used to avoid division by zero errors.

There are costs associated with scaling. Of the two methods, matrix scaling is more expensive per iteration because it requires looking at the data twice. We tested all of these scalings on the models in the MCPLIB test set and report in Table 1 the number of detected singular solves, failures of the algorithm to find a solution, and the accumulated total time in seconds spent in the code over the entire test set. When a singular model was detected we use the least squares recovery method detailed in Section 3.1.2. Since diagonal scaling does not cause a significant decrease in the number of singular systems encountered when compared to no scaling, we disregard this method. The reason for this poor behavior is probably due to the fact that the diagonal elements do not necessarily reflect the actual scaling of the problem and to the heavy scale dependence of the test on the magnitude of the diagonal elements. Matrix scaling significantly reduces the number of singular systems detected. However, it does result in additional failures of the algorithm. In the next section, we look at recovery techniques and report results for both no scaling and matrix scaling because of the differences in the number of singular matrices detected.

Before continuing, we note that the scaling investigated here is not very exhaustive and more complex schemes might be tested. Furthermore, the scaling is being performed on

Table 2: The overall effects of using gradient steps

Scaling	Failures	Time (secs)
none	44	6,732
matrix	41	4,978

the linear model, when it might be more appropriate to look at the nonlinear model to determine the scaling. Finally, we did not investigate modifying other parameters, such as those encountered in the nonlinear model in Section 4, in conjunction with scaling which might lead to improved reliability and performance.

3.1.2 Singularity

Having looked at scaling we need to establish procedures to recover from the singularity problem and generate a reasonable direction. We have investigated three techniques. The first is the theoretical standby of using only gradient steps when the Newton system is unsolvable. A second technique is to use a diagonal perturbation of H_k to regularize the problem. The final method is to calculate a least squares solution of the system at hand.

Gradient Steps An initial recovery technique, and the simplest of those considered, is to simply resort to a gradient step whenever a singular model is detected. This approach is theoretically justified, but in practice frequently leads to a stationary point that does not solve the complementarity problem. However, we use this approach as the baseline against which we evaluate the rest of the methods. The results are given in Table 2 where we report the number of times the algorithm failed and total time for both scaled and unscaled models. In this case, matrix scaling performs better than no scaling in both the number of failures and total time.

Perturbation Perturbation involves replacing the linear model with one that does not have a singularity problem. We investigated using a diagonal perturbation where we replace H_k with $H_k + \Delta I$ for some $\Delta > 0$. Δ was chosen in the interval $[10^{-8}, 1]$, with $\Delta = \frac{\Psi}{10}$ whenever possible. When the perturbation is insufficient to overcome the singularity, we increase Δ to $\delta\Delta$ for some $\delta > 1$. We currently use $\delta = 10$, and allow the perturbation to increase only one time per iteration.

Table 3: The overall effects of different perturbations

Scaling	Strategy	Failures	Time (secs)
none	decreasing	43	5,079
	on-demand	40	5,488
matrix	decreasing	37	4,352
	on-demand	38	4,854

The other choice to make is when to add the perturbation. There are two options investigated:

- When the first singular model is encountered, calculate a value for Δ and monotonically decrease it from one iteration to the next. The new value is $\min\{0.4\Delta, 0.1\Psi\}$. This perturbation scheme is used in the PATH code, and uses a "decreasing" perturbation from one iterate to the next.

- Every time a singular model is encountered, calculate a value for Δ. Essentially, this is an "on-demand" perturbation.

When scaling was used, we first perturbed the problem and then scaled it.

To test these strategies, we ran the MCPLIB test set using each of the strategies. We report the number of failures in the algorithm and total time in Table 3. When the perturbation fails to find a non-singular matrix, the least squares method (to be described in the next section) was used to calculate a direction. In general, the use of perturbation leads to fewer total failures than only using the gradient method. Scaling the problem leads to a decrease in both total time and number of failures.

Least Squares Method Finally, we investigate the use of the LSQR iterative scheme (27) to find a solution to the least squares problem $\min \left\| H_k d + \Phi(x^k) \right\|^2$ and use the resulting d as our Newton direction. The practical termination rules mentioned in 3.2.3 were used for these tests. As discussed in the section on iterative methods, the linear model passed to LSQR is scaled using the matrix scaling option and preconditioned with the incomplete LU factorization calculated as in Section 3.2.1.

We investigated using matrix scaling and no scaling in the linear model that we try to factor. We present the results on MCPLIB in Table 4 where we report the total number of failures in the algorithm and time. The major downside to using the iterative technique to

Table 4: The overall effects of using LSQR to combat singularities

Scaling	Failures	Time (secs)
none	29	15,891
matrix	31	15,240

solve the least squares problem is that it is fairly slow because we allow a larger number of iterations and have strict termination tolerances. We did not study the effect of changing the termination criteria. However, the results indicate that this method is better than both the gradient step and perturbation schemes in terms of reliability.

