universität wien

DIPLOMARBEIT

Titel der Diplomarbeit

"COMPASS: A Free Solver for Mixed Complementarity Problems"

Verfasser

Stefan Schmelzer

angestrebter akademischer Grad

Magister der Naturwissenschaften (Mag.rer.nat)

Wien, April 2012

Studienkennzahl lt. Studienblatt:	A 405
Studienrichtung lt. Studienblatt:	Mathematik
Betreuer:	o. Univ.-Prof. Dr. Arnold Neumaier

Abstract

This thesis presents COMPASS, a globally convergent algorithm for solving the Mixed Complementarity Problem (MCP). The mathematical theory behind it is based on the PATH solver, the standard solver for complementarity problems. The fundament of COMPASS is a stabilized Newton method; the MCP is reformulated as the problem of finding a zero of a non-smooth vector valued function, the normal equation. A pivot technique is used to solve a general first order approximation of the normal equation at the current point in order to obtain a piecewise linear path connecting consecutive iterates. A general descent framework uses a smooth merit function establishing non-monotone descent criteria; a non-monotone stabilization scheme employs a watchdog technique and pathsearches reducing the number of function and gradient evaluations. An implementation in MATLAB/Octave was developed and is an integral part of this thesis. Simulation results on random problems, as well as a short course on economic models as an example of a field of application are included.

Contents

1 Introduction

The aim of this thesis is to present COMPASS, a globally convergent algorithm for solving the Mixed Complementarity Problem (MCP). An MCP is a certain kind of mathematical problem, that has become quite important in recent history, due to the discoveries that many different kinds of problems may be formulated as an MCP, and to the development of stable and efficient numerical solution procedures.

The term Complementarity Problem first arose in the 1960s, e.g. in Richard Cottle's Ph.D. thesis [9], together with the Variational Inequality, introduced by Philip Hartman and Guido Stampacchia [20]. A Detailed history on these problems, a theory on existence and uniqueness of solutions as well as a short overview of algorithms can be found in detail in Patrick T. Harker and Jong-Shi Pang's work [19] from 1990. However in 1994 Michael C. Ferris and Steven P. Dirkse presented their solver PATH [12], which was the first solution routine for which a strong global convergence theory existed. PATH was further developed since, see e.g. [16] and [15], and has become the standard solver for Complementarity Problems. Apart from PATH several other solvers for the MCP exist, e.g. MILES [33], a generalization of the classical Josephy Newton method for the NCP, SMOOTH [8], that makes use of smooth approximations of nonsmooth reformulations of the NCP, or PROXI [3], based on a reformulation of the MCP in terms of nonsmooth equations and a nonsmooth version of Newton's method, to name just a few. A comparison of the most competitive of them was carried out in the same testing environment in [5], in which PATH turned out to be the overall best in terms of speed and success, both among the large and small scale problems tested.

Part of the motivation for this thesis is of economic kind, due to the fact that nowadays, many economic models are formulated as an MCP.

In 1985 Mathiesen's finding [25] to formulate the economic equilibrium as an MCP gave rise to the development of different forms of computable economic models, and their solution routines. Until then economic models were of a more theoretical character; towards the end of the 19th century Leon Walras [36] lay the foundation of General Equilibrium (GE) theory, which was formalized in a model framework by

Arrow and Debreu in the 1950s [1]. These models were used in the 1950s and 60s and could typically be solved merely by hand, and give only aggregate insights of economic structures.

Applied General Equilibrium (AGE) models were developed later, see e.g. [22], and solved using linear solving procedures or fixed point theorems [28]. Computable General Equilibrium (CGE) models, are a result of combining economic mathematical theory and computational methods, where the general equilibrium theory is assumed, and real economic data is used in order to estimate among other indicators, sectoral and price level development. Since the 1980s, due to improved computational methods, these CGE models moved into the focus of economic science and completely replaced the GE and AGE approaches, see [28].

After the appearance and further development of PATH, see [12], [16] and [15], a considerably large community of economists and modellers started to develop many kinds of models in the MCP format, and to use PATH as the solution routine. GAMS, the General Algebraic Modelling Software, [6] was developed by a group around Thomas Rutherford in order to ease economic modelling, to provide an interface for formulating these models as certain mathematical problems, e.g. complementarity problems, and to establish a link to different solution procedures, as PATH. Among the users of the software packages GAMS/PATH are internationally renowned institutions as the World Bank, the International Monetary Fund, but also different groups at several Universities (National Technical University of Athens, University of Graz), and other economic research facilities (IHS Vienna).

COMPASS was developed to be a clone of the PATH solver in the course of this thesis. COMPASS is published under the GNU General Public License and hence is free software. The purpose of this thesis is to provide detailed understanding of the PATH solver, and to describe the implementation of COMPASS (the developed algorithm was implemented in MATLAB/Octave, see section 3.6), rather than to develop a solver that could be compared with PATH in terms of performance. However, details and techniques described in this thesis match those in the PATH solver except for some details that will be documented later.

The COMPASS solver is an implementation of a stabilized Newton method for solving the MCP. In order to apply a Newton-like method, it is necessary to transform the MCP into the equivalent problem of finding a zero of a function, namely the usually nonsmooth *normal map*, as proposed by Robinson [30]. The normal map would correspond to the gradient function when applying a standard Newton method to a (at least) continuously differentiable function from $\mathbb{R}^n$ into $\mathbb{R}$. Nonsmoothness of

the normal map is what causes most of the complexity of the COMPASS algorithm. Similar to linearization of the gradient function (or quadratic approximation of the objective function) in standard Newton-like techniques, a general first order approximation of the normal map is computed. Finding a zero of this approximation turns out to be equivalent to a *linear MCP*. The solution of this linear MCP is found with the help of a pivot technique, similar to the one described by Lemke [23]. This pivot algorithm provides not only a solution of the linear MCP, which is equal to the solution of the approximation of the normal map at the current point, but also a *piecewise linear path* connecting the current point and the solution. This path is the element in the COMPASS algorithm corresponding to the Newton direction p^k in a standard Newton method.

Whether the endpoint of this path, i.e. the solution to the approximation, is accepted as the next iterate, depends on the *non monotone stabilization (NMS)* - scheme [12]. If it is not accepted, the generated path constructed between the current point and the Newton point is searched for an acceptable point near the Newton point. This procedure is called *pathsearching*, and corresponds to the linesearch procedure in the smooth case.

The algorithm is ended in a general descent framework, see [15], which uses a smooth merit function, and allows that the descent criteria are not checked in every step, and if checked, are compared to a reference value, typically the maximum over a fixed number of previous merit function values, instead of to the last merit function value only. This relaxes the constraint of monotone descent from one iterate to the next, which improves efficiency and increases robustness of the algorithm, see [12] and [15].

The Algorithm terminates successfully when the zero of the merit function has been found; The zero of the normal map corresponds to a zero of the smooth merit function, as well as to a solution of the MCP.

The thesis is structured as follows. Chapter 2 gives definitions of the relevant problems, and provides the mathematical basics necessary for the implementation of COMPASS and its convergence theory, among them the concept of the merit function, a general pivot technique to solve a linear MCP and a crash technique. Chapter 3 gives a detailed description of the COMPASS algorithm, the global convergence result and an implementation of COMPASS in MATLAB/Octave. Documentation of personal experience during the implementation procedure, as well as numerical results of applications of COMPASS on different kinds of random problems are provided in Chapter 4. Chapter 5 gives a short historical overview of the development of

economic general equilibrium theory in combination with its formulation as a Mixed Complementarity Problem. A detailed example of how to implement such a model as an MCP is included. Chapter 6 concludes.

A word about notation is in order. Lower indices of vectors or matrices will denote their components, e.g. z_i will be the ith component of the vector $z = (z_i)_{i \in I}$, where I is the index set of the vector's components. Upper indices will be used to indicate a certain element in a sequence of vectors or Matrices, e.g. z^k will be the kth vector in a sequence or set of vectors $(z^k)_{k \in K}$, where K is index set of the sequence.

The entry of an $n \times m$ matrix M, $n \in I$, $m \in J$, in the ith row and jth column will be denoted by M_{ij}, the ith column will be denoted by $M_{i:}$ and the jth row as $M_{:j}$. The submatrix consisting of all those entries contained in the rows with indices in $K \subseteq I$ as well as in the columns with indices in $L \subseteq J$ will be denoted as M_{KL}.

Complementarity of two nonnegative vectors x and y will mean that their inner product $x^T y$ equals zero, and will be denoted as $x \perp y$ or $inf(x,y) = 0$.

In references to equations the abbreviations LHS and RHS are used to denote the left, or right hand side of the equation, respectively.

2 Numerical and Mathematical Basics

2.1 The Mixed Complementarity Problem

In this section, some definitions and basic concepts as well as minor results surrounding complementarity problems are presented. All of them are quite commonly used in the literature, more information can be taken from [3], [11], [19] and [24]. Among them are the *variational inequality*, the *linear* and *mixed complementarity problems*, as well as the relations between them.

2.1.1 Variational Inequality (VI)

Given a function $f : \mathbb{R}^n \to \mathbb{R}^n$ and a set $X \subseteq \mathbb{R}^n$, the *variational inequality*, $VI(f,X)$, is the problem of finding $z \in X$, such that

$$f(z)^T(x - z) \geq 0 \qquad \text{for all } x \in X. \tag{2.1.1}$$

Typically the set X is assumed to be closed and convex. The variational inequality arises in many optimization problems, usually f denotes the gradient of some real valued function F that should be minimized over the set X. In such a case, at an interior solution of VI(f,X), when $z \in \text{int}(X)$, the gradient has to be zero, since at a solution of VI(f,X) the gradient and any direction starting from the solution and pointing inside the set X, have to enclose an angle that is less than 90 degrees. Hence at a solution at the boundary of X, the steepest descent direction points away from the set X in a "perpendicular" way. In both cases, necessary first order optimality conditions are satisfied.

In this context it is useful to define the *normal cone* to a closed convex set C at a point $z \in C$ to be the set

$$N_C(z) = \{y \in \mathbb{R}^n | y^T(x - z) \leq 0 \quad \text{for all } x \in C\}. \tag{2.1.2}$$

It is the set of vectors that includes an angle greater than 90 degrees with any direction in C emanating from z.

Hence the variational inequality (2.1.1) can be reformulated as to find a $z \in X$ such that $f(z) \in N_X(z)$, for X closed and convex, which is a common kind of formulation in the literature.

2.1.2 Complementarity Problems

For a given function $f : \mathbb{R}^n \to \mathbb{R}^n$ the *nonlinear complementarity problem*, $NCP(f)$, is the problem of finding $z \in \mathbb{R}^n_+$, such that

$$f(z) \geq 0 \quad \text{and} \tag{2.1.3}$$
$$f(z)^T z = 0, \tag{2.1.4}$$

Given a Matrix $M \in \mathbb{R}^{n \times n}$ and a vector $q \in \mathbb{R}^n$, the *Linear Complementarity Problem*, $LCP(M,q)$, is the problem of finding vectors z and $w \in \mathbb{R}^n_+$, such that

$$Mz + q = w \quad \text{and} \tag{2.1.5}$$
$$z^T w = 0, \tag{2.1.6}$$

Sometimes complementarity conditions like (2.1.6) are equivalently written in the form $\inf(z,w) = 0$, or in the simplest manner $z \perp w$.

It is explicative to note that here M and q define an affine linear map f from $\mathbb{R}^n$ into itself, defined as $f(z) := Mz + q$, hence the LCP can be reformulated as the problem of finding $z \in \mathbb{R}^n_+$, such that

$$f(z) \in \mathbb{R}^n_+ \quad \text{and} \tag{2.1.7}$$
$$z \perp f(z), \tag{2.1.8}$$

which makes the fact obvious that it is equivalent to (2.1.3) and (2.1.4) when f is linear.

2.1.3 Mixed Complementarity Problem (MCP)

Given a function $f : \mathbb{R}^n \to \mathbb{R}^n$ and a box $B = [l,u]$, where $l,u \in \overline{\mathbb{R}}^n$, the *Mixed Complementarity Problem*, $MCP(f,B)$, is the problem of finding $z \in B$, such that for each index $i \in \{1,...,n\}$

$$\text{either} \quad l_i \leq z_i \leq u_i \quad \text{and} \quad f_i(z) = 0, \tag{2.1.9}$$
$$\text{or} \quad z_i = l_i \quad \text{and} \quad f_i(z) > 0, \tag{2.1.10}$$
$$\text{or} \quad z_i = u_i \quad \text{and} \quad f_i(z) < 0. \tag{2.1.11}$$

This relationship is sometimes denoted as $l \leq z \leq u \perp f(z)$ (see, e.g. [24]).

For the purposes of this work it will be of importance to recast the MCP in the following equivalent way:

Given f and B as above,

$$\text{find} \qquad z \in B, \quad w,v \in \mathbb{R}^n_+ \tag{2.1.12}$$

$$\text{s.t.} \qquad f(z) = w - v \tag{2.1.13}$$

$$(z - l)^T w = 0 \tag{2.1.14}$$

$$(u - z)^T v = 0. \tag{2.1.15}$$

It is not hard to see that the vectors w and v are actually only "dummy" vectors, which are defined to be the positive and negative parts of $f(z)$, respectively, at the solution z. Hence in the following a solution of the MCP will either refer to $z \in B$, or a triple (z,w,v), satisfying (2.1.12)-(2.1.15), as convenience dictates.

There is a strong connection between the VI and complementarity problems. The following Theorem makes the relationship clear, and is easily proved (see e.g. [11]).

Theorem 1. *Given $f : B \to \mathbb{R}^n$ and a box B:*
$z \in B$ solves MCP(f,B) <=> z solves VI(f,B)

Hence any complementarity problem (CP) is equivalent to a VI, but not the other way round (the set X in VI(f,X) can actually be arbitrary), and the CP class is a special case of the VI problem class.

For an extensive treatment of existence and uniqueness of solutions to these problem classes, the interested reader is referred to [19].

The COMPASS solver, as will be outlined in detail later, uses the MCP in this latter form of (2.1.12)-(2.1.15). It is then transformed into the problem of finding a zero of a nonsmooth function, the *normal map*, for the solution of which a stabilized Newton method is applied (see chapter 3).

It is obvious, that the MCP, when picking the box B to be the positive orthant ($l = 0, u = \infty$), reduces to the NCP (or the LCP, if f is linear). It is possible though, to reformulate an arbitrary MCP as a NCP, as shown in [18], but doing so and solving the latter is quite error prone, besides that the numbers of variables is usually higher; apart from that, the NCP solver would not be able to exploit possibly existing special structures of the problem. Therefore algorithms are usually aimed at solving complementarity problems in the MCP format (see [3]).

2.2 The Newton Method

All Newton-like optimization methods use in each step a quadratic approximation q^k of the objective function, the index k denoting the kth step) at the current point x^k, beginning at some starting point x^0, given as

$$q^k(x^k + s) = f(x^k) + (g(x^k))^T s + \frac{1}{2} s^T B^k s. \tag{2.2.1}$$

Here in (2.2.1), f is the objective function, g its gradient and B^k a suitable approximation to the Hessian matrix $G(x^k)$ of f at the current point x^k.

This quadratic approximation is being minimized by finding a zero of the approximations gradient. The minimizer of q^k, called Newton point, or (x_N^k), is used as the next iterate of the algorithm. In the following I will dismiss the indices k and some functions increments if the context makes the meaning clear, due to notational convenience. I will further equip variables which should be indexed $k+1$ with a bar.

Minimizing q is a well defined problem if B is positive definite. The gradient of q, $g + Bs$, is zero if and only if $s = -B^{-1}g$. Hence

$$p := B^{-1} g(x) \tag{2.2.2}$$

is used in the classical Newton method as the correction term for the current point x^k, i.e.

$$\bar{x} = x + p. \tag{2.2.3}$$

Most commonly a linesearch procedure is performed, after the Newton direction p is computed, searching for a point along the direction p that satisfies some descent criterion. Clearly p is a descent direction for f, since $g^T p = -g^T B^{-1} g$ which is nonpositive for $g \neq 0$ and B positive definite.

There are many different types of Newton-like methods, mostly varying in the choice, computation or factorization of the Hessian approximation B, but also in computational techniques, linesearch procedures, etc.

Most Newton type optimization procedures have excellent local convergence, but are not necessarily globally convergent. Damping strategies and different linesearch schemes may yield a larger domain of convergence, while convergence speed is usually slowed down.

As a conclusion, the procedure of the standard Newton-like method can be divided into three parts. The first part consists of finding a quadratic approximation of the

objective function (i.e. a linearization of the first derivative or gradient function), a problem which basically boils down to finding and computing an approximation of the Hessian. In a second step the search direction is evaluated, which is done by solving (2.2.2). The third step is the linesearch procedure along the Newton direction.

As will be outlined in the following chapters, COMPASS uses a damped Newton-like method in order to find a zero of a function $f : \mathbb{R}^n \to \mathbb{R}^n$, that can be compared with the first derivative g in the classical Newton procedure above. The problems that arise and imply a variation of the three steps described before, are that the objective function will be nonsmooth, and of special structure due to a projection operator. Hence a linearization, a search direction, as well as a linesearch would not make sense, and these concepts will be extended to a general first order approximation, a piecewise linear path, and a pathsearch procedure, respectively, as described in chapter 3.

2.3 A Pivot Technique

The purpose of this section is to describe a basic pivot technique, similar to Lemke‘s algorithm ([23]), or that of Dantzig, derived from his simplex method ([10]), for solving the linear complementarity problem (LCP).

In general there are two main methods for solving LCPs, *interior point methods* and *pivot methods*. The advantage of interior point methods is polynomial complexity of the problem in the problem dimension compared to exponential complexity in the case of pivot methods [3]. However, COMPASS uses such a pivot method, in order to obtain a path, connecting the iterates with each other. This is important, e.g. in the case that a solution can not be found: Points along the constructed path will yield progress in terms of some descent requirement (3.2.14), and can hence also be used as next iterates, so the work was not wasted (see also section 3.3).

The technique described here will provide the basis for the modified pivot technique used in the COMPASS algorithm, since, as shall be described in a succeeding chapter, the core work of COMPASS is the iterative solution of a series of linear mixed complementarity problems.

In Chapter 4 of [10], that gives a detailed description of the simplex method, the main focus lies on finding $\lambda_j \geq 0$, $j \in \{1,...,m\}$ for given n-vectors $P_0, P_1,...,P_m$, and

scalars $c_1, ..., c_m$ such that

$$\sum_{j=1}^{m}(\lambda_j P_j) = P_0, \tag{2.3.1}$$

and that the value

$$z = \sum_{j=1}^{m}(\lambda_j c_j)... = max \tag{2.3.2}$$

becomes maximal. In this context, a set of λ_j satisfying (2.3.1) is said to be a *feasible* solution, and, if in addition, the value of z is maximal, it is called a *maximal feasible* solution. The simplex method, which is not the focus of this section and hence shall not receive much more attention, is the procedure of finding such a maximal feasible solution.

