The Semismooth Algorithm for Large Scale Complementarity Problems

Todd S. Munson*
Francisco Facchinei†
Michael C. Ferris*
Andreas Fischer‡
Christian Kanzow§

June 1999

Abstract

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 briefly discuss relevant theory associated with the algorithm and describe a sophisticated implementation in detail. Particular emphasis is given to robust methods for dealing with singularities in the linear system and to large scale issues. Results on the MCPLIB test suite indicate that the code is robust and has the potential to solve very large problems.

*Computer Sciences Department, University of Wisconsin — Madison, 1210 West Dayton Street, Madison, WI 53706, USA; e-mail: {tmunson,ferris}@cs.wisc.edu. The research of this author was partially supported by National Science Foundation Grants CCR-9619765 and CDA-9726385 and Air Force Office of Scientific Research Grant F49620-98-1-0417.

†Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Via Buonarroti 12, 00185 Roma, Italy; e-mail: soler@dis.uniroma1.it.

‡Department of Mathematics, University of Dortmund, 44221 Dortmund, Germany; e-mail: fischer@math.uni-dortmund.de.

§Institute of Applied Mathematics, University of Hamburg, Bundesstrasse 55, 20146 Hamburg, Germany; e-mail: kanzow@math.uni-hamburg.de. The research of this author was supported by the DFG (Deutsche Forschungsgemeinschaft).

1 Introduction

In operations research, complementary slackness arises frequently when considering linear programs; at optimality either the dual variable (multiplier) is zero or the primal slack variable is zero. However, this is just the tip of the iceberg. Not only are the optimality conditions of nonlinear programming a complementarity problem, but a whole host of problems from economics and engineering are naturally modeled in the complementarity framework [16, 17]. In order to make complementarity more accessible to general operations researchers, recent extensions to the AMPL and GAMS modeling languages have been proposed [10, 14], thereby making algorithms such as MILES and PATH accessible to general practitioners. Some of the successes of this approach are given in [25, 27, 38].

However, due to the success 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 condition 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 solve a vast majority of the standard suite of test models [8].

In the past few years there has been extensive theoretical research associated with the use of nonsmooth Newton methods for complementarity problems, with much emphasis on extending the domain of local convergence. Building on the success of early iterative linearization algorithms [26, 28], one approach is based on a piecewise linear approximation to the normal map [35], and resulted in the implementation of the PATH solver [9], currently the most widely used complementarity problem solver. While it may be argued that piecewise linear maps are more effective at approximating piecewise smooth maps, generating the "Newton" step typically involves the arduous task of solving a linear complementarity problem. A seemingly more attractive approach is to use an algorithm based on solving a 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. However, a serious effort to produce a sophisticated implementation has been lacking.

In this paper, we develop a code based upon the semismooth algorithm. We begin by briefly discussing the theoretical foundations of the semismooth algorithm. Many of the results contained in the section 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 used to overcome problems with singularity and ill-conditioning, and strategies to recover from finding non-optimal stationary points of the merit function. Issues related to the use of iterative solvers for large scale problems are outlined. We demonstrate the code's robustness on the problems in the MCPLIB [8] test collection by performing two tests, one using direct solution methods and another using only iterative techniques. Both indicate that the code is reliable and scalable.

We also compare the semismooth algorithm to PATH, showing comparable robustness, albeit using more computational time. These findings demonstrate that further investigation is warranted into the use of inexact Newton directions and general purpose preconditioners within the context of the semismooth algorithm. Since the underlying design of the algorithm requires only linear equation solutions, adapting much of the extensive literature on iterative solvers would undoubtedly be beneficial to the code.

2 Mathematical Foundation

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

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

The semismooth solver is based on a reformulation of the mixed complementarity problem as a nonlinear system of equations. In order to describe this formulation, we first recall that a mapping $\phi : \Re^2 \to \Re$ is called an *NCP-function* if it satisfies

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

Two examples of NCP-functions are the Fischer-Burmeister [18] function

$$\phi_{FB}(a, b) := \sqrt{a^2 + b^2} - a - b \tag{1}$$

and the penalized Fischer-Burmeister [4] function

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

where $\lambda \in (0, 1)$ is a given parameter. These two functions play an essential role in this paper, and we will come back to them in a moment.