3.2 Iterative Techniques

Three iterative techniques for finding the Newton direction were investigated: LSQR, GMRES, and TFQMR. We recall that the systems of equations solved will generally be neither symmetric nor positive definite. Therefore, we cannot directly use the popular techniques from optimization algorithms such as conjugate gradients. The algorithms tested are a representative set of those meeting our requirements and are all directly coded in our implementation.

LSQR (27) is based upon the bidiagonalization method developed in **(author?)** (19) that implicitly solves the least squares problem, $\min \left\| H_k d + \Phi(x^k) \right\|^2$. The method is essentially a reliable variant of conjugate gradients applied to the normal equation, $H_k^T H_k d = -H_k^T \Phi(x^k)$.

The GMRES (32) method uses the Arnoldi procedure to construct an orthonormal basis for the Krylov subspace $\mathcal{K}_m \left(H_k, -\Phi(x^k) \right)$, where

$$\mathcal{K}_m(A, r) := \text{span} \left\{ r, Ar, \ldots, A^{m-1} r \right\}$$

for some matrix $A \in \Re^{n \times n}$ and a vector $r \in \Re^n$. We then use this basis to find a vector in the generated subspace minimizing the residual, $\left\| H_k d + \Phi(x^k) \right\|^2$. Our implementation uses Householder reflections for the orthogonalization process to preserve stability, maintains the current optimal value of the residual at each iteration using plane rotations, and restarts after m iterations. We remark that because our matrices are not guaranteed to be positive definite, restarted GMRES can stagnate and make no progress. The main difficulty with GMRES lies in choosing the restart frequency. If it is too small, we can fail to converge entirely, and if it is too large, the per iteration cost and storage requirements become significant.

The TFQMR (32) algorithm uses the Lanczos biorthogonalization algorithm to construct bases for the Krylov subspaces

$$\mathcal{K}_m\left(H_k, -\Phi(x^k)\right) \text{ and } \mathcal{K}_m\left(H_k^T, -\Phi(x^k)\right)$$

satisfying a biorthogonality condition. These bases are used to find a vector with approximate minimum residual in $\mathcal{K}_m\left(H_k, -\Phi(x^k)\right)$.

Scaling the linear system is crucial to the success of the iterative methods. Therefore, we will always apply the matrix scaling to H_k and use the resulting scaled matrix. Scaling significantly reduces the number of iterations required in all cases and thus the total time spent in the iterative method. The other key to achieving good performance from an iterative method is to select an appropriate preconditioner. The strategy discussed in the sequel calculates an incomplete LU factorization as a preconditioner.

3.2.1 Preconditioner

The preconditioner used in all of our tests of the iterative methods uses an incomplete LU factorization of H_k. We limit the number of nonzeros in the incomplete factorization to a small multiple of the number of nonzeros in $\nabla F(x)$. As the factorization progresses, elements with an absolute magnitude less than a drop tolerance are removed from the factorization. For large problems with sparse LU factorizations, this strategy amounts to using an exact preconditioner. We must also explicitly treat the cases where the incomplete LU factorization routine terminates because of "singularity". In this case, we add a diagonal perturbation to H_k for the purposes of constructing the preconditioner. The complete algorithm follows.

Algorithm 3.1 *(Preconditioner Calculation)*

(P.0) (Initialize)
Let $j = 0$, $\nu_{0,0} = 10^{-12}, \Delta_{0,0} = 0$.

(P.1) (Scale)
Calculate the diagonal matrices R *and* C *defined in (7) and (8) and use them to scale* H_k.

(P.2) (Factor)
Perform an incomplete factorization of $RH_kC + \Delta_{k,j}I$ *using* $\nu_{k,j}$ *as the drop tolerance.*

(P.3) (Success)

If incomplete factorization is successful, use the incomplete LU factorization as the preconditioner and go to P.8.

(P.4) (Unsuccessful)

If $j \geq 5$, use the identity matrix as the preconditioner and go to P.8.

(P.5) (Memory)

If the factorization routine terminated due to the memory restriction, increase the drop tolerance, $\nu_{k,j+1} = \sqrt{\nu_{k,j}}$, and set $\Delta_{k,j+1} = \Delta_{k,j}$.

(P.6) (Singular)

If the factorization routine terminated because the matrix was deemed singular, set $\nu_{k,j+1} = \nu_{k,j}$ and update the perturbation.

(P.6a) If $\Delta_{k,j} = 0$, then $\Delta_{k,j+1} = 1$.

(P.6b) Otherwise, $\Delta_{k,j+1} = \max\{10\Delta_{k,j}, \sqrt{\nu_{k,j}}\}$.

(P.7) (Update)

Let $j = j + 1$ and go to P.2.

(P.8) (Termination)

Let $\nu_{k+1,0} = \max\{\nu_{k+1,j}^2, 10^{-12}\}$. If $\Delta_{k,j} > 10^{-5}$ then $\Delta_{k+1,0} = \frac{\Delta_{k,j}}{10}$. Otherwise, $\Delta_{k+1,0} = 0$.