Necessary for the simplex method to work, is an assumption of nondegeneracy of the problem, i.e. that every subset of n points of $P_o, ...,P_m$ is linearly independent, and hence forms a basis of $\mathbb{R}^n$, which motivates the following terminology: For a fixed j, the variable λ_j is called *basic*, if it is positive in (2.3.1), and *nonbasic* if it is equal to zero. A solution of (2.3.1) with at most n points P_i with positive weights λ_i and $n - m$, or more, points P_i with $\lambda_i = 0$, is called a *basic feasible solution.*

When applying a *pivot technique* for solving the linear complementarity problem (LCP), given $q \in \mathbb{R}^n, M \in \mathbb{R}^{n\times n}$, find z,$w \in \mathbb{R}^n_+$, such that

$$Mz + q = w, \tag{2.3.3}$$

$$z^T w = 0, \tag{2.3.4}$$

the terminology and the way in which the problem is cast are very similar to that of the simplex method. It is easy to see that the system of equations (2.3.3) can be rewritten in the manner of (2.3.1), where the variables w_i and z_i correspond to the λ_j:

$$Mz - Iw = -q, \tag{2.3.5}$$

where I denotes the $n \times n$ unity matrix. Hence the only difference between the two problems is the complementarity constraint (2.3.3) instead of a maximizing constraint, and a similar solution method can be applied. Both problems are solved by a sequence of *pivot steps*, a sequence of actions that are automatically performed according to

initially defined *pivot rules*. Specifying these rules for (2.3.5), and hence providing a solution algorithm for the LCP, will be the subject of the remainder of this section.

Regarding terminology, in a solution of the LCP, i.e. when (2.3.3) and (2.3.4) are satisfied, the components (or *variables*) w_i and z_i of the vectors z and w, respectively, are called *complementary* for all $i \in \{1,...,n\}$, and each of them is the *complement* of the other. Finding such a solution of the LCP is carried out in the form of (2.3.5).

For given M and q, a solution of (2.3.5) is said to be a *feasible solution*, if z and w are nonnegative. It is trivial to see that (2.3.5) is the same as

$$z_1 M_{:1} + ... + z_n M_{:n} - w_1 I_{:1} - ... - w_n I_{:n} = -q, \tag{2.3.6}$$

rewritten in terms of variables and columns, but the context of the following is more obvious in this notation. Each variable pair, z_i, w_i, in a feasible solution of (2.3.6) can hence be characterized as either a *complementary pair*, if one variable is zero and the other is positive, a *basic pair*, if both variables are positive, or a *nonbasic pair*, if both variables are zero. A solution is called *basic feasible solution, (BFS)*, if it is feasible, the number of variables (z_i and w_i) that are positive is exactly n, as is the number of zero valued variables, and if the columns in (2.3.6) corresponding to these nonzero variables form a basis of $\mathbb{R}^n$. If, additionally (2.3.4) is satisfied, i.e. if there are only complementary variable pairs, the solution is called *complementary basic feasible solution*. Finally, a basic feasible solution (BFS) is called *almost complementary* if it contains exactly one basic pair, exactly one nonbasic pair, and all the other variable pairs are complementary pairs.

Clearly, the basic feasible solutions of (2.3.5) are exactly the extreme points of the convex set

$$Z := \{z | Mz + q = w \geq 0, \quad z \geq 0\}. \tag{2.3.7}$$

With these preparations in mind, the problem of finding a solution of the LCP transforms to that of finding a *complementary basic feasible solution* of (2.3.5). The procedure of finding it assumes as given, an almost complementary BFS, which is also the end point of an *almost complementary ray*, an unbounded edge of Z, along which each point satisfies $z_i w_i = 0$ but for one index, β, with $z_\beta w_\beta > 0$. In practice such an almost complementary ray is relatively easy to find, if one entire column of M is positive. This can always be achieved by artificially augmenting M with an additional positive column, as shall be described in section 2.3.2.

The last missing ingredient for the description of the algorithm is the *assumption of nondegeneracy*, which states that each combination of n vectors of columns of the

matrices I and M, are linearly independent, hence form a basis of $\mathbb{R}^n$. This is a necessary assumption (basically on M) and is needed in each step of the algorithm.

2.3.1 Lemke's Algorithm

Given an almost complementary BFS, i.e. $z_i w_i = 0$ for all i except for one index, say β, with $z_\beta > 0$ and $w_\beta > 0$ (basic pair), and one index, say ν, with $z_\nu = w_\nu = 0$ (nonbasic pair), which is also the end point of an unbounded almost complementary edge of Z, the algorithm starts as follows:

The vector $M_{:\nu}$, the column of M corresponding to the variable z_ν of the nonbasic pair, enters the basis (hence equation (2.3.5)) with a coefficient θ, in a linear way. All other coefficients in (2.3.5) adapt also in a linear way, so that the equal sign still holds. Due to the assumption of nondegeneracy, any nonbasic column (i.e. a column corresponding to a nonbasic variable) of M, say $M_{:i}$, can be expressed as a linear combination of all other basic columns of M and I,

$$M_{:i} = \sum_{j=1}^{n} (x_{ji} M_{:j}) + \sum_{k=1}^{n} (y_{ki} I_{:k}), \tag{2.3.8}$$

where only the basic columns of M and I have nonzero weights x_{ji} or y_{ki}. Hereby, for each column $M_{:i}$, two vectors $X_i = (x_{j,i})_{j=1,\ldots,n}$ and $Y_i = (y_{k,i})_{k=1,\ldots,n)}$ are defined, which will be updated in every pivot step, since they depend on which columns of M and I are basic, and which covering vector will enter the basis next. In total usually exactly n columns will have nonzero weights in (2.3.8), which completely determines X_i and Y_i. In this terminology, (2.3.8) can be rewritten as

$$M_{:i} = MX_i + IY_i, \tag{2.3.9}$$

and after multiplying (2.3.9) for the index ν by θ and subtracting from (2.3.5), one obtains

$$\begin{aligned} &(z_1 - \theta x_{1\nu}) M_{:1} + \ldots + (z_n - \theta x_{n\nu}) M_{:n} - \\ &(w_1 - \theta y_{1\nu}) I_{:1} - \ldots - (w_n - \theta y_{n\nu}) I_{:n} + \theta M_{:\nu} = -q \end{aligned} \tag{2.3.10}$$

where again only the basic columns of M and I have nonnegative coefficients. Equivalently in compact notation this becomes

$$M(z - \theta X_\nu) - I(w - \theta Y_\nu) + \theta M_{:\nu} = -q. \tag{2.3.11}$$

The new variable θ is increased linearly, starting at zero. Equation (2.3.11), with positive θ, is not a basic feasible solution (but definitely an almost complementary feasible solution), since there are too many variables positive at this point. Along with θ, all other coefficients in (2.3.10) change linearly. Now two cases can occur. Either (i) all coefficients increase forever, i.e. $z_i - \theta x_{i\nu}, w_i - \theta y_{i\nu} \to \infty$ for all i, or (ii) eventually one coefficient becomes zero in (2.3.10), as θ increases. The first case is referred to as *ray termination*, and is described in section (2.3.2). In the latter case, the first coefficient that becomes zero, say $z_\zeta - \theta x_{i\zeta}$, determines a new BFS, since now there are exactly n columns with positive coefficients in (2.3.10), which form a basis of $\mathbb{R}^n$ because of the nondegeneracy assumption on M. The coefficients of the columns of M and I in such a resulting equation (representing this new BFS), then replace the variables z_i and w_i, i.e. a variable update is performed, $z_i := z_i - \theta x_{i\nu}$, $w_i := w_i - \theta y_{i\nu}$ for all i, $z_\nu := \theta$, such that

$$z_1 M_{:1} + ... + z_n M_{:n} - w_1 I_{:1} - ... - w_n I_{:n} + z_\nu M_{:\nu} = -q, \tag{2.3.12}$$

where again only basic variables are positive, the others are still zero. Only z_ζ became nonbasic, z_ν is now basic.

Besides being a BFS, this solution is also at least almost complementary; in case the coefficient that just became zero (with index ζ) was of a complementary pair, i.e. $\zeta \neq \beta$ (the index of the basic pair), clearly $z_\zeta = 0$ and $w_\zeta = 0$ constitute a new nonbasic pair; hence the solution consists of exactly one (new) nonbasic pair, namely the old basic pair, and all other pairs are complementary pairs. In this case, w_ζ, or in general the complement of the variable that just left the basis (became zero), which is of the new nonbasic pair, will be the next variable to enter (2.3.12) in the next step: The vectors X_ζ and Y_ζ are computed, and the procedure is repeated. This happens whenever a coefficient becomes zero in (2.3.10), or equivalently, whenever a column of M or I leaves the basis. The procedure of reaching from one basic feasible solution to the next and updating the corresponding variables is called a *pivot step*.

In case the new nonbasic variable z_ζ (the one that just turned zero) was the complement of a *basic* variable $w_\zeta \neq 0$, then $\zeta = \beta$, i.e. the basic pair turned into a complementary pair. In this case the algorithm terminates successfully, since all pairs are complementary pairs, hence a new complementary BFS has been found, and the LCP has been solved.

2.3.2 Termination and Existence of Solutions

Increasing the coefficient corresponding to the column $M_{:\nu}$ (or $I_{:\nu}$), the covering vector of the entering variable of the nonbasic pair, with the help of θ in (2.3.10), as outlined above, will either go on forever without another variable becoming zero, or eventually force another basic variable to become zero and leave the basis. In the first case an almost complementary ray is created, and the pivot procedure terminates. This is a successful termination, in the sense, that one can be sure that the algorithm cannot find a solution, if the procedure was started at such an almost complementary ray, as the next result implies (see [10], p.314):

Theorem 2. *If the path along the edges of the set Z, generated by the pivot procedure described above, is initiated at the end point of an almost complementary ray, then the procedure terminates either in another almost complementary ray, or in a complementary BFS of (2.3.5).*

In case the algorithm is started at an arbitrary extreme point of Z, the procedure may cycle, i.e. the initial almost complementary BFS may reoccur after some iterations. Fortunately, an almost complementary BFS of (2.3.5), also being the end point of an almost complementary ray of Z, as is obviously desired, since then cycling can not happen because of theorem 2, can always be achieved by the following method: Consider the system of equations

$$Mz + q + z_0 e = w, \tag{2.3.13}$$

where z_0 is an artificial variable, and e its *covering vector*, the unity vector (1,...,1). A solution (z, w, z_o) of (2.3.13) with $z^T w = 0$ is said to be complementary if $z_0 = 0$, and almost complementary if $z_0 > 0$. Now for the set

$$Z_0 := \{z | Mz + q + z_0 e = w \geq 0, \quad z \geq 0 \quad z_0 \geq 0\}, \tag{2.3.14}$$

consider the following almost complementary ray of Z_0: Let all w_i be basic, all z_i be nonbasic, then for z_0 sufficiently large,

$$w = q + z_0 e > 0, \tag{2.3.15}$$

and hence determines an almost complementary feasible solution of (2.3.13), with only complementary pairs z_i, w_i. If now z_0 is decreased to its maximal value, say $z_0 = z_0^*$, so that for $w^* = q + z_0^* e$, $w_i^* > 0$ for all i but for one index ν, where $w_\nu^* = 0$, then $w^* = q + z_0^* e$, z_0^*, $z = 0$ constitutes an almost complementary BFS of (2.3.13),

with exactly one nonbasic pair $z_\nu = w_\nu = 0$.

The algorithm is initiated at this point. Now z_ν enters the basis with its covering vector $M_{:\nu}$, and the algorithm proceeds as described in the preceding section. In the case that z_0 leaves the basis at value zero, a complementary basic feasible solution of (2.3.13) has been found, which is also a complementary BFS of (2.3.5).

The remainder of this section provides some results about existence of solutions of the LCP, as well as on the termination properties of the algorithm, depending on special properties of the matrix M. In order to present them, some definitions are in order.

A matrix $M \in \mathbb{R}^{n\times n}$ is said to be *copositive plus*, if it satisfies

$$u^T Mu \geq 0 \text{ for all } u \geq 0, \tag{2.3.16}$$

$$(M + M^T)u = 0 \text{ if } uMu = 0 \text{ and } u \geq 0. \tag{2.3.17}$$

This class of matrices includes for example all *strictly copositive* matrices, i.e. matrices satisfying $u^T Mu > 0$ for $u \geq 0$ but $u \neq 0$, as well as all positive semidefinite matrices, i.e. those satisfying $u^T Mu \geq 0$ for all u.

The (i,j) *minor* (often denoted $[M_{ij}]$) of an $n \times n$ matrix M is defined as the determinant of the matrix formed by removing from M its i^{th} row and j^{th} column. For a set $K \subset \{1,...,n\}$, the (K,K) *minor*, $[M_{KK}]$, the determinant of the matrix obtained by removing simultaneously the columns $M_{:K}$ and rows $M_{K:}$ of M, is called a *principal minor* of M. Theorem 3 summarizes the relevant results, proofs of which can be found in chapter 23 of [10].

Theorem 3. *(i) For a copositive plus matrix M, if the algorithm terminates in a ray, then the LCP has no solution.*
(ii) If M is strictly copositive, the process terminates in a complementary BFS of (2.3.13).
(iii) If M has positive principal minors, the process terminates in a complementary basic feasible solution of (2.3.13) for any q.

2.4 Preprocessing an MCP

The algorithm PATH in its current implementation uses a so called *preprocessor* that simplifies the MCP in order to ease and quicken the solving procedure for the actual algorithm. The methods used in the preprocessor are different from each other, and apply to certain specifics of the MCP to be solved. Many different cases are

considered because not all of the methods may be applied to each single problem. After the problem was simplified, and that simplified problem was solved, the solution has to be transformed back in order to fit the original MCP.

This preprocessing routine is not necessary for PATH in order to solve a problem, but fasten and improve its performance, similar like the crash technique, as described in section 2.5. The broadness of the preprocessor, and its character of being of "only" additional help to an MCP solver were the reasons for not implementing it in the COMPASS code, for doing so would have exceeded the scope of this thesis. Still the main components of the preprocessing routine of PATH shall be listed here for matters of completeness, but also in order to provide some insights for the interested reader, and a teasing bite to lure the indefatigable programmer into extending the code for a powerful preprocessor. For a greater degree of details the reader is referred to the paper of Michael C. Ferris and Todd S. Munson [17], that was released in 1999, when the preprocessor was implemented in the PATH algorithm. This publication includes complete documentation on the here only outlined procedures and, of course, served as main source of literature for this chapter.

The preprocessor is active at three points during the PATH algorithm, the first time before solving the problem:

The idea in this step is to uncover and exploit polyhedral structure in an MCP: Consider an MCP given as a box constrained variational inequality

$$0 \in f(x) + N_{[l,u]}(x), \tag{2.4.1}$$

where $N_{[l,u]}(x)$ is the normal cone, see (2.1.2) of $[l,u]$ at x. If the variables can be split into (x,y), and (2.4.1) is of the form

$$0 \in \begin{bmatrix} F(x) - A^T y \\ Ax - b \end{bmatrix} + \begin{bmatrix} N_X(x) \\ N_Y(y) \end{bmatrix}, \tag{2.4.2}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, X a Cartesian product of closed intervals, and Y a Cartesian product of intervals of the form $\mathbb{R}$, $\mathbb{R}_+$, or $\mathbb{R}_-$, then the following theorem can be applied in order to simplify the MCP, as was proved by Robinson [31] in 1998.[1]

Theorem 4. *Given an MCP with the structure of (2.4.2), X, Y as above, then*

1 Theorem 4 consists exactly of propositions 1 and 2 of [31].

- *If* $(\bar{x},\bar{y})$ *solves (2.4.2) then* $\bar{x}$ *solves*

$$0 \in F(x) + N_{X\cap\{x|b-Ax\in Y^0\}}(x), \tag{2.4.3}$$

where $Y^0 := \{y|y^T\bar{y} \leq 0, \forall \bar{y} \in Y\}$ *is the* polar cone *of* Y*.*

- *If* $\bar{x}$ *solves (2.4.3), then the optimization problem*

$$\begin{array}{ll} \min\limits_{y\in Y} & (A\bar{x}-b)^T y \\ s.t. & 0 \in F(\bar{x}) - A^T y + N_X(\bar{x}) \end{array} \tag{2.4.4}$$

has a nonempty solution set, and for any $\bar{y}$ *solving (2.4.4),* $(\bar{x},\bar{y})$ *solves (2.4.2).*

The preprocessor needs information which of the elements in the Jacobian of f are linear and which are nonlinear.[2] Theorem 4 is applied to one single row and column with the necessary skew symmetric structure of (2.4.2). In the resulting problem of the form (2.4.3) it is checked, whether the general constraint $b - Ax \in Y^0$ of the polyhedral set $X \cap \{x|b - Ax \in Y^0\}$ can be moved into the bound constraint X, yielding new sets, $\widetilde{X}$ and $\widetilde{Y}^0$. The new polyhedral set $\widetilde{X} \cap \{x|b - Ax \in \widetilde{Y}^0\}$ is identical to the old one, but the MCP obtained after the transformation via theorem 4 is typically simpler.

In the first stage the preprocessing routine performs three kinds of reductions in an iterative manner; each time only one row and one column with the skew symmetric structure is detected, and the problem is checked for reductions in $X\cap\{x|b-Ax \in Y^0\}$ with the help of theorem 4:

1. *Simple reductions* can be made to the MCP in three cases. First, when an eligible row contains zero elements, when a row contains one element, or when a row contains elements in exactly two columns, one of them being a column singleton. All these cases result in the reduction of at least one variable of the problem.

2. Constraints for which only one point is feasible, given the variable's bounds, are called *forcing constraints.* A large reduction in the size of the problem might be possible, if one knows that only one solution is possible for a certain constraint, since all variables appearing in the constraint can be fixed.

2 Such routines already exist for some programming languages, e.g. this information is already provided in the GAMS environment (see [6]) for PATH; otherwise, it would have to be supplied by the user.

3. With the help of an existing algorithm (see [35]) it is possible to find *duplicate rows* in the Jacobian; clearly redundant rows can cause problems to many solvers due to singularity of the matrix. Any inconsistencies are determined, and typically one of the constraints in relation with one of the redundant rows might be removed.

First simple reductions are checked in $X \cap \{x | b - Ax \in Y^0\}$, then forcing constraints, then redundant rows. Once such a turn is made, the new Jacobian is again checked for simple reductions, and another turn is taken, since the structure of the Jacobian may change and originally nonlinear rows and columns may become linear after reductions of variables or omission of constraints. This goes on until no more eligible rows and columns are detected.

The dimension of the new problem is reduced through these reductions. All reductions are remembered in the *stack* of the preprocessor. After each single transformation the information on the change made to the problem is put on top of the stack. After solving the simplified problem, it can, again iteratively, be transformed back in its original form in the exact order of this stack in order to obtain the correct solution to the original problem.