We partition the index set $I = \{1, \ldots, n\}$ in the following way:

$$\begin{array}{rcl} I_l & := & \{i \in I \,|\, -\infty < l_i < u_i = +\infty\}, \\ I_u & := & \{i \in I \,|\, -\infty = l_i < u_i < +\infty\}, \\ I_{lu} & := & \{i \in I \,|\, -\infty < l_i < u_i < +\infty\}, \\ I_f & := & \{i \in I \,|\, -\infty = l_i < u_i = +\infty\}, \end{array}$$

i.e., I_l, I_u, I_{lu} 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, we extend an idea by Billups [1] and define an operator $\Phi : \Re^n \to \Re^n$ componentwise as follows:

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

It is easy to see that

$$x^* \text{ solves MCP} \iff x^* \text{ solves } \Phi(x) = 0.$$

Note that Φ is not in general differentiable. A standard technique to solve the mixed complementarity problem is to apply a nonsmooth Newton method (see [34, 33]) to the system $\Phi(x) = 0$ and globalize it using the corresponding merit function

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

Assuming that Ψ is continuously differentiable and recalling that the *B-subdifferential* of Φ at a point $x \in \Re^n$ is defined by [33]

$$\partial_B\Phi(x) := \{H \in \Re^{n\times n} \,|\, \exists\{x^k\} \subseteq D_\Phi : x^k \to x \text{ and } \Phi'(x^k) \to H\},$$

where D_Φ denotes the set of differentiable points of Φ, we can follow the pattern from [6] and write down the basic semismooth solver for MCP.

Algorithm 2.1 *(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*

$$H_k d = -\Phi(x^k). \tag{3}$$

If this system is not solvable or if the descent condition

$$\nabla\Psi(x^k)^T d^k \leq -\rho\|d^k\|^p \tag{4}$$

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

(S.3) *(Line Search)*
Compute $t_k := \max\{\beta^\ell \,|\, \ell = 0, 1, 2, \ldots\}$ *such that*

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

(S.4) (Update)
Set $x^{k+1} := x^k + t_k d^k, k \leftarrow k+1$, and go to (S.1).

Note that Algorithm 2.1 actually represents a whole class of methods since it depends heavily on the definition of Φ which, in turn, is completely determined by the choice of the two NCP-functions ϕ_1 and ϕ_2. Note that, usually, ϕ_1 plays the central role in the definition of Φ; for example, if there is no variable with both finite lower and upper bounds, then ϕ_2 is not used in the definition of Φ. In particular, this is the case for the standard nonlinear complementarity problem.

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

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

The reader may wonder why we do not take $\phi_2 = \phi_{CCK}$ as well; the simple reason is that we were unable to prove some of the subsequent results for this case. In fact, basically all of these results are based on a suitable overestimation for the generalized Jacobian $\partial\Phi(x)$. Typically, such an overestimation can be obtained by exploiting Theorem 2.3.9 in [5] that contains a convex hull operation. It is often possible to remove this convex hull and to get a simpler overestimate for $\partial\Phi(x)$. However, when using $\phi_1 = \phi_2 = \phi_{CCK}$, we were not able to remove the convex hull, so we decided not to take $\phi_2 = \phi_{CCK}$ in the definition of the operator Φ_{CCK}.

Both from a theoretical and a numerical point of view [4], the operator Φ_{CCK} has stronger properties than Φ_{FB}, at least for the standard nonlinear complementarity problem. Hence Φ_{CCK} will be used by default in our implementation of Algorithm 2.1. However, in some situations, it is also helpful to have 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 [11] for the case of $\Phi = \Phi_{FB}$. Since the proofs for $\Phi = \Phi_{CCK}$ are very similar (although quite technical and lengthy), we skip the proofs of all these results here.