This preconditioning algorithm is guaranteed to terminate in a finite number of steps. In the unsuccessful case, no preconditioning will be used in the iterative method. Steps P.5 and P.6 are used to update the drop tolerance and perturbation respectively when the incomplete factorization fails. The square root is used in the drop tolerance update because it increases rapidly towards one. The initial value for the perturbation is chosen to be rather large in Step P.6a in an attempt to prevent $RH_kC + \Delta_{k,j}I$ from being poorly conditioned. In Step P.8, it is possible for small perturbations to be zeroed out, resulting in a large increase (to 1) at the next pass. This proved to be useful in some singular models.

Furthermore, we do not want to attempt a large number of incomplete factorizations every time we calculate a Newton direction. Therefore, we use the final values of the drop tolerance and perturbation from the current preconditioner calculation in the next attempt,

Table 5: Statistics of models used for iterative method testing

	bai_haung	bratu	obstacle	uruguay
Dimension	4,900	5,625	2,500	2,281
Number of nonzeros	29,120	33,750	15,000	90,206
Drop tolerance	1.7e-2	3.0e-4	3.0e-4	3.0e-4

as outlined in Step P.8. The drop tolerance and perturbation are slowly decreased in this step so that the complete LU factorization will eventually be used if possible.

We note that iterative methods converge much faster if a good preconditioner is used. In fact, these methods typically have a very good numerical behaviour for certain discretized partial differential equations where good preconditioners are available. Our experience with the MCPLIB test set indicates that the incomplete LU factorization chosen works well for the majority of the problems in this test set. Alternative preconditioners specific to a particular application can easily be added to the code.

3.2.2 Evaluation

Four problems in the MCPLIB test set were encountered where the preconditioner calculated with Algorithm 3.1 resulted in an incomplete LU factorization (i.e. $\nu > 10^{-12}$) and for which all the H_k encountered are known to be nonsingular. These four models, `bai_haung`, `bratu`, `obstacle`, and `uruguay`, are used to evaluate each of the iterative techniques. Table 5 provides the size of the problem, number of nonzeros, and drop tolerance used when calculating the preconditioner for each of these models. For all of these models, no perturbation was required.

To condense the results, we only show the performance of the iterative methods on the first linear system. The termination criteria for these tests was based upon the relative residual, $\frac{\|H_0 d_i + \Phi(x^0)\|}{\|\Phi(x^0)\|}$. The iterative methods terminated when the relative residual is less than 10^{-8}. In all cases, we chose an initial guess of $d_0 := 0$ since, at least locally, the next iterate generated by the semismooth method is close to the current iterate.

When using the GMRES method, we need to choose the restart frequency m (see description earlier in this section). We varied the value of m by choosing values between 10 and 50. The results for each value of m tested are given in the accompanying tables.

All of the trials were run on the same machine using the same executable so that we can make a valid comparison. Table 6 reports the results on the four test problems. The

Table 6: Iterative method results on problems using incomplete preconditioner

	bai_haung		bratu		obstacle		uruguay	
Method	Its	Res	Its	Res	Its	Res	Its	Res
LSQR	553	1.1e-6	31	3.4e-9	15	9.1e-10	21	8.1e-11
GMRES 10	70	1.1e-9	10	2.3e-9	8	1.5e-9	6	2.4e-9
GMRES 20	60	1.2e-10	13	6.9e-10	8	1.5e-9	6	2.4e-9
GMRES 50	36	3.6e-8	13	6.9e-10	8	1.5e-9	6	2.4e-9
TFQMR	49	2.7e-8	15	7.8e-10	9	1.1e-10	6	1.5e-6

total iterations (Its) and relative residual (Res) at the solution are given for the first linear system. The relative residual reported was calculated using the actual iterate generated by the method, as opposed to the updated residual vector.

The evidence on the nonsingular systems reported suggest that both GMRES and TFQMR are quite effective at solving the systems. LSQR also appears to be robust, but it may require a large number of iterations in order to converge. Note that results shown later demonstrate the robustness of these methods on the entire test suite. The robustness and effectiveness of LSQR is also in accordance with the results of **(author?)** (7).

3.2.3 Termination Rules

While the termination rule given above is reasonable for evaluation, we now return to the subject of practical termination rules. The relative residual calculated above is not applicable as a termination criterion unless we know a priori that the linear model has a solution. This is an unreasonable assumption to make because, as demonstrated in the direct methods section, some of the systems can be singular.

GMRES and TFQMR were terminated when the relative residual becomes less than 10^{-8} or the algorithm breaks down. While this is somewhat stringent, lesser values proved not to generate good directions. A break down of the method is detected when we encounter division by small constants. In addition, TFQMR is terminated when the difference between the iterates (i is the linear solver iteration counter), $\left\| d_i^k - d_{i+1}^k \right\|$, is less than 10^{-10} for 10 consecutive iterations.