The second part of the preprocessor is invoked just before the actual algorithm is solved, after the first part, where polyhedral structure in an MCP is exploited and the MCP is transformed into a simpler MCP. Now, complementarity theory is used to eliminate variables by exploiting structure of rows and columns of the Jacobian. The changes made to the problem in this step do not have to be remembered, neither transformed back; the MCP is not touched in this step, just variables, whose values can be uniquely determined before solving are fixed and removed:

1. The *range of f over X* can be calculated in order to obtain lower and upper bounds for $f(z)$. If an upper bound is lower than zero, or if a lower bound is greater than zero, the corresponding variable can be fixed and hence removed form the problem.

2. Again, *duplicate rows and columns* can be determined (with the same routine as above, [35]). Also here, depending on special cases, variables can be fixed, or row or column duplicates removed.

3. Special *double block structure* of the problem can be exploited by the preproces-

sor. If the problem is of the form

$$\begin{bmatrix} F(x) \\ G(x,y) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{2.4.5}$$

with $F : \mathbb{R}^n \to \mathbb{R}^n$, the problem can be split into two problems that are consecutively solved: The solution $\bar{x}$ to $F(x) = 0$ then serves as exogenous input to solve $G(\bar{x},y) = 0$, for y only.

After this second step, the reduced problem is passed to the actual solver, which provides the solution to this reduced, simplified problem. In order to obtain the solution to the original problem, the information on the stack is taken (beginning from the top, i.e. the last transformation made in the first step of the preprocessor) and the solution is transformed back. This involves transforming the problem into the polyhedrally constrained setting (2.4.3) and to reverse the initial changes. Problem (2.4.4) is then solved using $\bar{x}$ in order to obtain $\bar{y}$, i.e. the solution $(\bar{x},\bar{y})$ to the problem before the current modification (top information of the stack) was made.

This procedure is prolonged until the stack is empty.

2.5 A Crash Technique

A crash technique is a method for finding an approximation to the active set at the solution of the problem one wants to solve, before the actual algorithm starts.

The main work in COMPASS is a pivot technique, similar to the one described in the previous section. At the beginning of the algorithm, determining the right set of basic and nonbasic variables (see section 3.3) would require too much work of the pivot technique, work, that the crash technique does a lot more quickly.

The crash technique provides an efficient start of COMPASS's optimization procedure: The starting point might have a completely different active set than the solution, hence a point is determined (that will later correspond to the basis in the pivot technique) with an active set that will typically be a good approximation to the active set at the solution. As the algorithm proceeds, the basis for the pivotal technique is taken to be the same as the one at which the technique terminated in the last iteration; so with the help of the crash technique, the pivot technique will be called much fewer times than without it.

This procedure was implemented in the PATH solver in 1997, see [14].

There are different types of crash techniques given in the literature for MCPs in the form of normal maps, basically they differ in how to get the search direction d^k in (2.5.1) below. In COMPASS, as in PATH, a projected Newton technique is used.

For a given $z^k \in B$, let

$$z(\alpha) := \pi_B(z^k - \alpha d^k), \tag{2.5.1}$$

where α is a search parameter, and d^k is the search direction. The next iterate z^{k+1} will be chosen to be $z(\alpha)$, for a suitable α. Due to the projection operator in (2.5.1), $z(\alpha)$ will be piecewise linear. All that remains before performing a search for α along this path, is to determine an appropriate search direction d^k (the detail in which most crash techniques differ from each other).

The set of all active indices, with right value of the according index of the function f, $\mathcal{A}$, is defined:

$$\mathcal{A} := \{i \in \{1,...,n\} \quad | \quad (l_i = z_i, f_i(z) \geq 0) \quad or \quad (u_i = z_i, f_i(z) \leq 0)\}. \tag{2.5.2}$$

$\mathcal{A}$ is the set of the so called *fixed indices*, for which the requirements of the MCP are already satisfied; hence for the reduced system, consisting only of the indices in $\mathcal{A}$, the MCP is already solved.

The set $\mathcal{I}$ is defined to be the complement of $\mathcal{A}$:

$$\mathcal{I} := \{i \in \{1,...,n\} \quad | \quad i \notin \mathcal{A}\}. \tag{2.5.3}$$

The direction d^k is now computed via the reduced system

$$\nabla f_{\mathcal{I}\mathcal{I}}(z^k) d^k_{\mathcal{I}} = f_{\mathcal{I}}(z^k). \tag{2.5.4}$$

Here, $d^k_{\mathcal{I}}$ is the Newton direction of the reduced system, that considers only those indices, that are not yet fixed.

Now d^k is obtained by augmenting $d^k_{\mathcal{I}}$ with zeros for all indices in $\mathcal{A}$, so that a search in direction d^k will correspond to searching only along the indices in $\mathcal{I}$. This direction is, by construction, exactly the Newton direction of the reduced system. Speaking geometrically, the direction is chosen as to minimize f in the face of B that is determined by the currently active variables, where the component values of f of these variables also have the right sign, i.e. point inside the set B. The search along this direction can, and will in most cases, exceed the current face of B, and hence lead to a redefinition of the sets $\mathcal{A}$ and $\mathcal{I}$ in each iteration of the crash technique.

Unlike in the path generation phase of the COMPASS algorithm, where only one index at each pivot step changes from basic to nonbasic and otherwise (see section 3.3), during this crash technique many indices will change from inactive to active, and vice versa simultaneously in each step.

The purpose of this technique is to reach a point near the solution quickly, ignoring the different constraints of the set B when searching the direction, only focusing on a decrease in f, and on a change in the active (or fixed) index set:

Now since $z(\alpha)$ can be determined completely by α, in (2.5.1), α is chosen to reduce $\|f_B(x(\alpha))\|$, the norm of the *normal map* (see (3.1.1)), at $x(\alpha)$ that is defined as

$$x(\alpha) := \underset{x}{\operatorname{argmin}}\{\|f_B(x)\| \quad | \quad z(\alpha) = \pi_B(x)\}. \tag{2.5.5}$$

The resulting search for an appropriate α, along the resulting piecewise linear path $z(\alpha)$ defined in (2.5.1), is carried out in terms of the variable x (whose projection on B is $z(\alpha)$), and uses $\|f_B(.)\|$ as a merit function and chooses the least $m \in \{0,1,2,...\}$, such that for $\alpha = (\frac{1}{2})^m$,

$$\|f_B(x(\alpha))\| \leq (1 - \sigma\alpha)\|f_B(x(0))\|. \tag{2.5.6}$$

Determining $x(\alpha)$ in (2.5.5) for a given α is actually a trivial task. After $z(\alpha)$ is computed from (2.5.1), searching for an x satisfying (2.5.5) is equivalent to searching for an x in $N_B(z(\alpha))$ that minimizes the norm of f_B, where

$$f_B(x) = f(z) + x - z, \tag{2.5.7}$$

if z is the projection of x onto B (i.e. $z = z(\alpha)$). Since any x in $N_B(z)$ can be expressed by a unique pair of vectors w,$v \geq 0$ satisfying the complementarity conditions $w^T v = 0$, $w^T(z - l) = 0$, and $v^T(u - z) = 0$, as

$$x = z - w + v, \tag{2.5.8}$$

the problem may be transformed into that of finding such vectors w and v with the above properties, that minimize the norm of

$$f(z) - w + v, \tag{2.5.9}$$

being equal to $f_B(x)$. This transformed problem is solved by simply projecting $-f(z)$ onto $N_B(z)$, yielding the vectors w and v, which minimize (2.5.9) due to the minimum distance property of the canonical projection (see [14]).

So for each α in the search, computing the vectors $w(\alpha)$ and $v(\alpha)$, and hence the desired $x(\alpha)$ from (2.5.8) can be done in the same loop that checks the condition (2.5.6).

In case the submatrix $\nabla f_{\mathcal{II}}(z^k)$ used in (2.5.4) is rank deficient, a *perturbation parameter* ε is used in order to guarantee the system (2.5.4) to be soluble. ε is chosen just large enough, so that the matrix $\nabla f_{\mathcal{II}}(z^k) + \varepsilon I$ is invertible, (i.e. the crucial entries therefore are artificially made numerically nonzero). This technique was proposed by Stephen C. Billups [4], who proved that it can enable an algorithm to escape from a region near a local minimizer with nonzero $\|f_B(z^k))\|$. At subsequent crash steps, ε is reduced based on the residual of the merit function, see [24].

The crash procedure is implemented in the COMPASS solver at the beginning of the procedure, before the actual solution algorithm starts, in order to find a good approximation of the active set at the solution quickly. However, near to the solution it certainly does not guarantee convergence, as the theory behind the actual COMPASS algorithm (the non-monotone stabilization scheme and the general descent framework, see sections 3.4 and 3.5) does. Hence, the crash algorithm should be stopped as soon as one of the following criteria is satisfied:

1. There is no change in the active set.
2. Decrease in $\|f_B(x(\alpha))\|$ becomes smaller than $\sigma = 0{,}05$ (there is no theoretical assurance that the directions used are descent directions).
3. The path length α becomes lower than $\alpha_{min} = 2^{-12}$.
4. The maximum number of steps in the crash phase exceeds $k_{max} = 50$.

Additionally, the crash phase is omitted, if the problem dimension is less than $n_{min} = 10$.

The numbers used in these conditions were shown to be most efficient (in terms of speed and numerical improvement of the objective function) by Steven P. Dirkse and Michael C. Ferris in 1997. It was shown in their paper ([14]), that implementing this crash technique at the beginning of the algorithm substantially improves efficiency of the optimization procedure, and is hardly ever costly to perform.

2.6 The Merit Function in COMPASS

A merit function is typically used as a measure of descent in optimization problems. It might not be the actual objective function of the problem, but a related function,

which depicts the scale of descent more accurately, or is easier to handle. The merit function used in COMPASS and discussed in this section was developed by Michael C. Ferris, Christian Kanzow and Todd S. Munson in 1999. It is part of a general descent framework developed in their work (see [15]), which was implemented into version 4.0 of the PATH solver and replaced the euclidean norm of the normal map in the algorithms descent requirements.

The MCP can be reformulated in terms of the Fisher-Barmiest function, a mapping $\phi : \mathbb{R}^2 \to \mathbb{R}$ which is defined as

$$\phi(a,b) := \sqrt{a^2 + b^2} - a - b. \tag{2.6.1}$$

It has been widely used concerning complementarity problems of all kinds, since it satisfies the properties of an NCP-function:

$$\phi(a,b) = 0 \qquad \Longleftrightarrow \qquad a \geq 0, \quad b \geq 0 \quad \text{and} \quad ab = 0. \tag{2.6.2}$$

As proposed by Stephen C. Billups in [3] , the operator $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ is defined component wise as follows:

$$\begin{aligned}
\Phi_i(z) &:= \phi(z_i - l_i, f_i(z)) \\
&\quad \text{if } i \in \{i \in I | -\infty < l_i < u_i = +\infty\}, && (2.6.3)\\
\Phi_i(z) &:= -\phi(u_i - z_i, -f_i(z)) \\
&\quad \text{if } i \in \{i \in I | -\infty = l_i < u_i < +\infty\}, && (2.6.4)\\
\Phi_i(z) &:= \phi(z_i - l_i, \phi(u_i - z_i, -f_i(z))) \\
&\quad \text{if } i \in \{i \in I | -\infty < l_i < u_i < +\infty\}, \text{ and} && (2.6.5)\\
\Phi_i(z) &:= -f_i(z) \\
&\quad \text{if } i \in \{i \in I | -\infty = l_i < u_i = +\infty\}. && (2.6.6)
\end{aligned}$$

The following result can easily be shown (see e.g. [3]).

Theorem 5. *Given a function f from $\mathbb{R}^n$ into itself and a box $B = [l,u]$, where $l,u \in \bar{\mathbb{R}}^n$, let Φ be defined as in (2.6.3)-(2.6.6), then a vector $z^* \in \mathbb{R}^n$ is a solution of MCP(f,B) if and only if it solves the system of nonlinear equations $\Phi(z) = 0$.*

The merit function used in COMPASS is the function $\Psi : \mathbb{R}^n \to \mathbb{R}$ defined as

$$\Psi(z) = \frac{1}{2}\Phi(z)^T\Phi(z). \tag{2.6.7}$$

While Φ is usually nondifferentiable, it was shown in [15] that Ψ is continuously differentiable everywhere. Another result of this paper was that a stationary point of the constrained reformulation of the MCP,

$$\min \quad \Psi(z) \qquad \text{s.t.} \quad z \in [l,u] \tag{2.6.8}$$

is a solution of MCP(f,B), under some appropriate assumptions on the principal submatrix of f. These properties make Ψ more than suitable to be the merit function used in the algorithm.

3 The COMPASS Solver

The COMPASS algorithm was developed in the course of this thesis; it is a free clone of the PATH solver, which has become the standard solver for Mixed Complementarity Problems. The purpose of this thesis is to provide detailed understanding of the PATH solver, and to describe the implementation of COMPASS (the developed algorithm was implemented in MATLAB/Octave, see section 3.6), rather than to develop a solver that could be compared with PATH in terms of performance. However, the interested programmer is more than welcome to improve and enhance the code and make COMPASS more competitive. The details and techniques described in this section match those in the PATH solver. Any discrepancies or modifications will explicitly be mentioned.

The COMPASS solver is an implementation of a stabilized Newton method for solving the MCP. In order to apply a Newton-like method here, it is necessary to transform the MCP into the equivalent problem of finding a zero of a function, namely the usually nonsmooth *normal map*, as proposed by Robinson [30]. The normal map would correspond to the gradient function when applying a standard Newton method to a at least continuously differentiable function from $\mathbb{R}^n$ into $\mathbb{R}$. Nonsmoothness of the normal map is what causes most of the complexity of the COMPASS algorithm, which is outlined briefly in the remainder of this introduction.

Similar to linearization of the gradient function (or quadratic approximation of the objective function) in standard Newton-like techniques, a general first order approximation of the normal map is computed.

Finding a zero of this approximation, in the following referred to as the *Newton point*, turns out to be equivalent to solving a *linear MCP*. The solution of the latter is found with the help of a special pivot technique, similar to the one used by Lemke [23], as described in section (2.3). The result of this pivot algorithm is not only a solution of the linear MCP, which is equal to the solution of the approximation (i.e. the Newton point), but also a *piecewise linear path* from the current point to the Newton point. This path is the element in the COMPASS algorithm corresponding to the Newton direction p^k in a standard Newton method.

Whether this point is accepted as the next iterate, depends on the *non monotone*

stabilization (NMS) - scheme. If it is not accepted, the generated path constructed between the current point and the Newton point is searched for an acceptable point. This procedure is called *pathsearching*, and corresponds to the linesearch procedure in the smooth case.

The Algorithm terminates successfully when the zero of the merit function, and hence the normal map has been found. As will be stated soon, the zero of the normal map corresponds to a solution of the MCP, so in that case the problem is solved.

In the following the algorithm is described in detail. An implementation of COMPASS in MATLAB is provided at the end of this section.

3.1 The Normal Equation and its Approximation

The COMPASS solver, as PATH and most other solvers for Complementarity Problems make use of the MCP formulation (2.1.9)

$$\begin{array}{ll} \text{Given} & f : \mathbb{R}^n \to \mathbb{R}^n \text{ and a box } B = [l,u], \text{ where } l,u \in \overline{\mathbb{R}}^n, \\ \text{find} & z \in B, \quad w,v \in \mathbb{R}^n_+ \\ \text{s.t.} & f(z) = w - v \\ & (z-l)^T w = 0 \\ & (u-z)^T v = 0. \end{array}$$

rather than the NCP or VI formulations, even though in most fields of application the problems could be formulated in different and simpler forms.

In order to solve MCP using a Newton-like technique, it is necessary to recast the problem in the form of a function that should be minimized. For this purpose, it is suitable to consider the *normal equation (NE)*,

$$f_B(x) := f(\pi(x)) + x - \pi(x) = 0, \tag{3.1.1}$$

where f_B, the *normal map*, is defined with the help of π, the canonical projection of $x \in \mathbb{R}^n$ onto B. The definition of the normal map, as well as the theorem below, and the approximation described in the next section were first proposed by Stephen M. Robinson in 1992, see [30].

The following result yields a one to one correspondence between points $x \in \mathbb{R}^n$, solving NE, the normal equation (3.1.1), and triples (z,w,v) solving MCP. This becomes obvious if one remembers the meaning of the vectors w and v, the positive and negative parts of $f(z)$ at a solution z of MCP, as they have already been

characterized in section (2.1.3).

Theorem 6. *If $x \in \mathbb{R}^n$ solves NE, then $z = \pi(x)$ is a solution of MCP. Conversely, if z is a solution of MCP, then $x = z - f(z)$ solves NE. At such a solution $x = z - w + v$.*

Hence for finding a solution of MCP, a Newton method may be applied to solve NE, which is exactly what constitutes the core process of COMPASS.

If the procedure described here was to be compared to a standard Newton-like method applied to a smooth real valued function, the normal map would correspond to the first derivative of the objective function, i.e. the gradient function. The main difference that causes problems here is that the normal map is in general nonsmooth, due to the projection operator π. Thus, a linear approximation of f_B, like in the case with a smooth first derivative, doesn't make sense. This problem is dealt with by approximating the normal map with a general first order approximation.

A general fist order approximation of a function $f : \mathbb{R}^n \to \mathbb{R}^n$ at a point $y \in \mathbb{R}^n$ is a mapping $A : \mathbb{R}^n \to \mathbb{R}^n$, such that

$$\lim_{x \to y} \frac{\|f(x) - A(x)\|}{\|x - y\|} = 0. \tag{3.1.2}$$

As for the COMPASS algorithm, the normal map f_B is approximated at the current point x^k by the point-based first order approximation A_k, obtained by linearizing f around $\pi(x)$ in (3.1.1), yielding

$$A_k(x) = f(\pi(x^k)) + (\nabla f(\pi(x^k)) + \varepsilon I)(\pi(x) - \pi(x^k)) + x - \pi(x), \tag{3.1.3}$$

where ε is a perturbation parameter as proposed by Steven C. Billups [4], similarly defined and updated as in the crash procedure in section 2.5, see [24]. Equivalently, using $M^k := \nabla f(\pi(x^k)) + \varepsilon I$ and $q^k := f(\pi(x^k)) + M^k \pi(x^k)$, the approximation becomes

$$A_k(x) = M^k \pi(x) + q^k + x - \pi(x). \tag{3.1.4}$$

While the zero of this approximation of the normal map may not be unique, the path generation procedure, described in the next section, however, provides a unique point, which will be referred to as x_N^k, or *Newton point* in the remainder of this chapter. It is not hard to see, that finding an x in $\mathbb{R}^n$ satisfying (3.1.4) constitutes a linear MCP, for the solution of which a pivot procedure will be used similar to that described in section 2.3.

3.2 Path Generation

This section describes, how, starting from the current point x^k, the COMPASS algorithm computes a unique path to the zero of the approximation A_k. The new iterate x^{k+1} will either be this Newton point, i.e. the zero of the approximation, or an appropriate point along the constructed path, in case the Newton point will not provide satisfying descent in the merit function.