Proposition 2.2 *Let Φ belong to $\{\Phi_{FB}, \Phi_{CCK}\}$ and Ψ be the corresponding merit function. Then the following hold:*

1. *Φ is semismooth.*
2. *If F' is locally Lipschitzian, then Φ is strongly semismooth.*

3. *Ψ is continuously differentiable on $\Re^n$.*

4. *If x^* be a strongly regular solution of MCP, then x^* is a BD-regular solution of $\Phi(x) = 0$.*

For a precise definition of (strong) semismoothness we refer the reader to [30, 34, 33]. Here, we only note that (strong) semismoothness is one of the two ingredients which are needed to prove local (quadratic) super-linear convergence of a nonsmooth Newton method. The second ingredient is the so-called BD-regularity assumption. A solution x^* of $\Phi(x) = 0$ is called *BD-regular* if all matrices $H \in \partial_B \Phi(x)$ are nonsingular. A strongly regular solution of MCP is defined in the sense of Robinson [36]; see also [11] for additional details.

The previous result allows us to state the following convergence properties of Algorithm 2.1. The proof is analogous to those given in [6] for Φ_{FB} and the standard nonlinear complementarity problem.

Theorem 2.3 *Let $\Phi \in \{\Phi_{FB}, \Phi_{CCK}\}$ and $\{x^k\}$ be a sequence generated by Algorithm 2.1. 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 entire sequence $\{x^k\}$ converges to x^*.

(b) The search direction d^k is eventually given by the Newton equation (3).

(c) The full stepsize $t_k = 1$ is eventually accepted in Step (S.3).

(d) The rate of convergence is Q-super-linear.

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

3 The Linear System

The heart of the semismooth algorithm lies in the linear algebra. Most of the time is spent solving the Newton system, $H_k d = -\Phi(x^k)$, where in general H_k is neither symmetric nor positive definite. Effective mechanisms for solving this system using either iterative techniques or a direct method are indispensable and have great impact upon the success of the algorithm. The key advantage of the semismooth algorithm over PATH [9] is that the former only solves a single linear system per iteration while the latter uses a pivotal based code to solve a linear complementarity problem. The pivotal based code relies upon the availability of a direct factorization and rank-1 updates which limits its applicability to medium sized or large, structured problems. Semismooth has no such restriction.

We begin our analysis by investigating iterative techniques for finding the Newton direction as these will enable the algorithm to solve very large problems. We present three of the methods considered and discuss time and space requirements and scaling issues. The methods were evaluated by applying them to two reasonably large models representative of the test suite. From the results, we conclude that LSQR [32] is the most reliable. Practical termination criteria are also mentioned.

We then discuss the issues involved and options available when using direct methods. 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 test set are given and our final choice of strategies for solving the linear system is provided.

Wherever possible, we want to use the best available technique to determine the Newton direction. While LSQR is very reliable, typically the best technique in terms of time is to use a direct method to factor H_k. However, when the size of the factors grows too large, we want to resort to the iterative technique. Therefore, the rules implemented are designed so that large, structured problems with sparse decompositions will use the factorization software, while problems that are either very dense or have large factors will not. Currently, a restriction on the size of the decomposition of 12 million nonzeros is imposed. We no longer consider using a direct method if this condition is violated.

3.1 Iterative Techniques

Three iterative techniques for finding the Newton direction were investigated: LSQR, GMRES, and QMR. 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.

LSQR [32] is based upon the bidiagonalization method developed in [22] which 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 code only requires a workspace of 3 n-vectors in addition to the storage of H_k, d, and $\Phi(x^k)$. The cost per iteration consists of $4nnz + 12n$ floating point operations, where nnz is the number of nonzero elements in H_k.

The GMRES [37] method uses the Arnoldi procedure to construct an orthonormal basis for the Krylov subspace $\mathcal{K}_m \left(H_k, -\frac{\Phi(x^k)}{\|\Phi(x^k)\|} \right)$, 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 method has a workspace requirement of $(m+2)$ n- and 4 m-vectors and uses $2nnz + 4n(1+2i) - 4i^2$ operations per iteration where $i \in [1, m]$ is the iteration number. Furthermore, we use around $4mn$ operations at the end of each m iterations to generate the minimizing vector. 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 QMR [37] algorithm uses the Lanczos biorthogonalization algorithm to construct bases for the Krylov subspaces $\mathcal{K}_m\left(H_k, -\frac{\Phi(x^k)}{\|\Phi(x^k)\|}\right)$ and $\mathcal{K}_m\left(H_k^T, -\frac{\Phi(x^k)}{\|\Phi(x^k)\|}\right)$ satisfying a biorthogonality condition. The bases generated are then used to find a vector with approximate minimum residual in $\mathcal{K}_m\left(H_k, -\frac{\Phi(x^k)}{\|\Phi(x^k)\|}\right)$. The QMRPACK [20] code tested uses the coupled two-term recurrence variant of the look-ahead Lanczos algorithm. We allowed m look-ahead steps to be tried, which results in a workspace of $10m+1$ n- and $8m+18$ m-vectors. Typically, m is chosen to be small. The code uses at least $4nnz + 14n$ floating point operations per iteration and requires a backsolve at the end to determine the vector with approximate minimal residual.