The implementation of LSQR uses the termination rules developed in **(author?)** (27). They are to terminate if any of the following holds:

1. $\text{cond}(H_k) \geq \text{CONLIM}$

Table 7: Iterative method results on the MCPLIB test set

Method	Failures	Total time (secs)
LSQR	30	15,100
GMRES 10	33	11,053
GMRES 20	33	13,326
GMRES 50	32	11,964
GMRES 100	33	14,155
GMRES 200	34	16,734
TFQMR	31	20,872

2. $\|r_i\| \leq \text{BTOL} \left\|\Phi(x^k)\right\| + \text{ATOL} \|H_k\| \|d_i\|$

3. $\frac{\left\|H_k^T r_i\right\|}{\|H_k\|\|r_i\|} \leq \text{ATOL}$

where $r_i = -(H_k d_i + \Phi(x^k))$. Justification of these rules and a demonstration of their effectiveness is given in **(author?)** (27). We note that LSQR builds up estimates of $\|H_k\|$ and cond(H_k) by performing a small amount of additional computations per iteration of the code. The exact tolerances used are $ATOL = \epsilon^{\frac{2}{3}}$, $BTOL = \epsilon^{\frac{2}{3}}$, and $CONLIM = \frac{1}{10\sqrt{\epsilon}}$ where ϵ is the machine precision.

Furthermore, an iteration limit of $\min\{100000, 20n\}$ was used for all of the methods. The termination tolerances force us to find a point close to the exact solution of the linear system if it exists. We did not investigate using less stringent termination criteria.

The descent test in the semismooth algorithm description is used to verify that the direction provided by the iterative method is in fact a descent direction for the merit function. In cases where the descent test is not satisfied, we use the gradient direction.

Table 7 presents results over all of MCPLIB for each of the iterative methods implemented. This table reports the number of failures and total time used to solve the problems. The most reliable choice is LSQR on the MCPLIB test set, but all of the methods perform quite well.

3.3 Summary

The empirical results given above provides clear choices. For both large scale work and calculating a direction when the Newton system is singular we will use an iterative technique. We chose preconditioned LSQR as our default iterative method. GMRES or TFQMR can also be used by setting a run-time option. We remark that while this is the most reliable

choice, it is perhaps not the most efficient method. The effects of scaling the model we try to factor are indeterminant and we made the decision to use no scaling to achieve simplicity in the code. The effect of choices made in the nonlinear model that are discussed in the next section have a great impact upon the success of the algorithm. However, we did not investigate modifying those strategies in conjunction with the strategies in the linear solver.

4 The Nonlinear Model

At the nonlinear level of the algorithm, we are concerned with properties of the algorithm affecting convergence. These include numerical issues related to the merit function and calculation of H_k as well as crashing and the recourse taken when a stationary point of the merit function is encountered. These issues are discussed in the following subsections. We then summarize the results and present the final strategies chosen.

A difficulty with the semismooth code occurs when F is ill-defined because no guarantee is made that the iterates will remain feasible with respect to the box $[l, u]$. Such problems arise when using log functions or real powers that frequently occur in applications. Backtracking away from places where the function is undefined and restarting is typically sufficient for these models.

4.1 $\Phi(x^k)$ and H_k

As mentioned in Section 2, our implementation will use the penalized Fischer-Burmeister merit function. The value of λ chosen in (4) can have a significant impact upon the performance of the semismooth algorithm. We note that small values of λ, say less than 0.5, should not be used. In the case of a standard nonlinear complementarity problem (i.e., $l_i = 0$ and $u_i = +\infty$ for all $i = 1, \ldots, n$), emphasis would placed upon the $\max\{0, F_i(x^k)\} \max\{0, x_i^k\}$ term of the penalized Fischer-Burmeister function. This term is related to the complementarity error, but does not enforce $F_i(x^k) \geq 0$ and $x_i^k \geq 0$. If we were to solve the problem exactly, we would not be concerned. However, we use inexact arithmetic and terminate when the merit function is small, i.e. less than 10^{-12}. This criteria opens the possibility of finding a point satisfying the termination tolerance that is not close to a solution. The default choice in our implementation is to have $\lambda = 0.8$.

Furthermore, despite the fact that the penalized Fischer-Burmeister function is typically superior, there are some situations where the original function might be more appropriate.

Therefore, when using restarts (see Section 4.3.2) we also might change the merit function.

The calculation of $\phi(a, b)$ should be done carefully due to possible roundoff errors. To illustrate this point, assume that we have a machine with 6 decimal places of accuracy, and let $a = 10^{-4}$ and $b = 10^4$. Then a naive calculation of $\phi(a, b) = \sqrt{a^2 + b^2} - a - b$ would produce zero leading us to believe that we are at a solution to the problem when in fact we are not as the following calculation indicates:

$$\begin{aligned}
&\sqrt{10^{-8} + 10^8} - 10^{-4} - 10^4 \\
&= \sqrt{10^8} - 10^{-4} - 10^4 \\
&= 10^4 - 10^{-4} - 10^4 \\
&= 10^4 - 10^4 \\
&= 0.
\end{aligned}$$

The actual value of $\phi(a, b)$ should be on the order of -10^{-4}. We note that most machines have more than 6 decimal places of accuracy. However, some models in the MCPLIB test set were encountered where this type of roundoff error occurs.