The problem of finding a zero of the approximation A_k obtained in the last section is easily detected to be a linear MCP, if the following relationships (similar as in Theorem 6, where a solution (z,w,v) of MCP corresponds to a solution $x = z - w + v$ of NE) are considered:

For means of finding a zero of the approximation A_k, i.e finding an $x \in \mathbb{R}^n$, such that

$$A_k(x) = M^k\pi(x) + q^k + x - \pi(x) = 0, \tag{3.2.1}$$

the variables z,w, and v, are defined as

$$z := \pi(x), \qquad \text{the projection of } x \text{ onto } B = [l,u], \tag{3.2.2}$$

$$v := (x - z)_+, \qquad \text{the positive part of } (x - z), \text{ and} \tag{3.2.3}$$

$$w := (z - x)_+, \qquad \text{the negative part of } (x - z), \tag{3.2.4}$$

for each $x \in \mathbb{R}^n$. It is not hard to see that with these definitions any $x \in \mathbb{R}^n$ can be uniquely characterized as

$$x = z - w + v, \tag{3.2.5}$$

hence these vectors z, w and v are said to be the *components* of x. Furthermore,

$$v \geq 0, \quad w \geq 0, \quad \text{and} \quad v^T w = 0. \tag{3.2.6}$$

Vectors satisfying (3.2.2)-(3.2.6) will be called *valid triples* in the following, and are said to *comprise* x.

In these terms finding $x \in \mathbb{R}^n$ solving (3.2.1) is equivalent of finding a valid triple (z,w,v) comprising x, such that

$$M^k z + q^k - w + v = 0. \tag{3.2.7}$$

The latter problem now turns out to be the following **linear MCP**:

$$\begin{aligned}
&\text{Given} \quad M^k \in \mathbb{R}^{n\times n}, q^k \in \mathbb{R}^n \text{ and a box } B = [l,u], \text{ where } l,u \in \overline{\mathbb{R}}^n, \\
&\text{find} \quad z \in B, \quad w,v \in \mathbb{R}^n_+ \\
&\text{s.t.} \quad M^k z + q^k = w - v \qquad (3.2.8) \\
&\qquad (z-l)^T w = 0 \qquad (3.2.9) \\
&\qquad (u-z)^T v = 0. \qquad (3.2.10)
\end{aligned}$$

Hence finding a zero of the approximation of the normal equation is in fact a linear MCP, which is solved within COMPASS by a pivot technique similar to the one used by Lemke [23], as described in section 2.3. This pivot technique will not only provide the desired Newton point, but also a path from the current point x^k to the Newton point x^k_N.

Any such path can be parametrized as $p^k(t)$, $t \in [0,T]$, $T \leq 1$, and must clearly satisfy

$$p^k(0) = x^k \text{ and } p^k(T) = x^k_N. \qquad (3.2.11)$$

More generally, for any Newton like optimization technique, a path, that satisfies (3.2.11), and additionally

$$A_k(p^k(t)) = (1-t) f_B(x^k) \quad \forall t \in [0,T] \qquad (3.2.12)$$

is called a *Newton path*. In our case the norm of the approximation A_k will actually go to zero linearly along the path that will be constructed here, and (3.2.12) will be satisfied. This fact, and the definition of A_k imply

$$\|f_B(p^k(t))\| = \|A_k(p^k(t)) + o(t)\| = (1-t)\|f_B(x^k)\| + o(t), \qquad (3.2.13)$$

and for $\sigma \in (0,1)$ and t small,

$$\|f_B(p^k(t))\| \leq (1-\sigma t)\|f_B(x^k)\|. \qquad (3.2.14)$$

Hence the norm of the normal equation must decrease along the path, at least near $t = 0$. In this context, p is said to satisfy the requirements of a *descent path* for the normal equation. Before the smooth merit function Ψ was introduced in 1999 [15], (3.2.14) was used as descent criterion for possible new iterates while searching the path in case the Newton point was not acceptable. The relaxation parameter σ was

taken to be close to zero, so that almost any decrease in $\|f_B\|$ was acceptable [12].

Any iterate x^k of the optimization procedure is comprised of, and hence can be expressed uniquely in terms of a triple (z^k,w^k,v^k) using the identity (3.2.5). Since this can be applied to any point in $\mathbb{R}^n$, and hence any point lying on the path, $x_s = p^k(s)$ for some $s \in [0,T]$, this motivates the characterization of the desired path p^k as

$$p^k(t) = x^k(t) = z^k(t) - w^k(t) + v^k(t) \quad \forall t \in [0,T], \tag{3.2.15}$$

where in this notation $x_0^k = x^k$ and $x^k(T) = x_N^k$, analogously for the components, $z^k(t), w^k(t)$ and $v^k(t)$, that will form valid triples for each $x^k(t)$ and hence *comprise the path.*

The pivot procedure described in the following section will construct the path from x^k to x^{k+1} by solving the linear MCP (3.2.8)-(3.2.10). Each pivot step will result in a linear piece of the path, and the path will satisfy the requirement of a Newton path (3.2.12).

3.3 The Pivot Technique in COMPASS

This section describes how to obtain, starting from the current point x^k, the next iterate, x^{k+1}, which usually is also the Newton point x_N^k (the zero of the approximation A_k), and additionally a piecewise linear path $p^k(t)$, connecting them to each other. This is done by solving the linear MCP (3.2.8)-(3.2.10), which is an equivalent problem with coinciding solutions, as shown in the previous section. Similar to the procedure described in section 2.3, a pivot procedure is used to solve this linear MCP. Each pivot step of the procedure will provide a linear piece of the path from x^k to x^{k+1}.

Other than in section 2.3, instead of having to solve

$$Mz - Iw = -q, \tag{3.3.1}$$

for z and w with $z^T w = 0$, here the problem is that of finding a valid triple (z,w,v) such that

$$M^k z - Iw + Iv = -q^k, \tag{3.3.2}$$

that then comprises x_N^k, the solution of $A_k(x) = 0$ (see (3.2.1) and (3.2.7)).

It will be convenient to make a formal distinction between the indices; because of the complementarity conditions (3.2.2)-(3.2.6), at any point during the procedure,

for each $i \in \{1,...,n\}$, exactly one of the three following cases can occur:

$$\text{(i)} \quad l_i < z_i < u_i, \quad w_i = v_i = 0 \tag{3.3.3}$$

$$\text{(ii)} \quad l_i = z_i, \quad w_i \geq 0, \quad v_i = 0 \tag{3.3.4}$$

$$\text{(iii)} \quad z_i = u_i, \quad w_i = 0, \quad v_i \geq 0. \tag{3.3.5}$$

For the remainder of this section, the *assumption of nondegeneracy* will be necessary. It requires that all columns $M^k_{:i}$, for those i satisfying (i) (the covering vectors of the variables $l_i < z_i < u_i$), all columns $I_{:i}$, for those i satisfying (ii) (the covering vectors of the variables $w_i > 0$) and all columns $I_{:i}$, for those i satisfying (iii) (the covering vectors of the variables $v_i > 0$), form a basis of $\mathbb{R}^n$. Clearly this assumption (on the matrix M^k) is necessary in order to apply a pivot procedure (see section 2.3, and [10]).

Now let us look at the actual problem of finding a solution of (3.3.2). At the beginning of the procedure, i.e. at the current point x^k, of course A_k equals f_B per definition, hence the equation

$$M^k z^k - I w^k + I v^k = -q^k + r^k \tag{3.3.6}$$

holds, where $r^k := f_B(x^k)$, and (z^k,w^k,v^k) comprises x^k. For all indices i for which (ii) or (iii) holds, $z_i M_{:i}$ is moved to the RHS (right hand side) of (3.3.6), so that only *basic variables* (z_i with (i), w_i with (ii), and v_i with (iii)) with their covering vectors (the corresponding columns of M, or I), also called *basic columns*[3] remain on the LHS. Now (3.3.6) has the form

$$M^k_B z^k_B - I_{Bw} w^k_B + I_{Bv} v^k_B = -q^k + r^k - M^k_{nonB} z^k_{nonB}, \tag{3.3.7}$$

where the subscript B (or Bv and Bw, respectively) denotes basic columns and variables, and $nonB$ stands for nonbasic. In the case that $l_i = z_i$ (or $z_i = u_i$, respectively) and $w_i = v_i = 0$, a choice could actually be made, whether w_i (or v_i) with covering vector $I_{:i}$, or z_i with covering vector $M_{:i}$ shall enter the basis. However, in order to include as many slack (corresponding variable value equal to zero) unity matrix columns as possible, and to thereby minimize the chance of encountering rank deficiency (see section 3.3.1), in such a case $I_{:i}$ and w_i (or v_i) at value zero are included in the basis and z_i and $M_{:i}$ moved into the remainder at the RHS.

3 The terminology (*basic*) is due to the fact that these columns form a basis of $\mathbb{R}^n$, because of the assumption of nondegeneracy.

Now if one considers the subproblem of finding (z_B,w_B,v_B), such that

$$M_B^k z_B - I_{Bw} w_B + I_{Bv} v_B = -\tilde{q}, \tag{3.3.8}$$

where $\tilde{q} := q^k - r^k + M_{nonB}^k z_{nonB}^k$, then clearly (z_B^k, w_B^k, v_B^k) is a complementary basic feasible solution (BFS).

Starting from this point, the variable t, the argument of the path, will enter the basis with covering vector r^k. As in a previous section on pivot techniques in equation (2.3.8), vectors X_z^k, X_w^k, X_v^k can be uniquely determined to be the coefficient vectors in the unique expression of r^k in terms of the basic columns on the LHS in (3.3.7),

$$r^k = M_B^k X_z^k - I_{Bw} X_w^k + I_{Bv} X_v^k. \tag{3.3.9}$$

The components of these vectors will be the linear relative changes that have to be made to the coefficients on the LHS of (3.3.7) as t is increased linearly, starting from 0. Now by simply multiplying (3.3.9) by t, and adding it to (3.3.7), the pivot procedure can be assessed in terms of the equation

$$\begin{aligned} M_B^k(z_B^k - tX_z^k) - I_{Bw}(w_B^k - tX_w^k) + I_{Bv}(v_B^k - tX_v^k) + tr^k \\ = -q^k + r^k - M_{nonB}^k z_{nonB}^k. \end{aligned} \tag{3.3.10}$$

Here all coefficients on the LHS will change linearly with t, so that for small t the equal sign still holds. Hence they will form a unique path, parametrized by t, comprised by $z^k(t) := z_B^k - tX_z^k$, $w^k(t) := w_B^k - tX_w^k$, $v^k(t) := v_B^k - tX_v^k$ for the basic indices, where the nonbasic indices will remain constant. As in section 2.3.2, the coefficients on the LHS of (3.3.10) along a linear piece of this path cannot form a BFS, since there are $n+1$ columns and variables involved (now including r^k and t). As t increases, two cases can occur: Either no variable hits a bound, i.e. $v^k(t)$ and $w^k(t)$ increase forever, and the bounds of the basic components of $z^k(t)$ are $\pm\infty$, depending on the sign of each component, and they increase or decrease forever as well. This is the *ray termination* case, which will be treated later in this chapter. In the second case, at least one variable approaches its bound as t increases, until t reaches a value, say t^*, such that the first coefficient in (3.3.10), say j, hits its bound (i.e. the first index for which the case (i)-(iii) changes (see (3.3.3)-(3.3.5))). The column corresponding to the j^{th} variable that hit its bound leaves the basis (in the case of $w_i^k(t)$ and $v_i^k(t)$, disappears, in the case of $z_i^k(t)$, it changes subscripts from B to $nonB$ and moves to the RHS in (3.3.10)). Now, exactly n columns are involved in the solution (including

r^k), and hence it is a BFS[4]. As in (2.3.13), such a BFS ($z^k(t)$,$w^k(t)$,$v^k(t)$,t) of (3.3.10) is said to be *almost complementary*, if $0 < t < 1$, and *complementary*, if $t = 1$.

The goal of the procedure described here is to find a complementary BFS, i.e. a solution of (3.3.10) with $t = 1$, since at this point the r^k-terms will cancel out, and a solution of (3.2.7) has been found, hence the Newton point x_N^k solving $A_k(x) = 0$ obtained.[5] Since the constructed path is linear in t, this easily shows that it satisfies (3.2.12), the requirements of a Newton path, as claimed earlier.

In the case of an almost complementary BFS (if $0 < t < 1$), the following set of *pivot rules* determines which variable shall enter the basis next (i.e. appear on the LHS of (3.3.10) with its covering vector, and linearly be increased or decreased, beginning from its current bound):

Pivot rules:

- If v_i leaves the basis at zero, the next variable entering the basis will be z_i, starting at value u_i, its upper bound.
- If w_i leaves the basis at zero, the next variable entering the basis will be z_i, starting at value l_i, its lower bound.
- If z_i leaves the basis, two cases can occur:
  1. If z_i leaves the basis at its lower bound l_i, the next variable entering the basis will be w_i, starting at value 0.
  2. If z_i leaves the basis at its upper bound u_i, the next variable entering the basis will be v_i, starting at value 0.
- If t leaves the basis at 1, the pivot procedure stops, and the Newton point can be obtained from the variable vectors $z^k(1)$, $w^k(1)$ and $v^k(1)$ comprising it by using (3.2.5). It may then be used as the next iterate of the COMPASS algorithm x^{k+1} (see section 3.4).

The period starting with a variable entering the basis, during which the algorithm computes a linear piece of the path until another variable eventually leaves the basis, is called a *pivot step*. Clearly at the beginning of each step coefficient vectors as in (3.3.9) have to be computed in order to allow a new column to enter the basis. Each

4 Here the assumption of nondegeneracy has to be extended to include the vector r^k.

5 It is important to note that regardless of this notion of complementarity $z^k(t)$, $w^k(t)$ and $v^k(t)$ form valid triples along the whole path (i.e for each t), in the sense that they comprise a unique point $x^k(t) = p^k(t)$.

pivot step results in a new linear piece of the path, hence the hereby constructed path from x^k to x^{k+1} may have many linear pieces. The points $x^k(t)$ at which variables and columns enter or leave the basis (i.e. the corners of the path, connecting the linear pieces) are called *breakpoints*.

Clearly during the whole procedure the value of the variable t depends on the other entering variables and on the coefficient vectors, as obtained in (3.3.9), that are computed anew in each step. Hence it is possible that the value of t can also decrease during the procedure, which does not correspond to the notion of a path, that is parametrized by t. However it proved useful in terms of robustness, if the value of t is allowed to cycle [12]. In order to obtain a properly parametrized path in the end of the procedure, a subsequence of the path is selected, over which the value of t is monotonically increasing. It is this path that will be searched in the case that the returned point fails to yield satisfying descent [16].

In the following subsections, some details regarding undesirable termination possibilities or implementation difficulties are explained. The main problems that arise in practice are a rank deficient basis (which is a tough assumption in the theory of the algorithm), termination of the pivot procedure in a ray, or cycling. Treatments of these difficulties have been implemented in PATH in 1998, as originally described in [13], [16] and [24].

3.3.1 Rank Deficiency

If the basis corresponding to the triple (z^k,w^k,v^k) at the current point x^k is rank deficient, the described method fails to construct the desired path, because the assumption of having an invertible basis is necessary in the above theory. In practice, this may not only happen at the beginning of the pivot procedure (i.e. at current point x^k), but also in the middle of the path (e.g. in the case when two variables hit their boundaries simultaneously by coincidence) at a triple $(z^k(t),w^k(t),v^k(t))$, for $0 \leq t \leq 1$. Clearly, since the covering vectors of w and v are linearly independent columns of unity matrices, the problem that causes rank deficiency usually lies in the submatrix of M corresponding to the indices for which (i) holds, see (3.3.3). In such a case COMPASS chooses a new set of linearly independent columns, i.e. a new basis, which is still consistent with the same current iterate x^k, the starting point of the pivot procedure, or with the current triple $(z^k(t),w^k(t),v^k(t))$, respectively. This is done with the help of *artificial variables*, and slightly differs whether a basis in the middle of the pivot procedure, or the initial basis is concerned:

In each step of the pivot procedure, the inverse of the basis matrix is needed in order to compute the coefficient vectors as in (3.3.9). In contrast to the PATH algorithm, COMPASS inverts the basis in the beginning of the pivot algorithm, and then updates the inverse of the basis in each step according to a simple formula, see (4.2.3). Hence, each pivot step is only tentatively taken in COMPASS first, and it is checked, whether or not it is possible to update the inverse of the basis accordingly. If it is not possible, the resulting basis after the step would be rank deficient, i.e. not invertible, and the pivot procedure would have to stop. In this case, the tentative pivot step is not taken, and an artificial column is inserted into the basis and replaces a z-kind column (i.e. a column of the relevant submatrix of M), which is moved to the RHS of (3.3.10): Among all indices of kind (i) in the basis matrix, COMPASS determines an index with a replaceable linearly dependent column of the corresponding submatrix of M. There are some freedoms in choosing which columns should be made nonbasic. The choice is made according to a simple ratio check, which compares the candidates for leaving the submatrix of M with respect to stability of the update of the basis' inverse (see section 4.2.3).

If $M_{:j}$ is determined to leave the basis, then together with it, its variable z_j is made nonbasic (moved to RHS of (3.3.10)) before going on with the pivot procedure. This happens even though z_j is not at a bound; the artificial variable a_j enters instead (also at value zero) with covering vector $I_{:j}$. (3.3.10) is hence modified to become

$$M_B^k z_B^k(t) - I_{Bw} w_B^k(t) + I_{Bv} v_B^k(t) + Ia(t) + tr^k = -q^k + r^k - M_{nonB}^k z_{nonB}^k, \quad (3.3.11)$$

where the artificial variable vector a is initially zero. Only the components of a corresponding to indices of those z variables that are nonbasic but not at a bound and were moved to the RHS (denoted a_j above) will increase with t as the pivot procedure continues.

Artificial variables are treated just as w or v variables in COMPASS. If an artificial variable leaves the basis, the entering z variable is not at its bound, hence it can be increased or decreased in this case. That decision is made as to increase the value of t. If a choice of a leaving variable can be made, artificial variables are prioritized to z, w and v variables.

In this way rank deficiency is dealt with in an optimal way: In each pivot step, as many slack columns (unity matrix columns corresponding to w and v variables at value zero) as possible are made basic, and as few artificial variables as possible are introduced (only in case (i)). Obviously, this procedure to overcome rank deficiency guarantees to construct a new invertible basis, since the entering unity matrix columns

make the columns on the LHS of (3.3.10) linearly independent, and after a finite number of steps, the result would be a diagonal matrix with only $+1$ or -1 as entries. Since these artificial columns enter with an artificial variable at value zero, and since no columns of the original basis are lost in the equation (3.3.10), but merely moved to the RHS, this basis (as the whole equation) still corresponds to the current triple $(z^k(t),w^k(t),v^k(t))$, so the path is not abandoned, altered, or continued differently.