Scaling the linear system is crucial to the success of the iterative algorithms. We define a matrix scaling using the following diagonal matrices $R, C \in \Re^{n \times n}$ with diagonal entries:

$$R_{i,i} = \frac{1}{\max\left\{\sqrt{\Phi_i(x^k)^2 + \sum_j (H_k)_{i,j}^2}, 10^{-10}\right\}}, \tag{5}$$

$$C_{j,j} = \frac{1}{\max\left\{\sqrt{\sum_i (RH_k)_{i,j}^2}, 10^{-10}\right\}}. \tag{6}$$

We 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. When a row or column has a two norm close to zero, the scaling has no effect. Scaling significantly reduces the number of iterations required in all cases and thus the total time spent in the iterative solver.

3.1.1 Evaluation

We selected two reasonably large models with known solutions from the test suite on which to evaluate the iterative techniques. The `uruguay` model has 2,281 rows/columns and 90,206 nonzeros and is interesting because a direct factorization incurs a large amount of fill-in. `opt_cont255` is a structured problem with 8,192 rows/columns and 147,200 nonzeros. For each complementarity model, we only solved the first linear system arising from (3). 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 $d_0 := 0$. We did not investigate other choices.

Method	Iterations	Time	Status	Relative Error
LSQR	908	43	Solved	1.0e-8
GMRES 10	1,766	47	Solved	1.0e-8
GMRES 20	1,346	46	Solved	1.0e-8
GMRES 50	799	45	Solved	9.9e-9
GMRES 100	481	45	Solved	9.7e-9
GMRES 200	419	54	Solved	9.9e-9
QMR	466	41	Solved	6.5e-9

Table 1: Iterative Method Results: `uruguay`

Method	Iterations	Time	Status	Relative Error
LSQR	2,848	90	Solved	9.9e-9
GMRES 10	100,000	2,114	Iteration Limit	8.7e-4
GMRES 20	100,000	3,194	Iteration Limit	2.6e-4
GMRES 50	100,000	6,217	Iteration Limit	6.0e-5
GMRES 100	100,000	11,498	Iteration Limit	1.1e-5
GMRES 200	100,000	22,136	Iteration Limit	5.4e-8
QMR	100,000	5,288	Iteration Limit	1.6e-7

Table 2: Iterative Method Results: `opt_cont255`

When using the GMRES method, we need to choose the restart frequency. We varied the value of m by choosing values between 10 and 200. The results for each value of m tested are given in the accompanying tables. For QMR, we need to determine the number of look-ahead steps allowed. We placed an upper limit of 20 on these steps, although smaller values would suffice.

All of the trials were run on the same machine using the same executable so that we can make a valid comparison. Tables 1 and 2 report the results on the two test problems. The total iterations and time, exit status, and relative residual at the final point are given. Based upon the available information, we conclude that LSQR is the best choice for our purpose. This method is quite effective for the models we have generated. However, it may require a large number of iterations in order to converge. In these situations we are willing to sacrifice speed in exchange for reliability. Note that results shown later demonstrate the robustness of LSQR on the entire test suite. The robustness and effectiveness of LSQR is also in accordance with the results of [7].

3.1.2 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. Therefore, our implementation of LSQR uses the termination rules developed in [32]. They are to terminate if any of the following holds:

1. $\mathrm{cond}(H_k) \geq \mathrm{CONLIM}$
2. $\|r_i\| \leq \mathrm{BTOL}\left\|\Phi(x^k)\right\| + \mathrm{ATOL}\,\|H_k\|\,\|d_i\|$
3. $\frac{\left\|H_k^T r_i\right\|}{\|H_k\|\|r_i\|} \leq \mathrm{ATOL}$

where $r_i = -(H_k d_i + \Phi(x^k))$. Justification of these rules and a demonstration of their effectiveness is given in [32]. We note that LSQR builds up estimates of $\|H_k\|$ and $\mathrm{cond}(H_k)$ by performing a small amount of additional computation 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. These 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.