The reason for the loss of precision has to do with the subtraction of two large positive numbers. We can avoid such subtractions in the calculation of the Fischer by using the following evaluation suggested by **(author?)** (33) as Exercise 8 in Chapter 21:

$$\phi(a, b) := \begin{cases} \sqrt{a^2 + b^2} - (a + b) & \text{if } a + b \leq 0 \\ \dfrac{-2ab}{\sqrt{a^2 + b^2} + (a + b)} & \text{otherwise.} \end{cases}$$

This expression is obtained by multiplying the original Fischer function definition by

$$\frac{\sqrt{a^2 + b^2} + (a + b)}{\sqrt{a^2 + b^2} + (a + b)}$$

and simplifying the result. Since there is no subtraction of positive quantities, the calculation will be accurate to machine precision every time. The square root operation needed is computed by defining $s = |a| + |b|$. If $s = 0$ then the value is zero, otherwise $s\sqrt{(\frac{a}{s})^2 + (\frac{b}{s})^2}$ is the value computed. This eliminates overflow problems.

When the Fischer-Burmeister function is in use, the calculation of H_k uses the procedure developed in **(author?)** (2, 6). When the penalized function is in use, a modification of the method in **(author?)** (4) is extended to MCP models for calculating H_k.

4.2 Crashing

Projected gradient crashing before starting the main algorithm can improve the performance of the algorithm by taking us to a more reasonable starting point. To do this, we use a technique already tested in **(author?)** (7) and add a new step, S.0a, to the algorithm between S.0 and S.1. Let $[\cdot]_B$ be the projection of $(\cdot)$ onto the box $B = [l, u]$. In this new step, we start with $j = 0$ and perform the following:

1. Calculate $d^j = -\nabla\Psi(x^j)$.

2. Let $\bar{\ell}$ be the smallest ℓ in $\{0, 1, 2, \ldots, \lfloor \frac{\log \tau}{\log \beta} \rfloor\}$ such that

$$\Psi([x^j + \beta^{\bar{\ell}} d^j]_B) \leq \Psi(x^j) - \sigma \nabla\Psi(x^j)^T (x^j - [x^j + \beta^{\bar{\ell}} d^j]_B)$$

   and set $t_j := \beta^{\bar{\ell}}$. If no such $\bar{\ell}$ exists, stop and set $x^0 = x^j$.

3. Otherwise let $x^{j+1} = [x^j + t_j d^j]_B$ and $j = j + 1$. Go to 1.

In the code $\tau = 10^{-5}$ and $\beta = 0.5$; furthermore, we only allow 10 iterations of the projected gradient crash method.

The crashing technique presented has iterates that remain feasible and improve upon the initial point with respect to the merit function. We believe that this is the key benefit from crashing – all iterates remain in B. Otherwise poor values of x^0 can frequently lead to failures in the semismooth algorithm. The crashing technique also gives us the opportunity to significantly affect the iterates generated during a restart.

4.3 Stationary Points

While stationary point termination is typically adequate for nonlinear optimization, determining a stationary point of the merit function that is not a zero is considered a "failure" by complementarity modelers. Much theoretical work has been carried out determining the weakest possible assumptions that can be made on the problem (and/or the algorithm) in order to guarantee that a stationary point of the merit function is in fact a solution of the complementarity problem. Some of these results restrict the problem class considered by employing convenient assumptions that cannot be easily verified for arbitrary models. Other techniques (such as nonmonotone linesearching) rely on a combination of heuristics and theory, while others are entirely heuristic in nature. The basic strategies we used to improve

the reliability of the semismooth solver include non-monotone linesearching and restarting. The positive effects of these strategies have been demonstrated in the literature and we just present the basic idea and any modifications made.

4.3.1 Non-monotone Linesearch

The first line of defense against convergence to stationary points is the use of a non-monotone linesearch (20; 21; 11). In this case we define a reference value, R^k and we use this value to replace the test in step S.3 of the algorithm with the non-monotone test:

$$\Psi(x^k + t_k d^k) \leq R^k + t_k \nabla\Psi(x^k)^T d^k.$$

Depending upon the choice of the reference value, this allows the merit function to increase from one iteration to the next. This strategy can not only improve convergence, but can also avoid local minimizers by allowing such increases.

In most cases, the reference value chosen by the code is the largest of the m best values of Ψ encountered so far. We begin by letting $\{M_1, \ldots, M_m\}$ be a finite set of values initialized to values $\kappa\Psi(x^0)$, where κ is used to determine the initial set of acceptable merit function values. The value of κ defaults to 1 in the code; $\kappa = 1$ indicates that we are not going to allow the merit function to increase beyond its initial value.

Having defined the values of $\{M_1, \ldots, M_m\}$ (where the code by default uses $m = 4$), we can now calculate a reference value. We must be careful when we allow gradient steps in the code. Assuming that d^k is the Newton direction (or a least squares solution to the Newton system in the presence of singularity, see 3.1.2), we define $i_0 = \text{argmax } M_i$ and $R^k = M_{i_0}$. After the nonmonotone linesearch rule above finds t_k, we update the memory so that $M_{i_0} = \Psi(x^k + t_k d^k)$, i.e. we remove an element from the memory having the largest merit function value.