In case the initial basis is rank deficient, a similar method is applied in COMPASS. The problem of solving (3.3.2) is augmented by a vector of artificial variables, a, and transformed into that of finding a valid triple (z,w,v) such that

$$M^k z - Iw + Iv + Ia = -q^k, \tag{3.3.12}$$

where initially $a \equiv 0$, just as above. There is no difference in the quality of artifiacial variables used in the middle or in the beginning of the pivot procedure. However, in the beginning the problem lies in the choice which column of M to replace, in order to be able compute an inverse of the basis:

The procedure starts with two unity matrices, one of which will evolve to be the basis (basis-unity matrix), and the other its inverse (inverse-unity matrix). One column of the relevant submatrix of M at a time is tried to be exchanged with the respective column in the basis-unity matrix, and the inverse-unitiy matrix is tried to be updated accordingly. In this way, the same ratio check as above can be applied. The procedure is iterated until no more columns of M can be moved into the hereby generated new basis matrix so that the corresponing inverse can still be updated accordingly. These new matrices are then used in the beginning of the pivot procedure as the new invertible basis and its inverse. Any z-variables whose covering vectors were initially in the basis, but now are not, are replaced by artificial variables as described above. Since these artificial variables enter at value zero, and since no columns of the original basis are lost in (3.3.2), but merely moved to the RHS, this basis still corresponds to the current point x^k, which remains to be the starting point of the pivot procedure.

More details on the technical procedures can be found in section 4.2.3 and e.g. in [16].

3.3.2 Ray Termination

The case of ray termination is not necessarily a bad outcome of the pivot algorithm, as may be assumed on first sight. The path is uniquely determined by the initial

condition (3.3.6) (i.e. by the starting point x^k) and the pivot rules, and hence the event of ray termination is a successful termination of the procedure. It states that the linear MCP cannot be solved while the procedure starts at x^k. Nevertheless, this does not mean, that a solution of $A_k(x) = 0$ does not exist: The procedure could be started at a point different from x^k, which might cause the algorithm to terminate in a solution of the liner MCP.

However, in this case, along the ray, in which the procedure terminated, clearly $(z^k(t),w^k(t),v^k(t))$ comprises a unique point in $\mathbb{R}^n$ for each t. A point on this ray is chosen very close to the base of the ray, the last breakpoint of the path, to be the candidate for the next iterate x^{k+1} in COMPASS, as in the PATH algorithm, see [16].

Along this ray, the value of t will clearly decrease linearly, since it would hit its bound otherwise. Hence if t is not positive at the base of the ray, choosing any point along the ray to be a candidate for a new iterate does not make sense. In this case, the procedure of constructing a path starting from x^k to solve the linear MCP was unsuccessful, (e.g. terminates in a ray with negative t value), a projected gradient step will be taken from x^k in order to determine the next iterate, see also section 3.4. An alternative measure, as implemented in PATH, is to retry to solve the linear MCP one more time, using a *Lemke start*. A Lemke start, or *ray start*, starts the problem as described in a previous section, see (2.3.15), along a ray, at which all w and v variables are positive, and all z variables are at a bound. This is only possible for z variables that are bounded at least in one direction. Free z variables have to be in the basis, so if the columns of M corresponding to a set of free variables are not linearly independent, a Lemke start is not possible.

Here no artificial variables are introduced as in the last section, but all z variables are set equal to one of their bounds (preferably the upper bound), and only slack columns, $I_{:j}$ columns corresponding to w_j or v_j variables at value zero, are made basic. This clearly means that the current point x^k is abandoned, and the path is started from a new point. If also such a Lemke start yields no solution of the linear MCP, a projected gradient step is taken.

3.3.3 Cycling

A more inconvenient outcome of the pivot algorithm is the case in which the algorithm cycles. Cycling is the term used for the event in which some set of variables and columns repeatedly enters and leaves the basis in a certain order at the same variable values. For example, the variable z_s enters the basis at value u_s for the first time and

replaces v_s, which was initially basic. As z_s decreases linearly, another variable hits its bound, say w_r, hence z_r enters next at value l_r. As z_r increases, the next variable hitting its bound can again be z_s, hitting u_s, hence v_s enters next. If in the next step v_s again leaves the basis, for z_s, the algorithm cycles. Unfortunately, if the path is started from the current iterate x^k, this can not be technically avoided, due to the uniqueness of the constructed path. If a Lemke start is used, this problem does not arise, since then the algorithm is known not to cycle (see Theorem 2, and [10]), however in this case the constructed path would not start at x^k, which is in general not desired. Clearly, constructing the path with a Lemke start will require many more pivot steps in general, because it does not use the already acquired information about closeness to a solution, which is contained in the basis at which the pivot procedure starts[6].

A second difficulty that arises is to detect whether cycling really occurs or not (at least it is very costly in terms of storing capacities). COMPASS deals with this problem by means of the following heuristic (see also [16]). It counts the times that each variable enters and leaves the basis. If one variable enters the basis more than $\bar{m}$ times, the algorithm assumes cycling and stops the pivot procedure. The point $(z^k(t),w^k(t),v^k(t))$ with the highest value of t along the piece of the path that cycles is then taken to be the candidate for the new iterate x^{k+1}.

Whether the candidate for the next iterate in COMPASS obtained by this pivot technique is accepted, or the path is searched for another point yielding satisfactory descent, depends on the watchdog technique as discussed in section 3.4.

3.4 The Non-Monotone Stabilization Scheme

The purpose of this section is to give a detailed description of how the main COMPASS algorithm works. In the course of this procedure, the *nonmonotone stabilization scheme (NMS)* will be lined out, a special technique that was already contained in the first version of PATH, in order to increase speed and stability and hence efficiency of the optimization procedure. This part of the algorithm is also referred to as the *outer algorithm*, or the *major iteration* algorithm or loop, whereas the pivot procedure is referred to as the *inner algorithm*, or the *minor iteration* loop.

The main characteristics of nonmonotone stabilization are that the monotone descent requirement on the merit function is relaxed, and secondly, that not each step

6 The pivot procedure starts at the point obtained by the crash technique at the beginning of the algorithm, and then consecutively uses the basis at which the previous step terminated.

taken by the algorithm is automatically checked for satisfying the descent criteria. In the following, a *step* will always denote motion from x^k to x^{k+1} in the main algorithm; the point x_N^k will denote the unique point returned by the pivot procedure described in section 3.3 and be referred to as Newton point, regardless of whether it is really the Newton point in the classical sense (the point x at which $A_k(x) = 0$) or not[7]. The conditions under which x_N^k is accepted as the new iterate, and what happens if it is not, are some of the topics of this section.

The steps taken in the COMPASS algorithm, can be categorized into *d-steps*, *m-steps*, *watchdog steps* and *projected gradient steps*. The kind that will typically occur most of the time is a d-step; it requires the Newton point x_N^k, to be close enough to x^k; it must satisfy (in z-terms, where z is the projection of x onto the box B),

$$||z_N^k - z^k|| \leq \Delta, \tag{3.4.1}$$

a *distance criterion* (see [24]). If it is satisfied, the descent criterion is not checked, still the merit function value is not allowed to increase too much. The measure of closeness, Δ, is decreased with each step of the algorithm. After at most $\bar{n}$ consecutive d-steps, or if the returned point x_N^k is not close enough to the previous point x^k, an m-step is taken. Hence, the purpose of d-steps is fast progress, so no time is lost with unnecessary function and gradient evaluations.

In an m-step the returned point is always checked whether or not it yields satisfying descent in the merit function. The descent criterion in COMPASS, as in the current implementation of PATH [24], is given as

$$\Psi(z_N^k) \leq \begin{cases} R - \sigma\nabla\Psi(z^k)^T(z^k - z_N^k), \\ \qquad if \quad \nabla\Psi(z^k)^T(z^k - z_N^k) < 0, \\ (1-\sigma)R, \quad otherwise \end{cases} \tag{3.4.2}$$

If it is satisfied, x_N^k is accepted as the new iterate x^{k+1}, and saved as a *checkpoint*. In this case the proceeding iterate, x^{k+2}, is also saved. The *reference value* R decreases with k, in order to force the merit function Ψ to decrease. Typically, R is taken to be the maximum over those values that Ψ attained at the last 5 checkpoints, and is

7 In the case of cycling or ray termination, the path is truncated at T_k, the maximum value of t, and $x_N^k := x(T_k)$ (see also sections 3.3.3 and 3.3.2)

updated whenever a new checkpoint is saved. The purpose of m-steps is hence to eventually guarantee descent of the algorithm, even though it does not have to be monotonic descent in each step.

In the COMPASS algorithm, since the merit function is by default also evaluated during d-steps, points returned by d-steps are also saved as checkpoints, if their merit function value is explicitly lower than the reference value[8].

Watchdog techniques appear in a number of optimization techniques. Their purpose is to increase speed by not checking each iterate for descent. In COMPASS, as in PATH, such a watchdog technique is implemented; it consists of d-steps, where the distance criterion (3.4.2) is not checked, m-steps, controlling the descent behaviour of the merit function along the sequence of iterates from time to time, and watchdog steps, that mark violation of necessary criteria, see [12], and bring the algorithm back on track.

A watchdog step is taken, if the descent criterion(3.4.2) is violated in an m-step, or if the merit function value increases too much during a d-step [24]. The algorithm is taken back to the last checkpoint, $z^{\bar{k}}$ (or $x^{\bar{k}}$), and a search for a point satisfying (3.4.2) is initiated: According to [24], $\alpha \in (\bar{\alpha},1)$, where $\bar{\alpha}$ is a minimum step size, is chosen such that the new iterate

$$z^{\bar{k}}(\alpha) := \begin{cases} (1-\alpha)z^{\bar{k}} + \alpha z^{\bar{k}+1}, & \text{line search} \\ \pi_B[(1-\alpha)x^{\bar{k}} + \alpha x^{\bar{k}+1}], & \text{arc search} \end{cases} \tag{3.4.3}$$

satisfies the descent criterion (3.4.2). Here the user can select whether a *line search* (used by default) or an *arc search* is used. The line search interpolates between $z^{\bar{k}}$ and $z^{\bar{k}+1}$, and searches the line segment between these projected points, while during the arc search the projection (on B) of the line connecting the two iterates $x^{\bar{k}}$ and $x^{\bar{k}+1}$ is searched [24]. The returned point, satisfying (3.4.2) is then saved as a new checkpoint, and the algorithm continues (with a d-step, if possible).

The watchdog technique, by ignoring possible small violations of (3.4.2) in d-steps, and the nonmonotone stabilization scheme, by allowing non-monotonic descent of the merit function when accepting new iterates in m-steps as well as during a pathsearch, reduce the number of necessary pathsearches. These pathsearches are a comparatively very time consuming part in the COMPASS algorithm.

8 The implementation of this simple check is not explicitly mentioned in the literature on the current version of PATH.

A projected gradient step is taken in COMPASS, if $\alpha \leq \bar{\alpha}$ in (3.4.3), i.e. if the step size is too small, but also in other cases, e.g. in case that the linear MCP results in ray termination with negative t value. In this case, the algorithm is moved back to the *best point*, z^b, the check point with lowest merit function value. At this point, a monotone projected gradient step is taken with the merit function Ψ:

A step size $\alpha \in (\bar{\alpha},1)$ is chosen such that

$$\Psi(z^b(\alpha)) \leq \begin{cases} \Psi(z^b) - \sigma\nabla\Psi(z^b)^T(z^b - z^b(\alpha)), \\ \qquad if \quad \nabla\Psi(z^b)^T(z^b - z^b(\alpha)) < 0, \\ (1-\sigma\alpha)\Psi(z^b), \quad \text{otherwise} \end{cases} \tag{3.4.4}$$

holds for the new iterate $z^b(\alpha)$, which is determined as

$$z^b(\alpha) = \begin{cases} (1-\alpha)z^b + \alpha\pi_B[x^b - \nabla\Psi x^b], \\ \qquad \text{with a line search, and} \\ \pi_B[(1-\alpha)x^b + \alpha(x^b - \nabla\Psi x^b)], \\ \qquad \text{with an arc search.} \end{cases}, \tag{3.4.5}$$

as documented in [24]. An arc search is the default search type implemented in PATH when using this kind of step. However, in the implementation of COMPASS, the searchtype is changed if no satisfying point could be found with the initial search type.

Projected gradient steps have been included in PATH in 1999, during the development of a general feasible algorithmic descent framework by Micheal C. Ferris, Christian Kanzow and Todd S. Munson [15]. PATH was enriched by this framework, in order to improve speed and robustness of the algorithm, and so the entire descent framework was also implemented in COMPASS. Among other things, the smooth merit function Ψ was developed and with it projected gradient steps were included in the code in case of violation of the minimum step size $\bar{\alpha}$. Ψ, as defined in (2.6.7), replaced the norm of the normal map, that was used as merit function until then.

Another feature that was implemented in PATH by that time were *restarts*. If no progress is being made in the code, the solver is restarted from the user supplied starting point, using a different set of parameters governing the algorithm. Some of the parameters that are changed during restarts are the minimum step size $\bar{\alpha}$, the size of the proximal perturbation parameter ε, whether or not the crash technique is carried out in the beginning, the parameter values used in the crash technique (see

(2.5)), and the choice of line or arc searches in (3.4.3) and (3.4.5). More details on restarts and parameter values can be obtained in [16] and [15].

In COMPASS restarts are also used in these cases. Additionally, and in contrast to PATH, if the step size becomes too small in a projected gradient step, COMPASS is restarted with a point that is close to the last bestpoint (the checkpoint with the best Ψ-value), and lies on the arc that was searched in the last projected gradient step. This is a heuristic that is similar to that used in version 3.0 of PATH [16]. Unfortunately current literature on PATH in its latest version ([24] or [15]) is silent on what the algorithm does in this case. This is the reason why a lot more restarts are usually used in COMPASS than in PATH. However, also since these kinds of restarts lead to only a slight increase of the merit function value, they are of a different quality than those used in PATH.

If the last merit function values just before a restart are of the same order of magnitude in the last 3 consecutive restarts (i.e. in case overall progress is too slow), then the algorithm assumes to be in an area of a local minimizer of the merit function, and takes a new restart beginning at a random initial point, hopefully moving away from the problematic area. Also this is a heuristic similar as in version 3.0 of PATH, before the general descent framework was included.

The enhancements made to PATH in 1998 [16] also replaced the so called *pathsearch damping*, a procedure of reconstructing the path obtained by the pivot procedure between the last checkpoint and the next Newton point and searching it for an acceptable new iterate [12]. Similar to the backward and a forward pathsearch procedures as proposed by Ralph [29], a so called *backtracking* pathsearch procedure was initially imposed in the PATH algorithm. This backtracking procedure required as much computational effort as constructing the path with the pivotal procedure itself. The line or arc search procedure, see (3.4.3), that is now implemented by default in PATH and in COMPASS, is faster, more stable, removes possible rounding errors, and checks a more varied sequence of breakpoints than the backtacking pathsearch, see [16].

3.5 A Global Convergence Result

As regarding theoretical convergence properties of the algorithm, Steven P. Dirkse and Michael C. Ferris provided a global convergence result in their work of 1994 [12], when presenting their solver PATH for the first time. Their work generalized the work of Ralph [29], which guaranteed existence of the paths used in their algorithm. It was

shown that this version of PATH was globally convergent at least at a Q-superlinear rate.

The concrete result presented in their work [12] is given in the following theorem.

Theorem 7. *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be continuous, $\alpha_0 > 0$ and $X_0 := \{x \in \mathbb{R}^n | \|f_B(x)\| \leq \alpha_0\}$. Let $\sigma \in (0,1)$ (relaxation parameter in descent criterion, see (3.2.14)), $\beta \in (0,1)$ (reduction factor for d-step distance criterion), $\bar{\Delta} > 0$ (initial distance in d-step criterion), and $\bar{m}$ (memory size for reference value), $\bar{n}$ (max number of consecutive d-steps) positive integers, be the parameters governing the pathsearch, and let $X_{\bar{\Delta}\bar{n}} := X_0 + \bar{\Delta}\bar{n}B$. Suppose the following assumptions hold.*

- *A is a uniform first order approximation of f on X_0.*
- *A_k, the point based first order approximation of f_B at $\pi(x^k)$, is uniformly Lipschitzianly invertible near each $x \in X_{\bar{\Delta}\bar{n}}$; i.e. for some δ, ε and $L > 0$ and for each $x \in X_{\bar{\Delta}\bar{n}}$, there exist sets U_x and V_x containing $x + \delta B$ and $f_B(x) + \varepsilon B$ respectively, such that $A_k|_{U_x} : U_x \to V_x$ is Lipschitzianly invertible of modulus L.*
- *For each $x \in X_{\bar{\Delta}\bar{n}}$ and $T \in (0,1]$, if $p : [0,T) \to \mathbb{R}^n$ is such that $p(0) = x$ and, for each $t \in [0,T)$, $A_k(p(t)) = (1-t)f_B(x)$ and A_k is continuously invertible near $p(t)$, then there exists $p(T) := \lim_{t \nearrow T} p(t)$ with $A_k(p(T)) = (1-T)f_B(x)$.*

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

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

However, PATH was enhanced by the general descent framework in 1999. Projected gradient steps, the smooth merit function Ψ, simpler pathsearches and the restart option have been adapted. It was shown in the corresponding paper by Michael C. Ferris, Christian Kanzow and Todd S. Munson [15] (in Theorem 4 and Theorem 5), that every accumulation point of the sequence $\{x^k\}$ that this general descent framework generates is at least a stationary point of the constrained reformulation of the MCP (2.6.8), see also section 2.6, and that if the accumulation point of this sequence is a strong regular solution of the MCP, the sequence $\{x^k\}$ converges to it at a Q-superlinear rate.

In practical issues this general descent framework has improved performance of the PATH algorithm significantly (see [15]).

3.6 An Implementation of COMPASS in MATLAB/Octave

In the course of this thesis I have implemented a first version of COMPASS in MATLAB/Octave. COMPASS is free software, and the code can be obtained by the user or interested reader at the homepage of Professor Arnold Neumaier at http://www.mat.univie.ac.at/~neum/software/compass/COMPASS.html or by contacting me via schmelzer@ihs.ac.at. In this section the two most important files of the current implementation of the COMPASS algorithm are given. With the help of these two files, the user can specify nearly all settings and options governing the algorithm, and solve her own problems.

The first file is the driver file. A simple MCP is given as default. Necessary inputs are the problem dimension, the bounds l, and u of the box B, the function f, and its Jacobian matrix Df. The maximum number of restarts, the maximum number of major and minor iterations, the precision until which COMPASS should calculate the solution (target), as well as the amount of information about the algorithms numerical progress (display depth) can be defined by the user within the driver file. In order to run COMPASS it suffices to specify the problem and simply call the driver file in MATLAB/Octave.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% COMPASS_driver: calls the COMPASS routine to solve a specified MCP.
% Copyright (C) 2012 Stefan Schmelzer
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This file is part of COMPASS.

% COMPASS is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.

% COMPASS is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
```

```
% GNU General Public License for more details.

% You should have received a copy of the GNU General Public License
% along with COMPASS.  If not, see <http://www.gnu.org/licenses/>.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%
% Comments on this file:

% This file illustrates the use of COMPASSmain.m by showing how to solve
% a mixed complementarity problem (MCP):

% Given  f: R^n -> R^n, and (possibly infinite) bounds l,u
% find   z in [l,u], w,v in R_+^n
% s.t.:  f(z) = w-v
%        <(z-l),w> = 0
%        <(u-z),v> = 0.