3.2 Direct Methods

We now turn our attention towards the issues involved in using direct methods to solve the Newton system. The factorization software needs to have routines to factor and solve, and should be able to uncover singularity problems and make a good approximation of the linearly dependent rows/columns in such cases. For reasonably sized problems we use the LUSOL [21] sparse factorization routines contained in the MINOS [31] 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 which we have adopted for all our results.

The major difficulty with the direction finding problem is dealing with those instances where the Newton system does not have a solution. These singularity problems frequently occur in real world applications. 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.

Following the success of scaling for the iterative techniques, 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 least squares solutions to the linear system. Empirical evidence is provided upon which we evaluate the methods.

We note here that the only requirements for the direct method are factor and solve routines and some way to determine singularity. The structure of the matrix to be factored does not change among iterations. Factorization routines other than LUSOL might be able to use this information to perform the factor/solve operations faster. However, we did not investigate this possibility.

Scaling	Detected Singular Matrices	Failures	Time
None	2138	28	13,922
Diagonal	2248	30	12,780
Matrix	1331	30	12,901

Table 3: Scaling Effects on Direct Methods

3.2.1 Scaling

LUSOL contains routines to detect when a matrix is singular or nearly singular. We study in this subsection 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 significantly 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 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 which are large and scale the entire row of the problem. Formally, we define a diagonal matrix R such that if $|(H_k)_{i,i}| > 100$ then $R_{i,i} = \frac{10}{|(H_k)_{i,i}|}$ and $R_{i,i} = 1$ otherwise. We then try to factorize the scaled matrix RH_k.

The other scaling method is the matrix scaling as used in the iterative techniques. We use diagonal matrices defined in (5) and (6) and attempt to factorize RH_kC.

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 entire test set and report in Table 3 the number of detected singular solves, failures of the algorithm to find a solution, and total time in seconds over the entire test set. When a singular model was detected we use the least squares recovery method detailed in Section 3.2.2. Since diagonal scaling does not improve significantly over 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. Matrix scaling significantly reduces the number of singular systems detected. However, it does result in additional failures of the algorithm. Since we were unable to definitively choose between no scaling and matrix scaling, in the next section we look at recovery techniques and report results for both.

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 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

Scaling	Failures	Time
none	28	4,836
matrix	24	3,630

Table 4: Gradient Results on 194 Singular Models

reliability and performance.

3.2.2 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 use LSQR to calculate a least squares solution of the system. All of the results in this section are only given for those models where singularity was detected by the linear solver. Only a few options were changed for each run, with the rest being held constant.

Gradient Steps The naive 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 4 where we report the number of times the algorithm failed and total time for both scaled and unscaled models. In this case, we note that matrix scaling performs better than no scaling.

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

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

- If the first model encountered is singular, calculate a λ and monotonically decrease it from one iteration to the next. The new value is $\min\{\kappa_1\lambda, \kappa_2\Psi\}$ where $\kappa_1 = 0.4$ and $\kappa_2 = 0.1$ by default. This strategy is used in the PATH code.
- Every time a singular model is encountered, calculate a value for λ.

Scaling	Strategy	Failures	Time
none	first	21	8,910
	demand	21	3,285
matrix	first	20	8,210
	demand	23	2,703

Table 5: Perturbation Effects on 194 Singular Models

Scaling	Failures	Time
none	9	8,513
matrix	11	7,534

Table 6: LSQR Results on 194 Singular Models

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

To test these strategies, we ran the set of singular models using each of the options. We report the number of failures in the algorithm and total time in Table 5. When the perturbation fails to find a nonsingular matrix, the least squares method (to be described in the next section) was used to calculate a direction. The use of perturbation leads to fewer total failures than only using the gradient method. The effects of scaling the problem are mixed, with it leading to a decrease in total time, but sometimes resulting in additional total failures.

Least Squares Method Finally, we investigate the use of the LSQR iterative scheme 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 3.1.2 were used.