When we decide to use a gradient step, it is beneficial to let $x^k = x^{\text{best}}$ where x^{best} is the iterate having the smallest value of Ψ. We then recalculate $d^k = -\nabla\Psi(x^k)$ using the best point and let $R^k = \Psi(x^k)$. That is to say that we force decrease from the best iterate found whenever a gradient step is performed. After a successful step we set $M_i = \Psi(x^k + t_k d^k)$ for all $i \in [1, \ldots, m]$. This prevents future iterates from returning to the same problem area.

Table 8: The performance of restart settings over the complete MCPLIB test set

Restart number	Failures
0 (first run)	67
1 (no crash)	63
2 (no crash, looser linesearch)	80
3 (no crash, ϕ_{FB}, strict linesearch)	89

4.3.2 Restarting

The rules for non-monotone linesearching and crashing are extremely useful in practice, but do not preclude convergence to a non-optimal stationary point. One observation relevant for complementarity solvers is that we know a priori the optimal value of the merit function at a solution if one exists. If the code detects that the current iterate is a stationary point that is not a solution, a recovery strategy can be invoked. One successful technique is the restart strategy of **(author?)** (13) where the recovery mechanism involves starting over from the user supplied starting point with a different set of parameters, thus leading to a different sequence of iterates being investigated. For the semismooth algorithm, the first restart turns the projected gradient crash method off. If no crash iterations were performed in the first attempt, the code will set $\lambda = 0.95$ to avoid wasting computational resources (by generating the same sequence of iterates as before). The second restart turns the projected gradient crash method off, sets $\lambda = 0.95$, and uses a less restrictive non-monotone linesearch criteria with $\kappa = 5$. The final restart uses ϕ_{FB} for in the definition of Φ, turns off the projected crash method, and uses $\kappa = 1$ in the non-monotone linesearch.

We caution that the restarts should be applicable to general models, otherwise they are not likely to be beneficial to the unseen problems encountered in the real world. That is, if we were to use the restart definition as the default, we should still solve most problems in the test set. We present in Table 8 the numbers of failures on the test set when restarts were used. Note in particular that the restarts (0, 1 and 2) that use the penalized Fischer-Burmeister function given in equation (4) outperform the one that uses the standard Fischer-Burmeister function defined in equation (3).

5 Numerical Results

Our implementation of the semismooth algorithm uses an enhanced version of the basic framework developed in **(author?)** (13) that helps to provide portability across platforms and interfaces to the algorithm from the AMPL (16) and GAMS (3) modeling languages, and the NEOS (12) and MATLAB (24) tools. The LUSOL (18) sparse factorization routines contained in the MINOS (26) nonlinear programming solver were used for factorization and preconditioning purposes. All codes were executed on the same 330 Mhz SUN Ultrasparc machine.

All of the linear algebra and other basic mechanisms are exactly the same between the semismooth algorithm and PATH. Therefore, the comparison made is as close to a true comparison of the algorithms as we can make. We note that the PATH code (version 4.4) is much more mature than the semismooth implementation. Both codes are continually being improved when deficiencies are uncovered.

To test the semismooth algorithm, we ran the code on all of the problems in the MCPLIB (8) suite of test problems; see Tables 9 and 10. The MCPLIB suite is being constantly updated, and several of the problems tested in this paper have been added recently to enhance the difficulty of the test suite. The time limits used in our testing were those provided in the problem description by the model developers and remain the same for all codes. The number of successes is reported first, followed by the number of failures in parenthesis. (Note that some test problems from the MCPLIB library have more than one starting point, e.g., problem `eppa` has 8 different starting points, while problem `electric` has only one starting point; this information is useful in order to understand the entries in Tables 9 and 10.) The "Direct" columns correspond to the semismooth algorithm that uses a direct factorization and only uses LSQR when H_k is reported singular.

In order to test the reliability of the iterative method, we ran all of the models in GAMSLIB and MCPLIB using only the iterative LSQR technique to calculate the Newton direction. For comparison purposes, we also show the performance of PATH 4.4 (10) on the same problems. These latter two results are given in the columns labeled "Iterative" and "PATH" respectively. In order to condense the information in the table, we have grouped several similar models together whenever this grouping results in no loss of information; for example, problems `colvdual` and `colvnlp` are grouped together as example `colv*`. We split the results into those that allow restarts and those that do not.

These results indicate that while the semismooth implementation is not quite as reliable as PATH, it does exceedingly well and is very robust. Furthermore, the results indicate that the iterative method is also robust. We note that the restart heuristic significantly improves the robustness of both semismooth and PATH.

We currently do not have any results on very large problems, but believe that based on the evidence, the code will scale well to the larger problems. In particular, the iterative version of the semismooth code requires significantly less memory than PATH, allowing the possibility of solving huge models. The current drawback of the semismooth code is the time taken by LSQR to solve the linear systems. We designed the code for robustness, and therefore chose parameters in the code to enhance reliability. This results in the PATH solver being much faster than the semismooth code - over the complete test suite, PATH took 2473 seconds, while the direct version of semismooth took 17650 seconds, of which 10117 seconds occurred in the failures, and the iterative only version took 14656 seconds, of which 8881 seconds occurred in the failures. Reducing this time, using for example some more aggressive algorithmic choices (see Table 3) is the subject of future research.