% The problem can be defined by the user, main options can be set, and
% the COMPASS solver is called:

% The MCP is reformulated in terms of the normal equation, a pointwise
% general first order approximation (at the current iterate) of which is
% recast as a linear MCP and tried to be solved in a pivot procedure.
% The point returned by this pivot procedure is taken to be the next
% iterate.
% A non-monototne stabilization scheme in combination with a watchdog
% technique is used in the outer algorithm so as to minimize a smooth
% merit function, the zero of which is known to be equal to a solution
% of the MCP.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


clear;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% create problem/ problem definition:

n=10;                    % problem dimension
x0= unifrnd(-1,2,1,n);   % initial value
l = unifrnd(-1,1,1,n);   % upper and lower bounds
u = l + rand(1,n);

% There are two ways to create the objective function f(x) and Jacobian
% Df(x) (which is a necessary input from the user):

% 1. The expression of the objective function can be stated in the file
% objfcn.m, that of the Jacobian in the file Jacobian_objfcn.m.

% f = @objfcn;
% Df = @Jacobian_objfcn;

% 2. The objectivefunction and Jacobian can be defined here directly, by
% typing the expressions after f=@(x) and Df=@(x), respectively.

% f=@(x)   <your expression here> ;
% Df=@(x)  <your expression here> ;

% To solve your own problem, simply change the bounds and function
% expressions above. Be sure to have '%' signs in front of lines that
% should NOT be read by octave/MATLAB.


% Example: a randomly generated general quadratic function:
 DM= diag(unifrnd(-1,1,1,n));
 A = unifrnd(-1,1,n);
 b = unifrnd(-1,1,1,n);
 f=@(x) (DM*x'.^2 + A*x' + b')';
 Df=@(x) 2.*diag(DM*x') + A;
```

```
%%%%%%%%%%%%%%%%%%%%
% main options:

max_major=15*n;        % restart the algorithm after max_majoriter
                       % major iterations.
max_minor=15*n;        % stop the pivot procedure after
                       % max_minoriter minor iterations.
max_restarts=20;       % stop COMPASS after min12 to max60 restarts
                       % without a solution.
target=1.e-8;          % quit when a point with merit function value
                       % <= target is found
display_depth = 0;     % display details on solving process:
   % 0 -solution only
   % 1 -major iteration info
   % 2 -major and minor iteration info


%%%%%%%%%%%%%%%%%%%%
% call solver:
disp(' ');
disp('         solving by COMPASS ...');disp(' ');
tic;

COMPASSmain

disp(' ');
disp('          ... solved by COMPASS:');disp(' ');
duration_in_seconds = toc


%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% display output:
if Psi_value < target
  disp(' ');
  disp('     l         z_k         u          w_k        v_k')
  Solution = [l' z_k' u' w_k' v_k']
```

```
  disp(' ');
  disp('call, major_iterations, av.minor_iterations, last_Psi')
  display_statistics = [(1:restarts+1)' tot_major' save_avg_minor' save_Psi']
else
  disp('no solution found');
  disp(' ');
  disp('     l         z_k         u          w_k        v_k')
  last_points = [l' z_k' u' w_k' v_k']
  disp(' ');
  disp('display_statistics:');
  disp('call, major_iterations, av.minor_iterations, last_Psi')
  [(1:restarts+1)' tot_major' save_avg_minor' save_Psi']
end
Psi_value
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ');
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ');
disp('%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% ');
disp(' ');
```

The second file that might be necessary for the user is called *COMPASSmain.m*. Any specific settings governing the algorithm can be specified here. The crash technique *crash.m*, the core of the path solver, the file *majoriter.m*, consisting of the outer (the major iteration, with the non monotone stabilization scheme and the general descent framework) and the inner algorithm (the minor iteration, which consists of the pivot routine), and the restart routine *restartCPMPASS.m* are called from within this file.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% COMPASSmain: Main file of the COMPASS routine
% Copyright (C) 2012 Stefan Schmelzer
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This file is part of COMPASS.
```

```
% COMPASS is free software: you can redistribute it and/or modify
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation, either version 3 of the License, or
% (at your option) any later version.

% COMPASS is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.

% You should have received a copy of the GNU General Public License
% along with COMPASS.  If not, see <http://www.gnu.org/licenses/>.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%
% Comments on this file:

% COMPASS is an algorithm to solve the mixed complementarity problem
% (MCP):

% Given  f: R^n -> R^n, and (possibly infinite) bounds l,u
% find   z in [l,u], w,v in R_+^n
% s.t.:  f(z) = w-v
%        <(z-l),w> = 0
%        <(u-z),v> = 0.

% A crash technique is used at the beginning of the algorithm in order
% to determine an approximation to the active set at the solution.

% Solving the normal equation, a reformulation of the MCP in terms of
% the nonsmooth normal map, as proposed by Robinson, is the core of the
% algorithm. A Newton like method is used:

% A general first order approximation to the normal map is
% computed. The problem of finding a zero of this approximation is
```

```
% recast as a linear MCP. This linear MCP is solved by a pivot
% technique similar to that of Lemke, or that described by Dantzig.
% The pivot procedure (inner, or minor algorithm) yields a piecewise
% linear path connecting the current iterate and the Newton point. Each
% pivot step results in a linear piece of the path.

% If the new iterate does not provide the necessary descent in the
% merit function, the constructed path (or the line segment connecting
% its endpoints) is searched for an appropriate point.
% A Non monotone stabilization scheme (NMS) and a watchdog techniqe are
% used in order to reduce the number of function and gradient
% evaluations necessary (outer, or major algorithm).

% This procedure is repeated until the normal equation, and hence the
% MCP, is solved.

% An example for how to use the program is in COMPASS_driver.m

% The program was written by Stefan Schmelzer (University of Vienna) as
% part of his diploma thesis. The current Version 1.0 is dated from
% April 16, 2012.

% Please inform the author at schmelzer@ihs.ac.at if you make
% serious use of this code.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%
% Options:            % (can be changed by user; some are automatically
                      % changed after a restart)

% crash technique
do_crash = 1;           % flag for running/stopping the crash procedure
min_dim_crash = 10;     % minimum problem dimension: n=10
sigma_crash = 0.05;     % minimum descent in crash technique
max_m_alpha_crash = 14; % minimum step size in crash technique is
                        % (0.5)^12
```

```
max_crash_count = 50;   % maximum number of crash iterateions
min_active_change = 1;  % minimum change in active index set
max_no_change_row = 3;  % max number of consecutive steps with
% insufficient change in active set.

% pivot technique
in_minoriter = 0;       % flag: is the algorithm in the file minoriter?
in_pivot = 0;           % flag: is the algorithm in the pivot procedure?
infty=1000;             % number used as infinity border
cyclesize = min(max(5, n/10), 20);   % size that the length of a cycle
                                     % (of pivotroutine) is assumed.
maybe_cycle = 2;        % iteration at which cycling possibility is
                        % checked.
definitive_cycling = 0.5*n;   % maximum number of the same variable
                              % entering without abort due to cycling
max_no_solution_LCP_row = 10; % maximum number of unsuccessful pivot
                              % procedures in a row before a restart.

% NMS
pathsearch_type = 1;    % chose: line(1) or arc(2)
pg_search_type = 2;     % chose: line(1) or arc(2)
sigma_NMS = 0.01;       % minimum descent in pg and watchdog searches
max_m_alpha_w = 20;     % minimum step size in watchdog searches
max_m_alpha_pg = 25;    % minimum step size in pg searches
max_d_count = 3;        % maximum number of consecutive d-steps
Delta0 = 0.5;           % measure of closeness in every first of
                        % consecutive d-steps
deltachange = 0.07;     % Delta is reduced by deltachange after
                        % consecutive d-steps

% restarts:
next_x_near = 0;        % flag to take a special x0 after restart
x_next = norm(u-l)*rand(1,n); % default special x0 after restarts
max_restarts = min(60,max(12,max_restarts));
                        % set max_restarts to min.12 and max.60 restarts.

% statistics:
```

```
restarts = 0;          % number of restarts in the algorithm
tot_major = [];        % major iterations per model run (before a
                       % restart is made)
save_avg_minor = [];   % average pivot steps per model run
save_Psi =[];          % last Psi_value before restart or total stop.


% initial settings:
x_k = x0;                              % current point
z_k = piB(x_k,l,u);                    % projection onto box B
Psi_value_0 = Psi(z_k,l,u,f,infty);  % merit fct value at x0.
Psi_value_l = Psi(l,l,u,f,infty);    % merit fct value at l.
Psi_value_u = Psi(u,l,u,f,infty);    % merit fct value at u.
Psi_value_init = Psi_value_0;
if display_depth ~= 0
  Psi_value_0
  Psi_value_init
end


%%%%%%%%%%%%%%%%%%
% call the crash technique:
crash
% Computes an approximation to the active set at the solution.
% Returns z_k, x_k, w_k and v_k
%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%
% call the core of the algorithm
majoriter
% iteratively solves MCP
%%%%%%%%%%%%%%%%%%


if restart == 1
```

```
 % The algorithm could not solve the problem with the initial settings.
 % It is restarted with different options, multiple times if necessray.

  while restart == 1

    %%%%%%%%%%%%%%%
    % call the restart routine:
    RestartCOMPASS
    % different options are defined
    % crash technique is called
    % majoriter is called.
    %%%%%%%%%%%%%%%%%%%%

  end
end
```

For a more detailed look at the code of COMPASS the interested reader is referred to the web page of Prof. Arnold Neumaier at the University of Vienna, where the code is available for free download at
http://www.mat.univie.ac.at/~neum/software/compass/COMPASS.html. It is also possible to contact the author directly via schmelzer@ihs.ac.at.

4 Documentation of Experience and Numerical Results

This chapter shall serve as a documentation of the experience that I have gained while writing the program, and will therefore be of more technical, and personal character than others. The COMPASS algorithm is the first program that I have written in my life. Naturally, I have therefore encountered many problems of low medium or high complexity, as well as gained a lot of experience in programming, and debugging. Some of these experiences are listed below, in order to help another possibly inexperienced reader to overcome similar problems, and to give the interested reader a more detailed look into some specifics of the program.

4.1 Debugging and Display of Solutions

In the beginning it was hard for me to know at which position in the code an error occurred, and how to detect it. I quickly learnt to help myself with different display statements at different positions in the code, e.g. inside if-statements, to see whether the code enters this case or not, or in loops, to see how many times the program runs through it.

It was also helpful to develop a routine that would run through a certain number of randomly generated problems, and save the initial points x_0, the box B and the function f each time. It allowed me to re-run a certain problem in which a possibly seldom appearing error would occur, and to run a certain problem that would create extraordinarily tricky errors multiple times with different display statements in order to see what exactly caused that specific problem and fix it.

Apart from that, especially at earlier stages of the development period, it was useful to create different routines that only run parts of the solver (e.g. the pivot routine for a randomly generated LCP). Some of these routines would stop at some point of the algorithm, and allow me to continue manually, in order to get a better picture of what is going on in the program. These routines were extremely important because of the *block structure* of the algorithm, which basically consecutively solves

many small subproblems in the minor and major iteration loops (e.g. crash technique, different step acceptance routines, curing rank deficiency etc.).

After the implementation was finished, it took some time to distinguish between output information necessary for the user, and such that was necessary for me in order to be made aware of malfunctioning of the program. In the final version there are three different levels of display depth that the user can chose from. Level 0 just displays the solution of the problem, and potential errors that are displayed from within MATLAB/Octave. Level 1 displays details about the major iteration loop, e.g. which kind of step was taken, or when restarts are being made. Level 2 additionally provides details about the pivot procedure, and other small routines (e.g. rank deficiencies) that might be interesting for the user. Output in level 1 and 2 is displayed simultaneously to solving the problem, and may hence serve as a log file.

However, independent of what display level was used, at the end of a run, the solution, its accuracy and the time taken by the solver, are being displayed, as well as the number of major iterations, the average number of minor iterations for each major iteration, and the last Ψ-value before each restart.

4.2 Problems encountered

4.2.1 Cycling and Violation of Bounds

At some point of the programming work, the pivot procedure was cycling with $t = 0$ all the time, and there was no progress at all. The problem was that in the first step of the procedure (in the file pivotinit.m) I allowed the value of t to be equal to zero when entering the basis.

In order for the reader to understand this issue, it is important that I explain how this pivot step works in the numerical process: All variables change linearly with the entering variable as the latter continuously changes its value, see (3.3.10), until any variable hits one of its bounds. In the program, this is implemented by a minimization procedure. The vector of weighted distances of each variable to its boundary value, weighted with the coefficient that the variable has in the representation of r, see (3.3.9), is computed and minimized.

Unfortunately, some w_i or v_i are likely to be zero in the basis, hence so were their distances to their bound, which were chosen to be the minimum distance. The pivot procedure just jumped between these zero values, never changing the value of t.

I simply excluded this case from happening by forcing the value of the entering

variable being strictly positive in all iterations from there on. This resulted in another error, which took me a while to detect. It is the error that some variables exceeded their bounds (see also section 4.2.2).

Clearly, if the value of a variable, say w_i is at its boundary and the entering variable is forced to rise, the value of w_i may be forced to fall below zero. This depends on the "direction" of the entering variable, namely on whether its value must rise (e.g. a v or w kind variable enters at value zero, or a z kind enters at its lower bound) or fall (a z kind variable enters at its upper bound), and on the coefficient that the covering vector of this w_i has in the representation of the entering covering vector, see (3.3.9). For example, if the direction is positive (entering variable must rise), and the coefficient of w_i is negative, w_i will rise, and there is no problem. If otherwise, the signs of the direction of the entering variable and the coefficient of the variable w_i (at bound) are such that the value of w_i falls, then clearly the entering variable must be prohibited from entering with a strictly positive value. Instead it enters at its boundary value, and a variable at bound whose value would exceed this bound is chosen to leave in the next step. There can be more than one variables that may leave in such a manner simultaneously. Clearly in this case t is prioritized over all other variables, as are a-kind variables over all the others but over t. In the program variables leaving in this manner are additionally remembered on a stack, and in each iteration of the same pivot loop a new variable, if possible, is chosen to leave in order to prevent cycling between them.

4.2.2 Rounding Errors

In a lot of cases, when equality of two expressions was checked in if- statements, the problem of machine precision arose. E.g. when the value of t became 1 during the pivot technique, and hence the Newton point was found, and the technique should terminate, the abort condition was formulated as

$$\text{if } t == 1 \qquad \text{abort pivot procedure;} \qquad \text{endif.} \tag{4.2.1}$$

However, rounding errors caused the value of t to be not exactly equal to 1, hence I had to adjust the abort condition with some numerical tolerance:

$$\text{if } |t-1| < 10^{-6} \qquad \text{abort pivot procedure;} \qquad \text{endif.} \tag{4.2.2}$$

Another bug related to rounding error like above had a rather curious history. It appeared somewhere in the code in a different from, but I realized that the Newton

point, returned by the pivot procedure and used in the different steps in the NMS scheme, was comprised by variables that violated their bounds.

It turned out that only variables that already entered the basis but stayed at their boundary when entering violated this same boundary in a consecutive step sometimes.

It was at that point when I developed a routine, which should check if the variables were nice, i.e. inside their boundaries. This routine simply projects the variables onto their boundaries in case of such a violation, which happened quite often due to small rounding errors in the program. These violations often resulted also in wrong signs of the directions in which the variables should move (see also the problem described at the end of section 4.2.1). The mathematical theory is precise, but the machine is not, so this should not happen theoretically, and the problem is simply cured by this projection.

This routine was programmed in a way that I could plant it wherever I wanted to in the entire code, which helped me a lot during the rest of the debugging procedure. Luckily, it was the solution to nearly all my problems, since at that time they were caused by these marginal violations of boundaries due to these and other rounding errors and limited machine precision.

4.2.3 Dealing with Rank Deficiency

Before starting the pivot procedure, and also during the pivot procedure, whenever a variable's covering vector leaves the basis, the possibility of rank deficiency of the basis matrix arises. At first I tried to deal with this problem by checking whether the basis was rank deficient or not. This was done by means of built in functions of Octave, *rcond*, giving an approximation of the inverse of the matrix' conditioning number, *rank*, determining the rank, or *det*, calculating the determinant. I then tried to reorder the basis (first r, then a then w, v, and z columns), and bring it in reduced row echelon form. Quite often, it happened that in this form, the matrix had full rank, even though, the machine had problems finding the inverse. However all of these attempts were insufficient in dealing with the problem. In addition, inverting the basis matrix in each pivot step, (the inverse is needed in (3.3.9) to calculate the coefficient vectors X_z^k, X_w^k, X_v^k), takes too much computing time. Hence I decided to try another ansatz to solve the problem.

Instead of inverting the basis matrix in each step, now the inverse matrix is updated whenever a column is exchanged in the basis, i.e. in each step when an variable update is made in the basis. An additional loop was implemented within the pivot

routine, which checks, whether or not it is possible to update the inverse of the basis. The formula how to update the inverse H of a square matrix B after column i was changed in B is given as

$$\bar{H} = H - \frac{HdH_{i:}}{1 + H_{i:}d}, \tag{4.2.3}$$

where $d = \bar{B}_{:i} - B_{:i}$ is the difference of the i-th columns of the updated and the old matrix B, and a bar over the matrices denotes the updated version.

So in order to know whether or not it is possible to update the inverse, one merely needs to check the size of the denominator in (4.2.3). In the additionally implemented loop, a tentative pivot step is taken, see section 3.3.1, and before the update is actually made the denominator is checked. If it is big enough, the step is taken, the inverse is updated according to the above formula. If it is too small, the step is not taken, and the routine curing rank deficiency is invoked instead. This routine checks which z-column should be replaced by a unity matrix column (artificial variable's covering vector, see section 3.3.1) with the same check (size of denominator), since also here, the criterion which column to replace is that the inverse of the basis matrix can be updated most safely.

Clearly, if r and other unity matrix columns (covering vectors of v or w kind variables) in the basis are linearly dependent, which may happen, this is not a cure. In this case, since a crucial assumption of the theory is violated, the pivot procedure must be stopped, the algorithm is taken back to the last bestpoint, and continues with a projected gradient step.

The above treatment only works in case rank deficiency would occur *during* the pivot procedure. However, if the initial basis is rank deficient, a different measure has to be taken, since the inverse does not exist in this case, so the above check makes no sense. In order to see which z-column should be replaced, the code starts out with replacing all z-kind columns with unity matrix columns of the respective indices, resulting in a matrix consisting of only zeros, except for the first diagonal, where entries are either 1 or -1 (covering vectors of w, v, or a-kind variables). The inverse of this matrix obviously exists and is easily determined. In an iterative fashion, it is then checked, which z-kind column might be taken back again into the basis, so that the inverse can be updated safely. This procedure is prolonged until no more updates

can be made, and finally a new regular initial basis[9] and its inverse are obtained.