We investigated using scaling and no scaling in the linear model that we try to factor. We present the results on the singular models in Table 6 where we report the total number of failures in the algorithm and time. The major downside to using the iterative technique to solve the least squares problem is that it is fairly slow because we allow many iterations and have low tolerances. We did not study the effect of changing the termination criteria. However, the results indicate that this method is better than the others tested in terms of reliability.

3.3 Summary

The empirical results given above provide clear choices. For both large scale work and calculating a direction when the Newton system is singular we will use the LSQR iterative technique. We remark that while this is the most reliable choice, it is perhaps not the most efficient method. While we always scale the linear system when an iterative solver is used, the effects of scaling the matrix 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 which 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 which 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 (2) 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 NCP, emphasis would be 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 which is not close to a solution. The default choice in our implementation is to have $\lambda = 0.8$ which can be changed using the `chen_lambda` option.

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. This is done by modifying the `merit_function` option to `fischer` or `chen` for the standard and penalized Fischer-Burmeister functions.

We also note that when $\phi_1 \neq \phi_2$, there are two different representations for Φ that lead to different performance. This situation only occurs when both variables are bounded (which is the only case when ϕ_2 is used). In this case, we use

$$\phi_2(x_i - l_i, \phi_1(u_i - x_i, -F_i(x))).$$

A symmetric alternative is to use

$$-\phi_2(u_i - x_i, \phi_1(x_i - l_i, F_i(x))).$$

Both of these functions are equally valid and can lead to different sequences of iterates being evaluated. We did not investigate this option further.

The calculation of $\phi(a,b)$ needs to be performed in such a way as to minimize the effect of roundoff error. If we have $a = 10^{-4}$ and $b = 10^4$ and are using a machine with 6 decimal places of accuracy, 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}. A better way to calculate $\phi(a,b)$ is as follows:

1. If $|a| > |b|$ then $\phi(a,b) = (\sqrt{a^2+b^2} - a) - b$.
2. Otherwise $\phi(a,b) = (\sqrt{a^2+b^2} - b) - a$.

This method gives a more accurate value of ϕ. 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.

The calculation of H_k uses the procedure developed in [1, 6] for the Fischer-Burmeister function, and for the penalized function, a modification of the method in [4] extended for MCP models.

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 [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 $t_j := \max\{\beta^\ell \mid \ell = 0, 1, 2, \ldots\}$ such that

$$\Psi([x^j + t_j d^j]_B) \le \Psi(x^j) - \tau\nabla\Psi(x^j)^T(x^j - [x^j + t_j d^j]_B).$$

3. If $t_j < \tau$ stop and set $x^0 = x^j$.

4. 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. Crashing can be turned off with the option `crash_method none`.

4.3 Stationary Points

While stationary point termination is typically adequate for nonlinear optimization, determining a stationary point of the residual 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 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 non-monotone 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 [23, 24, 12]. 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.

We now need to detail our choice of the reference value. We begin by letting $\{M_1, \ldots, M_m\}$ be a finite set of values initialized to $\kappa\Psi(x^0)$, where κ is used to determine the initial set of acceptable merit function values. The value of κ defaults to 1 in the code and can be modified with the `nms_initial_reference_factor` option; $\kappa = 1$ 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 3.2.2), we define $i_0 = \text{argmax } M_i$ and $R^k = M_{i_0}$. After the non-monotone 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 point with the absolute best merit function value encountered so far. 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.

A watchdog strategy [3] is also available for use in the code. The method employed allows steps to be accepted when they are "close" to the current iterate. Non-monotonic decrease is enforced every m iterations, where m is set by the `nms_mstep_frequency` option. Currently, we use this strategy only in a restart.

4.3.2 Restarting

The rules for non-monotone linesearching and crashing are extremely useful in practice, but do not preclude convergence to a nonoptimal 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 [15] where the recovery mechanism involves starting over from the user supplied starting point with a different set of options, thus leading to a different sequence of iterates being investigated. For the semismooth algorithm we use the restarts defined in Table 7. In addition, the first restart of the semismooth code will also include the option `chen_lambda 0.95` if no crash iterations were performed in its first run; otherwise, we would waste computational resources by generating the exact same sequences of iterates as the first attempt.