Acknowledgements

We are indebted to Michael Saunders for his advice regarding iterative solution techniques and Robert Vanderbei for his insightful comments on the numerical calculation of the Fischer function.

Todd Munson was partially supported by National Science Foundation Grants CCR-9619765 and CDA-9726385; and the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing, U.S. Department of Energy, under Contract W-31-109-Eng-38. Michael Ferris was partially supported by National Science Foundation Grants CCR-9619765 and CCR-9972372. Christian Kanzow was supported by the DFG (Deutsche Forschungsgemeinschaft).

References

[1] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*. SIAM, Philadelphia, PA, 1994.

Table 9: Comparative results

Problem	Without Restarts			With Restarts		
	Direct	Iterative	PATH	Direct	Iterative	PATH
asean9a, hanson	2(0)	2(0)	2(0)	2(0)	2(0)	2(0)
badfree, degen, qp	3(0)	3(0)	3(0)	3(0)	3(0)	3(0)
bai_haung	1(0)	1(0)	1(0)	1(0)	1(0)	1(0)
bert_oc	4(0)	4(0)	4(0)	4(0)	4(0)	4(0)
bertsekas, gafni	9(0)	9(0)	9(0)	9(0)	9(0)	9(0)
billups	0(3)	0(3)	0(3)	1(2)	0(3)	0(3)
bishop	0(1)	1(0)	1(0)	0(1)	1(0)	1(0)
bratu, obstacle	9(0)	9(0)	9(0)	9(0)	9(0)	9(0)
choi, nash	5(0)	5(0)	5(0)	5(0)	5(0)	5(0)
colv*	9(1)	9(1)	8(2)	10(0)	10(0)	10(0)
cycle, explcp	2(0)	2(0)	2(0)	2(0)	2(0)	2(0)
denmark	27(11)	26(12)	38(0)	38(0)	36(2)	38(0)
dirkse*	0(2)	0(2)	1(1)	0(2)	0(2)	1(1)
duopoly	0(1)	0(1)	0(1)	0(1)	0(1)	0(1)
eckstein	1(0)	1(0)	1(0)	1(0)	1(0)	1(0)
ehl_k*	12(0)	12(0)	10(2)	12(0)	12(0)	12(0)
electric	1(0)	1(0)	0(1)	1(0)	1(0)	1(0)
eppa	8(0)	8(0)	8(0)	8(0)	8(0)	8(0)
eta2100	1(0)	1(0)	1(0)	1(0)	1(0)	1(0)
force*	0(2)	0(2)	2(0)	0(2)	0(2)	2(0)
freebert	7(0)	7(0)	7(0)	7(0)	7(0)	7(0)
games	25(0)	25(0)	23(2)	25(0)	25(0)	25(0)
gei	2(0)	2(0)	2(0)	2(0)	2(0)	2(0)
golanmcp	0(1)	0(1)	1(0)	0(1)	0(1)	1(0)
hanskoop	8(2)	8(2)	10(0)	10(0)	10(0)	10(0)
hydroc*, methan08	1(2)	1(2)	3(0)	3(0)	3(0)	3(0)
jel, jmu	2(1)	2(1)	2(1)	2(1)	2(1)	3(0)
josephy, kojshin	16(0)	16(0)	16(0)	16(0)	16(0)	16(0)
keyzer	4(2)	4(2)	5(1)	5(1)	5(1)	6(0)
kyh*	0(4)	0(4)	2(2)	0(4)	0(4)	3(1)
lincont	1(0)	1(0)	1(0)	1(0)	1(0)	1(0)

Table 10: Comparative results (cont.)

Problem	Without Restarts			With Restarts		
	Direct	Iterative	PATH	Direct	Iterative	PATH
markusen	17(1)	17(1)	18(0)	18(0)	18(0)	18(0)
mathi*	13(0)	13(0)	13(0)	13(0)	13(0)	13(0)
mrtmge	1(0)	1(0)	1(0)	1(0)	1(0)	1(0)
multi-v*	0(3)	0(3)	3(0)	2(1)	3(0)	3(0)
ne-hard	0(1)	0(1)	1(0)	1(0)	1(0)	1(0)
olg	0(1)	0(1)	1(0)	1(0)	1(0)	1(0)
opt_cont*	5(0)	5(0)	5(0)	5(0)	5(0)	5(0)
pgvon*	3(9)	3(9)	10(2)	3(9)	3(9)	10(2)
pies	1(0)	1(0)	1(0)	1(0)	1(0)	1(0)
powell*	10(2)	10(2)	12(0)	12(0)	12(0)	12(0)
ralph	7(0)	7(0)	7(0)	7(0)	7(0)	7(0)
romer	2(0)	2(0)	2(0)	2(0)	2(0)	2(0)
scarf*	12(0)	12(0)	12(0)	12(0)	12(0)	12(0)
shubik	28(20)	36(12)	45(3)	45(3)	45(3)	48(0)
simple-*	1(1)	1(1)	1(1)	2(0)	2(0)	1(1)
sppe,tobin	7(0)	7(0)	7(0)	7(0)	7(0)	7(0)
tin*	129(3)	129(3)	123(9)	129(3)	129(3)	132(0)
trade12	2(0)	2(0)	2(0)	2(0)	2(0)	2(0)
trafelas	2(0)	1(1)	2(0)	2(0)	1(1)	2(0)
uruguay	4(0)	4(0)	4(0)	4(0)	4(0)	4(0)
xu*	35(0)	35(0)	35(0)	35(0)	35(0)	35(0)
Total	439(74)	446(67)	482(31)	482(31)	480(33)	504(9)