4.3 Numerical Results

For purposes of receiving numerical results of the program, an MCP as given in (2.1.9) was solved, where a general quadratic function

$$f(x) = Dx^2 + Ax + b \tag{4.3.1}$$

was used. Here D is a randomly[10] generated $n \times n$ diagonal matrix (i.e. entries that are not in the diagonal are zero), A is a randomly generated $n \times n$ matrix, and b a randomly generated n-vector.

Apart from the function f, also the box B (hence the bounds l and u for the variable z), as well as the starting point x_0 were generated randomly at each iteration. As can be seen from the results, the average duration of finding a solution to a problem, as well as the success (the frequency) of finding solutions, depends on how B and f are scaled. Hence two kinds of problem classes were simulated. In the first simulation all problems were scaled to one, meaning that the box B, x_0, as well as the entries of D, A and b were selected to be in the interval $[-1,1]$. In the second simulation these values could vary considerably higher (for specific settings see table 4.3). In the following, settings and results for both simulations are displayed. The problem dimension d was iterated, starting at $d = 10$ for both simulations, up to $d = 200$ in the first, and $d = 100$ in the second simulation.

The specifications and the main options for the first simulation are given in table 4.1. The maximum number of restarts allowed were 20. This is actually quite low if one considers the solution times displayed in the results. The maximal number of iterations in the major (outer loop, non-monotone stabilization (NMS) scheme) and minor algorithm (inner loop, pivot procedure) were 150. This number is generally large enough; the reason for restarts is usually not that the algorithm exceeds these limits but no satisfying descent in the NMS-scheme, followed by an unsuccessful projected gradient step (no suiting α was found), see section 3.4 for details. The precision parameter is the target value which the merit function in COMPASS has to reach in order for the current point to be accepted as a solution.

9 This basis still corresponds to the current point x_k. Artificial variables' covering vectors were inserted instead of z-kind columns; those were moved to the RHS of the LCP formulation of the first order approximation of the normal equation, so that the actual problem remains unchanged (see also section 3.3.1).

10 In this context, randomly generated always means that uniform distribution is used.

Option	Value
max restarts	20
max major iterations	150
max minor iterations	150
precision	e^{-9}
specifications on f ($f = Dx^2 + Ax + b$)	$D_{ii} \in [-1,1]$ $A_{ij} \in [-1,1]$ $b_i \in [-1,1]$
specifications on x_0 and B	$x_0 \in [-1,2]$ $l \in [-1,1]$ $u \in [l,l+1]$

Table 4.1: General options for first simulation

Table 4.2 shows the results of the first simulation for the different problem dimensions. A total of $n = 100$ random problems were generated with the respective settings (table 4.1) and tried to be solved. The share of successfully solved problems is given under "success rate". The average time (in seconds) needed to solve a problem in the case of successful termination, and the time needed until the maximum number of restarts were reached in the unsuccessful case, are given under "average time".

The only reason for unsuccessful termination of COMPASS is that too many restarts were necessary. Increasing the number of restarts that the solver can make also increases performance. This is the explanation for the fact that in the first simulation (with maximum 20 restarts) the success rate for problems with high dimension is lower than in the second simulation (whith maximum 50 restarts allowed). For example the 4th problem in table 4.2, with dimension $d = 100$, achieves a success rate of 86 percent if 70 restarts are allowed; the average time for successful problems doubles to reach 31.79 seconds and for unsuccessful problems nearly triples

dimension	success rate	average time	
		successful	unsuccessful
10	96%	0.33s	6.67s
20	98%	0.99s	9.65s
50	84%	4.84s	18.40s
100	58%	16.44s	37.99s
200	21%	40.53s	106.25s

Table 4.2: Results of first simulation

and reaches 106.88 seconds. This implies that the number of restarts allowed improves performance in a higher degree than the problem dimension reduces success, at least in the range of magnitudes considered here. It is hence a matter of the number of restarts, i.e. computing time that the user may provide, until most problems will eventually be solved.

Because the problem considered in the second simulation is more complicated for the solver, 50 restarts were allowed, in order to increase the success rate. The details for the specifications and options of the second simulation are given in table 4.3. Entries in D (but also in A) that are greater than 1, as well as a considerably large area of definition of the MCP (the box B) make the problem steep, and hence potentially uneasy to solve. This is why a scaling factor of (only) 2 was allowed for the diagonal matrix D, a factor of 5 for A (since the problem is only linear in A) and 7 for b. The results of the second simulation are given in table 4.4. One can see from the results that the difference in the success rate is not substantially worse than in the first simulation. However there is a significant difference in the average solving time. Both effects are due to the increased number of maximal restarts allowed, which clearly increases success and solution time as documented above.

The reason why problems with a larger domain $[l,u]$ or a steeper objective function f (both is the case in the second simulation) return worse results, is because of scaling issues. As the values of the objective function may get higher, and the slope steeper, the general first order approximations and the solutions of the linear complementarity problems in the pivot technique tend to be more inaccurate in terms of representing descent in the merit function. In this case parameters in the solver should have to be

Option	**Value**
max restarts	50
max major iterations	150
max minor iterations	150
precision	e^{-9}
specifications on f	$D_{ii} \in [-2,2]$
$(f = Dx^2 + Ax + b)$	$A_{ij} \in [-5,5]$
	$b_i \in [-7,7]$
specifications on x_0	$x_0 \in [-10,20]$
and B	$l \in [-10,10]$
	$u \in [l,l+10]$

Table 4.3: General options for second simulation

dimension	success rate	average time	
		successful	unsuccessful
10	86%	0.65s	2.97s
20	95%	5.33s	39.35s
50	82%	18.91s	92.46s
100	71%	53.39s	123.41s

Table 4.4: Results of second simulation

scaled accordingly in order to adjust e.g. the step length in projected gradient steps. Analyses regarding specifications of options or parameters in order to increase success under the consideration of differently scaled types of objective functions, or different magnitudes of the area of definition were not carried out here, for they would have exceeded the scope of this thesis. However, the problems considered in the second simulation have a considerably large area of definition; regarding practical issues, e.g. in applications with economic models (see section 5.2), variables are usually scaled to take values around 1, because of similar computational difficulties that may arise.

As a final remark it is necessary to mention that the simulations in this section were not carried out under the premise of optimizing specifications of options or parameter settings governing the algorithm, with respect to performance of COMPASS, but shall merely serve as a documentation that the developed solver is able to solve Mixed Complementarity Problems. The purpose of this thesis was to describe the PATH solver and implement a first version of a free clone, COMPASS, and not providing an equivalently powerful tool.

Any options can be specified by the user in the beginning, and can automatically be changed after each restart, according to the users wishes. Any key parameters can also be changed by the user, even if this would require some knowledge about the details of COMPASS' theory and the program itself.

The optimal choice of options and parameter values will definitely change according to which kind of problem shall be solved. Surely, different options and parameter settings will yield different behaviour of the solver, and since performance does depend on the specification of the problem, it is left to the user to specify certain parameters, as well as initial and restart options that will suit her problem best.

5 Economic Models as Example for Complementarity Problems

Among the applications of MCPs in recent science, maybe the most important one is the solution of economic problems. Many (applied) scientific research institutions use computable models of the economy, that try to assess implications of political measures (e.g. tax reforms) or external shocks (e.g. a rise in oil prices) on different parts of the economy. Also environmental issues like global warming or energy and resource security can be analyzed regarding reasons and effects with certain kinds of models.

Recently, it became very popular to formulate the underlying optimization problem of these models in terms of mixed complementarity problems, a trend that was definitely encouraged by the creation of the modelling software GAMS (general algebraic modelling software) and the MCP solver PATH in the mid 1980s and 1990s. Personally, during the last three years I have been working with this kind of models at the Institute for Advanced Studies (IHS) Vienna, an applied economic research institute, which was my initial motivation for this thesis. Therefore, I want to choose the concept of a computable general equilibrium (CGE) model as an example for application of the developed solver. I will show how such a model can be formulated as an MCP, and then outline a stylized example of such a model.

5.1 Competitive Market Equilibrium as an MCP

When talking about general equilibrium as an economic concept, one has to start with Leon Walras, who influenced this term in the 1880s. The idea is that the economy being able to be in a state of equilibrium, where it can stay for a long time. According to his thoughts, supply and demand have to be equal for all traded goods in the economy, and each entrepreneur (the agents in Walras' model responsible for managing and founding businesses and firms) had to make no profit, meaning that revenues from sold products had to equal the costs of production for each producing entity. Arrow and Debreu gave a formal definition of these concepts in 1954 [1], which

will be outlined soon.

In order not to confuse the reader I will stick to the same notation throughout this chapter. Following [25], I will talk about a *closed economy with production*, meaning that there are m activities that produce n commodities in an economic region that trades only among itself. Let for

- n commodities (goods and factors) and
- m production activities in the economy,
- p be the nonnegative n-vector of prices for each commodity,
- y be the nonnegative m-vector of activity levels for each activity,
- $d(p)$ be the m-vector of net market demand functions, and
- w be the m-vector of the economy's total endowment with each commodity.

In what follows, commodities will be indexed with $i \in I = \{1, \ldots, n\}$, and production activities with $j \in J = \{1, \ldots, m\}$. According to Scarf [34], who used the formalism of Arrow and Debreu, p^* and y^* then constitute an economic equilibrium, if

$$d(p^*) = Ay^* + w \quad \text{and} \tag{5.1.1}$$

$$\sum_i p_i^* A_{ij} \leq 0 \quad \forall j, \text{ with equality if } y_j^* > 0. \tag{5.1.2}$$

Here the $n \times m$ matrix A, also called *technology matrix*, provides information on how goods are produced in the economy, from intermediate inputs, capital, and labour. Negative entries denote inputs, positive entries denote outputs. An arbitrary entry A_{ij} will be the netput in the following, denoting the profit maximizing input (or output) of commodity i in activity j's production (see also (5.1.3) below). It is important to note that this matrix is scaled so that the sum over all elements in each column and row equals 1 (i.e. entries of A are called netputs for *unit production*). In an advanced setting, this matrix may also depend on the price vector p, since producers may change their way of producing depending on prices that they have to pay.

Supply of a good is related to the profit made by the producer according to *Hotelling's Lemma*, see [21], so that for each activity j, the unit production column can be written as

$$A_{:j}(p) = \left(\frac{\partial \Pi_j(p)}{\partial p_i}\right)_{i \in I}, \tag{5.1.3}$$

where $\Pi(p)$ is the m-vector of unit profit functions, $\Pi_j(p)$ being the unit profit function for each activity j, which is simply the difference between revenue and cost at unit production. Hotelling's Lemma states that the input (or output) of a good or factor in an activity's unit production equals the partial derivative of the unit profit function with respect to the price of that good or factor.

Each of the unit profit functions Π_j is assumed to be homogeneous of degree 1 in all prices, hence by Euler's homogeneous function theorem, we get

$$\Pi_j(p) = (\nabla \Pi_j(p))^T p = A_{\cdot j}(p)^T p. \tag{5.1.4}$$

In 1985 Mathiesen gave the following definition of a competitive market equilibrium, and thereby the foundation stone to the formulation of economic problems as complementarity problems was laid (see [25]):

A *competitive equilibrium* in a closed economy is given by a price vector p^* and a vector of activity levels y^* such that:

$-A(p)^T p^* \geq 0$	There is no positive profit for any activity.	(5.1.5)
$w + A(p)y^* - d(p^*) \geq 0$	There is no excess demand of any good.	(5.1.6)
$p^* \geq 0, \quad y^* \geq 0$	Prices or activity levels are nonnegative.	(5.1.7)
$(A(p)^T p^*)^T y^* = 0$	If an activity earns a deficit, it is idle; if an activity is active, it earns zero profits.	(5.1.8)
$p^{*T}(w + A(p)y^* - d(p^*)) = 0$	The price for a commodity in excess supply is zero; any positive prices imply that supplies equal demands.	(5.1.9)

It is very important to note that this definition is already designed to fit the concept of a complementarity problem: To repeat, the nonlinear complementarity problem (NCP) is given as

$$\begin{array}{ll} \text{given} & F : \mathbb{R}^n \to \mathbb{R}^n \\ \text{find} & z \in \mathbb{R}^n_+ \\ \text{s.t.} & F(z) \geq 0 \quad \text{and} \quad z^T F(z) = 0. \end{array} \tag{5.1.10}$$

Now it is easily seen (and was one of the main results of Mathiesen in 1985, [25]) that solving the problem of equations 5.1.5 to 5.1.9 is equivalent to solving the

complementarity problem 5.1.10, where

$$z = \begin{pmatrix} y \\ p \end{pmatrix} \quad \text{and} \quad F\begin{pmatrix} y \\ p \end{pmatrix} = \begin{pmatrix} -A^T p \quad (= -\Pi(p)) \\ w + Ay - d(p) \end{pmatrix} \tag{5.1.11}$$

It was, however, not until 2008, when Christoph Böhringer and Thomas Rutherford showed [7] that complementarity, namely the last two assumptions of Mathiesens definition, (5.1.8) and (5.1.9), are actually a *characteristic* of the economic equilibrium concept, rather than a *condition*, as assumed by Mathiesen. Hence, given goods and activities as well as supply and demand functions, the problem of finding price and activity level vectors that constitute an economic equilibrium can be formulated as an MCP.

5.2 Description of a Dynamic CGE Model

Computable general equilibrium models are a class of economic models that use real economic data to estimate how an economy will change according to political measures or external shocks. The model that is characterized in this chapter is a so called Ramsey model (Ramsey model of savings and investment). In such a model, the household agent maximizes his lifetime utility, the driving force in the model, which is comprised from intertemporal consumption and leisure. The production sectors minimize production costs in each period, and produce exactly as much as is demanded by the household agent, or *representative agent*. This representative agent buys the products, and provides capital and labour as input goods for production. It is furthermore assumed that the agents (production sectors and households) in the model know exactly as much about the future as the modeller (perfect foresight).

Regarding mathematical formulation of solving CGE models, the MCP format (2.1.9) should be used rather than the NCP format, see [3]. Even if the solver uses the MCP format, in our case the variables will only be able to take nonnegative values, and the MCP reduces to the special case:

$$\begin{aligned} &\text{Given} && f : \mathbb{R}^n \to \mathbb{R}^n \text{ and } B = [0,\infty], && (5.2.1)\\ &\text{find} && z \in B, \quad w \in \mathbb{R}^n_+ && (5.2.2)\\ &\text{s.t.} && f(z) = w && (5.2.3)\\ & && z^T w = 0 && (5.2.4) \end{aligned}$$

I now want to show how the Ramsey model can be translated into the complemen-

tarity theory developed above, by lining out an intuitive formulation of a stylized CGE model in the complementarity format. Publications by Christoph Böhringer and Thomas Rutherford, see e.g. [7], are the fundamental source of references in this chapter. If not further specified, most of the technical details in this section that are about to follow are implicitly taken from their work, or from enhancements of their work carried out at the IHS. Among these enhancements are recently developed models, see [26] for detailed documentation, that were used for several national studies, e.g. [2] or [27].

As in the previous section, we assume a competitive economy with n goods (including factors), m activities, and one household. The variables can be classified by the following categories (see e.g. [25]):

- p a nonnegative price vector as in section 5.1
- Y a nonnegative vector of activity levels as in section 5.1
- M a nonnegative vector of income of the agents (in our case there is only one agent)

Now a competitive market equilibrium is characterized by these variables if the conditions from Mathiesen's definition above hold. In the next sections the function f, representing the complementarity conditions in (5.1.11), will be specified for each of these kinds of variables. All equations presented hold in each time period, therefore due to simplicity time dependence of the variables is omitted in the remainder of this chapter; the dynamic framework is explained in a separate section below.

5.2.1 Zero Profit Conditions

The Zero Profit Conditions imply that no production activity makes a positive profit. The intuitive background for this assumption is the economic notion of *perfect competition*: This condition is clearly motivated by the underlying assumption of perfect competition; if there were a firm that would be making profit, then another firm could produce the same good at the same production costs, but sell them cheaper for less profit. In the limit, this reduces all firms in a specific sector to sell their goods at the same price, namely at the production price. The zero profit condition can be stated as

$$-\Pi_j(p) = c_j(p) - r_j(p) \geq 0, \tag{5.2.5}$$

where $\Pi_j(.)$ denotes the unit profit function of the production activity j, revenues r_j minus costs c_j for a produced unit of output, see [7]. The costs for production are comprised by the market prices of the input goods. This is typically implemented by the means of CES-functions (constant elasticity of substitution). According to changes in the prices of input goods, sectors in the model can substitute between the input goods up to a certain degree, depending on elasticities that need to be specified. An example for a CES-function would be

$$ces[p_1,p_2] = (\theta_1 p_1 + \theta_2 p_2)^\sigma, \tag{5.2.6}$$

where in this case $ces[p_1,p_2]$ is the cost function of a production activity that needs input goods of kind $i = 1,2$, and can substitute with an elasticity σ between them. θ_1 and θ_2 denote the shares of the input goods in the first period of the model.

The complementarity variables for the zero profit functions are the activity levels. According to (5.2.5) the complementarity conditions read as follows:

$$Y_j \geq 0 \quad \perp \quad c_j(p) - r_j(p) \geq 0, \quad \text{or equivalently} \tag{5.2.7}$$

$$Y^T(c(p) - r(p)) = 0, \tag{5.2.8}$$

establishing the connection to (5.2.4), with $f_J(p,Y) = (c(p) - r(p))$. Since the zero profit condition is formulated in a way that it is more than likely that $\Pi(p) = 0$, the activity levels will be positive throughout the solution procedure.

5.2.2 Market Clearance Conditions

The market clearance conditions state that all markets are cleared, or in other words that supply minus demand is nonnegative for all goods and factors in the economy, see [7]:

$$\sum_j Y_j \frac{\partial \Pi_j(p)}{\partial p_i} + w_i \geq d_i(p,M) \quad \forall i, \qquad \text{where:} \tag{5.2.9}$$

w_i	signifies the households initial endowment by commodity,
$\frac{\partial \Pi_j(p)}{\partial p_i}$	indicates the supply of good i per unit of operation of activity j (see Hotelling's Lemma), and
d_i	is the households utility maximizing demand for good i.

Firms produce exactly as many products as the consumers demand. No product is in excess supply, and no demand stays unfulfilled.

The complementarity variables for the market clearance conditions are the prices, and the complementarity conditions can be stated as

$$p \geq 0 \quad \perp \quad [\textstyle\sum_j Y_j \frac{\partial \Pi_j(p)}{\partial p_i} + w_i - d_i(p,M)]_{i \in I} \geq 0 \qquad \text{or equivalently} \tag{5.2.10}$$

$$p^T [\textstyle\sum_j Y_j \frac{\partial \Pi_j(p)}{\partial p_i} + w_i - d_i(p,m)]_{i \in I} = 0, \tag{5.2.11}$$

yielding the connection to (5.2.4), if $f_i(p,Y) = \sum_j Y_j \frac{\partial \Pi_j(p)}{\partial p_i} + w_i - d_i(p,M)\ \forall i \in I$.