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 the particular restart options were used. Note that the restarts (0, 1 and 2) using the penalized Fischer-Burmeister function given in equation (2) outperform the one using the standard Fischer-Burmeister function defined in equation (1).

Restart Number	Parameter Values
1	crash_method none
2	chen_lambda 0.95 crash_method none nms_mstep_frequency 4 nms_initial_reference_factor 5
3	crash_method none merit_function fischer nms_mstep_frequency 1 nms_initial_reference_factor 1

Table 7: Restart Definitions

	Failures	
Restart Number	GAMSLIB	MCPLIB
0 (first run)	3	63
1	3	63
2	3	64
3	12	76

Table 8: Restart Performance on 102 GAMSLIB and 533 MCPLIB Problems

5 Numerical Results

Our implementation of the semismooth algorithm uses an enhanced version of the basic framework developed in [15] which helps to provide portability across platforms and interfaces to the algorithm from the AMPL [19] and GAMS [2] modeling languages, and the NEOS [13] and MATLAB [29] tools. The LUSOL [21] sparse factorization routines contained in the MINOS [31] nonlinear programming solver were used for factorization purposes.

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 is much more mature than the semismooth implementation. Both codes are continually being improved when deficiencies are uncovered.

Finally, we remark that the factorization routine is a key component of the implementation. For the semismooth algorithm, we know the structure of the matrix to be factored and this structure does not change from iteration to iteration. Furthermore, rank-1 updates to the matrix are not required. These observations indicate that a factorization routine other than those provided by LUSOL might be more effective for the semismooth algorithm. However, we sacrifice some potential speed gains in preference for having a consistent choice of direct method among the codes tested.

To test the standard algorithm, we ran the code on all of the problems in the GAMSLIB and MCPLIB [8] suites of test problems. All of the models in GAMSLIB were solved and are not reported here. Also note that 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. In Tables 9 and 10 we present the results on the MCPLIB problems. The number of successes is reported first, followed by the number of failures in parenthesis. In order to test the reliability of the iterative method, we ran all of the models using only the iterative LSQR technique to calculate the Newton direction. For comparison purposes, we also show the performance of PATH 4.0 [11] on the same problems. These latter two results are given in the columns labeled "Iterative" and "PATH 4.0" 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 and requires much less memory resources. 42 of the additional failures in the "Iterative" column occur in the `asean9a`, `denmark`, and `opt_cont511` models in which the time limit was encountered. We also 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 2480 seconds, while the default version of semismooth took 13902 seconds and the iterative only version took 95491 seconds. When we only count the times where both solvers succeed, the results are much closer with PATH using 1823 seconds and semismooth 4332 seconds. However, as outlined in the introduction, we believe that many of the standard techniques for improving iterative linear equation solvers are applicable in the context of the semismooth algorithm. This is the topic of future research.

We have shown that a semismooth algorithm can be implemented as a very robust MCP solver. Particular care needs to be taken in implementing the evaluation of Φ and Ψ, and in treating numerical issues related to constructing the Newton step. The algorithm has great potential in the very large scale setting as indicated by our preliminary computations.

Problem	Without Restarts			With Restarts		
	Semi	Iterative	PATH 4.0	Semi	Iterative	PATH 4.0
asean9a, hanson	3(0)	2(1)	3(0)	3(0)	2(1)	3(0)
badfree, degen, qp	3(0)	3(0)	3(0)	3(0)	3(0)	3(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)	0(1)	1(0)	0(1)	0(1)	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)	10(0)	10(0)	10(0)	10(0)
cycle, explcp	2(0)	2(0)	2(0)	2(0)	2(0)	2(0)
denmark	30(10)	0(40)	40(0)	40(0)	0(40)	40(0)
dirkse*	0(2)	0(2)	1(1)	0(2)	0(2)	2(0)
duopoly	0(1)	0(1)	0(1)	0(1)	0(1)	1(0)
eckstein	1(0)	1(0)	1(0)	1(0)	1(0)	1(0)
ehl_k*	12(0)	12(0)	8(4)	12(0)	12(0)	12(0)
electric	1(0)	0(1)	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)	2(0)	2(0)	2(0)
freebert	7(0)	7(0)	7(0)	7(0)	7(0)	7(0)
games	25(0)	25(0)	20(5)	25(0)	25(0)	25(0)
gei	2(0)	2(0)	1(1)	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)	3(0)	2(1)	2(1)	3(0)
josephy, kojshin	16(0)	16(0)	16(0)	16(0)	16(0)	16(0)
keyzer	3(3)	3(3)	6(0)	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 9: Comparative Results