[2] S. C. Billups. *Algorithms for Complementarity Problems and Generalized Equations*. PhD thesis, University of Wisconsin–Madison, Madison, Wisconsin, August 1995.

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

[4] B. Chen, X. Chen, and C. Kanzow. A penalized Fischer-Burmeister NCP-function. *Mathematical Programming*, 88: 211–216, 2000.

[5] F. H. Clarke. *Optimization and Nonsmooth Analysis*. John Wiley & Sons, New York, 1983.

[6] T. De Luca, F. Facchinei, and C. Kanzow. A semismooth equation approach to the solution of nonlinear complementarity problems. *Mathematical Programming*, 75:407–439, 1996.

[7] T. De Luca, F. Facchinei, and C. Kanzow. A theoretical and numerical comparison of some semismooth algorithms for complementarity problems. *Computational Optimization and Applications*, 16: 173–205, 2000.

[8] S. P. Dirkse and M. C. Ferris. MCPLIB: A collection of nonlinear mixed complementarity problems. *Optimization Methods and Software*, 5:319–345, 1995.

[9] S. P. Dirkse and M. C. Ferris. The PATH solver: A non-monotone stabilization scheme for mixed complementarity problems. *Optimization Methods and Software*, 5:123–156, 1995.

[10] M. C. Ferris, C. Kanzow, and T. S. Munson. Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming*, 86:475–497, 2000.

[11] M. C. Ferris and S. Lucidi. Nonmonotone stabilization methods for nonlinear equations. *Journal of Optimization Theory and Applications*, 81:53–71, 1994.

[12] M. C. Ferris, M. P. Mesnier, and J. Moré. NEOS and condor: Solving nonlinear optimization problems over the Internet. Mathematical Programming Technical Report 96-08, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1998. Also available as ANL/MCS-P708-0398, Mathematics and Computer Science Division, Argonne National Laboratory.

[13] M. C. Ferris and T. S. Munson. Interfaces to PATH 3.0: Design, implementation and usage. *Computational Optimization and Applications*, 12:207–227, 1999.

[14] M. C. Ferris and J. S. Pang. Engineering and economic applications of complementarity problems. *SIAM Review*, 39:669–713, 1997.

[15] A. Fischer. A special Newton–type optimization method. *Optimization*, 24:269–284, 1992.

[16] R. Fourer, D.M. Gay, and B.W. Kernighan. *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press, 1993.

[17] R. W. Freund and N. M. Nachtigal. QMRPACK: A package of QMR algorithms. *ACM Transactions on Mathematical Software*, 22:46–77, 1996.

[18] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Maintaining LU factors of a general sparse matrix. *Linear Algebra and Its Applications*, 88/89:239–270, 1987.

[19] G. H. Golub and W. Kahan. Calculating the singular values and pseudoinverse of a matrix. *SIAM Journal on Numerical Analysis*, 2:205–224, 1965.

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

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

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

[23] L. Mathiesen. Computation of economic equilibria by a sequence of linear complementarity problems. *Mathematical Programming Study*, 23:144–162, 1985.

[24] MATLAB. *User's Guide*. The MathWorks, Inc., 1992.

[25] R. Mifflin. Semismooth and semiconvex functions in constrained optimization. *SIAM Journal on Control and Optimization*, 15:957–972, 1977.

[26] B. A. Murtagh and M. A. Saunders. MINOS 5.0 user's guide. Technical Report SOL 83.20, Stanford University, Stanford, California, 1983.

[27] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8:43–71, 1982.

[28] L. Qi. Convergence analysis of some algorithms for solving nonsmooth equations. *Mathematics of Operations Research*, 18:227–244, 1993.

[29] L. Qi and J. Sun. A nonsmooth version of Newton's method. *Mathematical Programming*, 58:353–368, 1993.

[30] D. Ralph. Global convergence of damped Newton's method for nonsmooth equations, via the path search. *Mathematics of Operations Research*, 19:352–389, 1994.

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

[32] Y. Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, Boston, Massachusetts, 1996.

[33] R. J. Vanderbei. *Linear Programming: Foundations and Extensions*. Kluwer Academic Publishers, Boston, Massachusetts, 1997.