Problem	Without Restarts			With Restarts		
	Semi	Iterative	PATH 4.0	Semi	Iterative	PATH 4.0
markusen	31(1)	31(1)	32(0)	32(0)	32(0)	32(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)	2(1)	3(0)	3(0)	3(0)
ne-hard	0(1)	0(1)	1(0)	0(1)	0(1)	1(0)
olg	0(1)	0(1)	1(0)	1(0)	0(1)	1(0)
opt_cont*	5(0)	4(1)	5(0)	5(0)	4(1)	5(0)
pgvon*	3(9)	2(10)	4(8)	3(9)	4(8)	6(6)
pies	1(0)	0(1)	1(0)	1(0)	1(0)	1(0)
powell*	10(2)	10(2)	12(0)	12(0)	12(0)	12(0)
ralph	8(0)	8(0)	8(0)	8(0)	8(0)	8(0)
romer	2(0)	2(0)	1(1)	2(0)	2(0)	2(0)
scarf*	12(0)	12(0)	10(2)	12(0)	12(0)	12(0)
shubik	44(4)	43(5)	37(11)	48(0)	48(0)	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*	128(4)	128(4)	129(3)	128(4)	128(4)	132(0)
trade12	2(0)	2(0)	2(0)	2(0)	2(0)	2(0)
trafelas	1(1)	1(1)	2(0)	1(1)	1(1)	2(0)
uruguay	7(0)	7(0)	7(0)	7(0)	7(0)	7(0)
xu*	35(0)	35(0)	35(0)	35(0)	35(0)	35(0)
Total	473(60)	438(95)	488(45)	505(28)	462(71)	522(11)

Table 10: Comparative Results (cont.)

Acknowledgements

We are indebted to Michael Saunders for his advice and insight into the iterative solution techniques.

References

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

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

[3] R. M. Chamberlain, M. J. D. Powell, and C. Lemaréchal. The watchdog technique for forcing convergence in algorithms for constrained optimization. *Mathematical Programming Study*, 16:1–17, 1982.

[4] B. Chen, X. Chen, and C. Kanzow. A penalized Fischer-Burmeister NCP-function: Theoretical investigation and numerical results. Preprint 126, Institute of Applied Mathematics, University of Hamburg, Hamburg, Germany, 1997.

[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. Mathematical Programming Technical Report 97-15, Computer Sciences Department, University of Wisconsin, Madison, Wisconsin, 1997.

[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, R. Fourer, and D. M. Gay. Expressing complementarity problems and communicating them to solvers. *SIAM Journal on Optimization, forthcoming*, 1999.

[11] M. C. Ferris, C. Kanzow, and T. S. Munson. Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming, forthcoming*, 1999.

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

[13] 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.

[14] M. C. Ferris and T. S. Munson. Complementarity problems in GAMS and the PATH solver. *Journal of Economic Dynamics and Control, forthcoming*, 1999.

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

[16] M. C. Ferris and J. S. Pang, editors. *Complementarity and Variational Problems: State of the Art*, Philadelphia, Pennsylvania, 1997. SIAM Publications.

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

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

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

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

[21] 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.

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

[23] 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.

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

[25] G. W. Harrison, T. F. Rutherford, and D. Tarr. Quantifying the Uruguay round. *The Economic Journal*, 107:1405–1430, 1997.

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

[27] A. S. Manne and T. F. Rutherford. International trade in oil, gas and carbon emission rights: An intertemporal general equilibrium model. *The Energy Journal*, 14:1–20, 1993.

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

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

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

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

[32] 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.

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

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

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

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

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

[38] F. Tin-Loi and M. C. Ferris. Holonomic analysis of quasibrittle fracture with nonlinear softening. In B. L. Karihaloo, Y. W. Mai, M. I. Ripley, and R. O. Ritchie, editors, *Advances in Fracture Research*, volume 2, pages 2183–2190. Pergamon Press, 1997.