5.2.3 Income Balance Conditions

The Income Balance Condition requires that total household expenditure must equal total household income, see [7]:

$$M := \sum_i p_i w_i. \tag{5.2.12}$$

Here w_i, the household initial endowment corresponds to its endowment with capital and time, that is provided to the activities in return for income. The household is typically not initially endowed with any other good that is produced in the economy.

It is necessary to note that there is not actually a complementarity condition at work here. The variable M denotes disposable money for consumption, hence it is simply assumed that all income is of such disposable kind. However for matters of completeness, and in order to characterize the relation to (5.2.4), the income balance may be stated in the complementarity format as

$$M = \textstyle\sum_i p_i w_i \geq 0 \quad \perp \quad M - \sum_i p_i w_i \geq 0, \quad \text{or equivalently} \tag{5.2.13}$$

$$M(M - \textstyle\sum_i p_i w_i) = 0, \tag{5.2.14}$$

where $f_M(p,Y) = M - \sum_i p_i w_i \equiv 0$ by definition.

5.2.4 The Dynamic Setting

Due to the effects of policy interference on investment and savings incentives modelling medium- to long-term development of an economy calls for an explicit intertemporal framework. The standard Ramsey model of savings and investment accounts for this intertemporal characteristic by the means of capital stock dynamics. In a deterministic setting, a logically consistent approach in dynamic modelling is the notion of perfect foresight, which states that the agents in the model know as much about the future as the modeller. In the standard Ramsey model of savings and investment this notion is coupled with the assumption of an infinitely lived representative agent who

makes explicit choices at the margin between current and future consumption, by maximizing welfare subject to an intertemporal budget constraint, see section 5.2.5.

Leaving utility from leisure out for now, the representative agent maximizes utility from consumption, $U(C(t))$ in each time period,

$$max: \quad U(C(t)) \tag{5.2.15}$$

$$st: \quad C(t) = Y(t) - I(t) \tag{5.2.16}$$

$$K(t+1) = (1-\delta)K(t) + I(t). \tag{5.2.17}$$

In both constraints the index j is implied, i.e. the identities must hold for each activity (the constraints are of dimension m). The first constraint implies that all goods are either consumed or used as investments, $I(t)$, for the next time period (machines or buildings). The second constraint describes the capital stock dynamics. An activity's capital stock K_j in the next period equals depreciated (δ) capital in the current period plus investments. The capital stock can be seen as anything that is not an intermediate input good, but is still needed by an activity for production, and has to be renewed every once in a while. The depreciation rate δ may be interpreted as the mortality rate of the aggregate capital stock. The set of equations (5.2.15)-(5.2.17) is typically referred to as the standard Ramsey model of savings and investments.

If one introduces two additional variables, the price for purchasing a unit of physical capital for the capital stock, $p^k(t)$, and the rental rate of capital $r_k(t)$ (also called return to capital, which is the actual interest that the household sees when lending its capital to the production activities as intermediate input, like labour), the Ramsey model can be integrated into the complementarity framework by adding the additional complementarity conditions

$$p^k(t) \geq 0 \quad \perp \quad (1-\delta)K(t) + I(t) - K(t+1) \geq 0, \text{ and} \tag{5.2.18}$$

$$r_k(t) \geq 0 \quad \perp \quad (r+\delta)K(t) - Y(t)\frac{\partial \Pi}{\partial p^k}(t) \geq 0, \tag{5.2.19}$$

see [7]. Here $p^k(t)$ is an m-vector, and the price for purchasing a unit of capital may change for each production activity, while the rental price r_k is the same for each activity. For matters of simplicity the index j is omitted in most of the equations in this section wherever the meaning is clear. Equations (5.2.18) and (5.2.19) can easily be transformed to fit the format of (5.2.4), as was done in the previous sections. Both of these conditions relate the price for a good to the demand of the good, in this case purchased or rented capital. If intertemporal demand equals supply of the capital stock, there is a positive price (5.2.18). If the lost opportunity costs (of

lending the capital stock to another agent, $(r+\delta)K(t)$, where r is the fixed interest rate), equal demand for capital in production, the return to capital $r_k(t)$ is positive (5.2.19)[11]. Hence these equations are easily detected to be additional market clearance conditions.

The complementarity conditions for the two new variables $I(t)$ and $K(t)$ relate to their prices, see [7] or [26]:

$$I(t) \geq 0 \quad \perp \quad [\sum_i p_i(t)a_i(t)] - p^k(t) \geq 0, \text{ and} \tag{5.2.20}$$

$$K(t) \geq 0 \quad \perp \quad p^k(t) - r_k(t) - (1-\delta)p^k(t+1) \geq 0. \tag{5.2.21}$$

Here a_i denotes the share of good i in activity j's investment bundle in (5.2.20), so investments of an activity j are positive if and only if the purchase price of capital for this activity equals the weighted price of its investment goods. The capital stock is positive if and only if the price of buying a unit of capital in one period equals the costs of lending it in this period and buying it in the next period. Here it is assumed that the capital stock is always positive, and changes only through investment levels, so equation (5.2.21) is actually a condition on the relation between r_k and p^k. Also the last two equations are easily written in the form of (5.2.4), and detected to be zero profit conditions, since they relate quantity variables to the relation between their revenues and costs.

The intertemporal setting is induced to the model by the equations (5.2.18) and (5.2.21). Clearly all variables are dependent on the additional time dimension, but the link between the periods exists in these two equations only, so it was not necessary to emphasize time dependence in the last sections.

Clearly, the concepts presented here constitute a theoretical fundament for an implementation of a stylized model, in which only a finite number of periods will be considered. Hence the capital stock dynamics must eventually come to an end in such an implementation of this theory. However since it is not the purpose of this chapter to give a full implementation of such a model, but rather to motivate it, the interested reader will have to be satisfied with the reference to the work of Thomas Rutherford [32] for further information on how to approximate an infinite horizon and include the *terminal capital stock constraint* into the MCP framework.

11 Actually equation (5.2.19) is an ordinary market clearance condition and already included in (5.2.10). Still for a better understanding of the subject it is separately listed here.

5.2.5 Intertemporal Utility Maximization

The infinitely lived representative agent maximizes total utility W over all periods. W is an intertemporal composite of consumption C and leisure LS. The problem of maximizing lifetime utility can be formulated as

$$max: \quad W = \sum_{t=0}^{T} (\frac{1}{1+\rho})^t U(CLS(t)), \tag{5.2.22}$$

$$st: \quad \sum_{t=0}^{T} p_{cls}(t) CLS(t) = p^k(0)^T K(0) + \sum_{t=0}^{T} [LSP(t) + LS(t)] p_{ls}(t) - p^k(T)^T K(T), \tag{5.2.23}$$

see [7], or [26]. Here $CLS(t)$ denotes the composite of consumption and leisure per period, LSP the labour supply (actually labour supply is not really a model variable but is the difference of total time endowment and leisure), ρ is the intertemporal utility preference parameter, p_{cls} is the CES-composite price for consumption and leisure (the price for leisure p_{ls} is the same as the price for labour, p_l, which is determined in the market clearance conditions in (5.2.10)), T is the last time period, and the the utility function U is given as

$$U(c) = \frac{c^{1-\frac{1}{\eta}}}{1-\frac{1}{\eta}}, \tag{5.2.24}$$

where η is typically greater than 1, so utility is of logarithmic growth. The constraint (5.2.23) gives the households budget constraint. In the dynamic setting it is assumed that the household uses his overall lifetime income for consumption and leisure. This lifetime income, the RHS of (5.2.23), includes the benefits from leisure and income from capital. It is assumed that the household owns the capital stock of the sectors, so income from capital lending is the same as the difference of the present value of the capital stock in the beginning and in the end. For further information the reader is referred to the work of Thomas Rutherford [32].

In order to formulate the complementarity conditions for these relations one needs additional price variables. p_C, the price for consumption, is the aggregate price of the weighted bundle of goods that the representative agent consumes, and p_w, the price of welfare, can be interpreted as the value (the price) that is assigned to one unit of intertemporal utility of consumption and leisure in monetary terms. With

these preparations, the complementarity conditions can be formulated as

$$C(t) \geq 0 \quad \perp \quad ces_c[(p_i(t))_{i\in I}] - p_c(t) = -\Pi_c(t) \geq 0, \tag{5.2.25}$$

$$CLS(t) \geq 0 \quad \perp \quad ces_{cls}[C(t),LS(t)] - p_{cls}(t) = -\Pi_{cls}(t) \geq 0, \tag{5.2.26}$$

$$LSP(t) \geq 0 \quad \perp \quad p_{ls}(t) - p_l(t) = -\Pi_{lsp}(t) \geq 0, \text{ and} \tag{5.2.27}$$

$$W \geq 0 \quad \perp \quad [\sum_t (\theta_t p_{cls}(t))^{1-\sigma}]^{\frac{1}{1-\sigma}} - p_w = -\Pi_w \geq 0, \tag{5.2.28}$$

see [26] or [7] , where $ces_{\cdot}[.]$ denotes constant elasticity of substitution functions as in (5.2.6), that resemble input cost functions of the respective goods; $\Pi_{\cdot}$ are the respective unit profit functions. θ_t is the share parameter and σ the elasticity of substitution between the prices of the consumption and leisure composites in each period. These equations relate activity levels to the differences in their prices and costs, so these can be seen as additional zero profit functions, see (5.2.7).

The complementarity conditions for the prices that relate to these equations, constitute additional market clearance conditions, since prices are related to the levels of demand and supply of the respective goods, see [26]:

$$p_c(t) \geq 0 \quad \perp \quad C(t) - CLS(t)\frac{\partial \Pi_{cls}}{\partial p_c}(t) \geq 0, \tag{5.2.29}$$

$$p_{ls}(t) \geq 0 \quad \perp \quad LS(t)] - CLS(t)\frac{\partial \Pi_{cls}}{\partial p_{ls}}(t) \geq 0, \tag{5.2.30}$$

$$p_{cls}(t) \geq 0 \quad \perp \quad CLS(t) - W\frac{\partial \Pi_w}{\partial p_{cls}}(t) \geq 0, \text{ and} \tag{5.2.31}$$

$$p_w \geq 0 \quad \perp \quad p_w W \sum_t [CLS(t)] - HH \geq 0. \tag{5.2.32}$$

Here the partial derivatives on the RHS denote the unit demand functions as in (5.2.9), e.g. $\frac{\partial \Pi_{cls}}{\partial p_c}(t)$ is the demand for consumption as a share in one unit of the consumption and leisure composite in period t. The equations state that prices are positive if and only if supply equals demand. The last equation states that total welfare (and all money spent for it) should equal overall household income from all periods, HH[12]. Hence the income balance condition (5.2.13) actually reads

$$\begin{aligned} HH \geq 0 \quad \perp \quad & p^k(0)^T K(0) + \sum_{t=0}^{T} [LSP(t) + LS(t)] p_{ls}(t) \\ & - p^k(T)^T K(T) - HH \geq 0 \end{aligned} \tag{5.2.33}$$

12 In (5.2.13) time dependence was implicitly assumed for household income, hence the variable "M" was used, in order to more easily distinguish it from the intertemporal income HH.

in the dynamic setting, see [26] or [32]. Again it is clear that there is not actually complementarity at work here, but HH is defined in this equation. The actual complementarity condition is given in (5.2.32)

The ideas in this chapter shall indicate that no matter what economic relationships one might want to incorporate into the model, as long as they can be formulated in a complementary manner it is technically relatively easy to include them. However, the concept of zero profit and market clearance conditions is crucial, since they constitute the fundament and the legitimation for formulating economic relations in this complementarity format, relating prices to excess demand, and activity levels to profit in a complementary fashion. Additions to these concepts definitely exist, for example Böhringer and Rutherford combined first order optimality conditions of a linear programming problem into the MCP format, interpreting the Lagrange multipliers as shadow price variables, see [7].

6 Conclusions

COMPASS is a globally convergent algorithm for solving the Mixed Complementarity Problem. It is published under the GNU General Public License and hence is free software. As documented it successfully solves Mixed Complementarity Problems with up to 200 variables. As a clone of PATH, however, it still suffers from poorer performance and a missing link to a modelling software as GAMS. In its current version problems will have to be formulated explicitly as an MCP by the user directly in MATLAB/Octave. An analysis regarding the specification of certain algorithm parameters and options with the objective to increase performance is subject to further research. Any devoted programmer or student is highly welcome to enhance or improve the code in any way she wants, however I would be glad to be informed about such an undertaking and offer my help as far as possible. The program is available for free download under
http://www.mat.univie.ac.at/~neum/software/compass/COMPASS.html or by contacting the author under schmelzer@ihs.ac.at.

Bibliography

[1] Arrow, K. and Debreu, G. (1954). Existence of an equilibrium for a general economy. *Econometrica*, 22:265–290.

[2] Balabanov, T., Friedl, B., Miess, M., and Schmelzer, S. (2010). *Mehr und qualitätsvollere Green Jobs (Green Jobs for a Sustainable, Low-carbon Austrian Economy).* Studie im Auftrag des Bundesministeriums für Arbeit, Soziales und Konsumentenschutz (Study Commissioned by the Austrian Ministry for Labour, Social Affairs and Consumer Protection).

[3] Billups, S. C. (1995). *Algorithms for Complementarity Problems and Generalized Equations.* PhD thesis, Universtiy of Wisconsin, Madison.

[4] Billups, S. C. (2000). Improving the robustness or descent-based methods for semismooth equations using proximal perturbation. *Mathematical Programming*, Ser. A 87:153–175.

[5] Billups, S. C., Dirkse, S. P., and Ferris, M. C. (1997). A comparison of large scale mixed complementarity problem solvers. *Computational Optimization and Applications*, 7:3–25.

[6] Brooke, A., Kendrick, D., and Meeraus, A. (1988). *GAMS: A Users Guide.* The Scientific Press, South San Francisco, CA.

[7] Böhringer, C. and Rutherford, T. (2008). Combining bottom-up and top-down. *Energy Economics, 30*, 2:574–596.

[8] Chen, C. and Mangasarian, O. L. (1996). A class of smoothing functions of nonlinear and mixed complementarity problems. *Computational Optimization and Applications*, 5:97–138.

[9] Cottle, R. (1964). *Nonlinear Programs with Positively Bounded Jacobians.* PhD thesis, Department of Mathematics, University of California Berkeley, CA.

[10] Dantzig, G. B. (2003). *The basic George B. Dantzig.* (R. W. Cottle, ed.) Stanford University Press.

[11] Dirkse, S. P. (1994). *Robust Solution of Mixed Complementarity Problems.* PhD thesis, University of Wisconsin, Madison.

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

[13] Dirkse, S. P. and Ferris, M. C. (1996). A pathsearch damped newton method for computing general equilibria. *Annals of Operations Research*, pages 211–232.

[14] Dirkse, S. P. and Ferris, M. C. (1997). Crash techniques for large scale complementarity problems. *Complementarity and Variational Problems: State of the Art*, (M. C. Ferris and J. S. Pang, eds.), SIAM, Philadelphia, Pennsylvania:40–61.

[15] Ferris, M. C., Kanzow, C., and Munson, T. S. (1999). Feasible descent algorithms for mixed complementarity problems. *Mathematical Programming*, 86:475–497.

[16] Ferris, M. C. and Munson, T. S. (1998). Interfaces to PATH 3.0: Design implementation and usage. *Computational Optimization and Applications*, 12:201 – 227.

[17] Ferris, M. C. and Munson, T. S. (2001). Preprocessing complementarity problems. *Complementarity: Applications, Algorithms and Extensions*, (M. C. Ferris, O. l. Mangasarian, and J. S. Pang, eds.), Kluwer Academic Publishers, Volume 50 of Applied Optimization:143–164.

[18] Ferris, M. C. and Pang, J. S. (1997). Engineering and economic applications of complementarity problems. *SIAM Review*, 39, No 4:699–713.

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

[20] Hartman, P. and Stampacchia, G. (1966). On some nonlinear elliptic differential functional equations. *Acta Mathematica*, 115:153–188.

[21] Hotelling, H. (1932). Edgeworth's taxation paradox and the nature of demand and supply functions. *Journal of Political Economy*, 40:577 – 616.

[22] Johansen, L. (1960). *A Multi Sectoral Study of Economic Growth.* Amsterdam: North-Holland Publishing Company.

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

[24] Li, Q. (2010). *Large-Scale Computing for Complementarity and Variational Inequalities*. PhD thesis, University of Wisconsin, Madison.

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

[26] Miess, M. (2012, forthcoming). *Computable General Equilibrium Models: Hybrid Top-Down Bottom-Up Energy Policy Modelling - Theory and Application*. Diploma thesis, University of Vienna, Austria.

[27] Miess, M., Schmelzer, S., and Schnabl, A. (2011). *Evaluierung der regionalen Beschäftigungs- und Wachstumsoffensive 2005/2006 (Evaluation of the Regional Employment and Growth Offensive)*. Studie im Auftrag des Bundesministeriums für Wirtschaft, Familie und Jugend (Study Commissioned by the Austrian Ministry of Economy, Family and Youth).

[28] Mitra-Kahn, B. H. (2008). Debunking the myths of computable general equilibrium models. *SCEPA Working Paper 2008-1*.

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

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

[31] Robinson, S. M. (1998). A reduction method for variational inequalities. *Mathematical Programming*, 80:161–169.

[32] Rutherford, T., Lau, M., and Pahlke, A. (2002). Approximating infinite-horizon models in a complementarity format: A primer in dynamic general equilibrium analysis. *Journal of Economic Dynamics and Control*, 26(4):577–609.

[33] Rutherford, T. F. (1993). MILES: A mixed inequality and nonlinear equation solver. Working Paper, Department of Economics University of Colorado, Boulder.

[34] Scarf, H. E. and Hansen, T. (1973). *The Computation of Economic Equilibria*. Yale University Press, New Haven.

[35] Tomlin, J. and Welch, J. (1986). Finding duplicate rows in a linear programming model. *Operations Research Letters*, 5(1):7–11.

[36] Walras, L. (1954). *Elements of Pure Economics.* Allen and Unwin Ltd.

List of Tables

A Appendix

A.1 Deutsche Zusammenfassung

Die vorliegende Arbeit präsentiert COMPASS, einen global konvergenten Lösungsalgorithmus für gemischte Komplementaritätsprobleme. Die zu Grunde liegende mathematische Theorie basiert auf dem PATH Solver, dem standard Lösungsalgorithmus für diese Art von Problemen. COMPASS ist unter der "GNU General Public License" Lizenz veröffentlicht und ist daher Freie Software. Das Fundament von COMPASS ist eine stabilisierte Newton Methode: Das gemischte Komplementaritätsproblem wird in der Form der Normalgleichung (normal equation) reformuliert. Eine allgemeine Approximation erster Ordnung dieser Normalgleichung kann als lineares gemischtes Komplementaritätsproblem dargestellt und mit Hilfe einer Pivot Technik gelöst werden. Diese Lösung entspricht dem Newton Punkt im standard Newton Verfahren, und wird daher hier auch so bezeichnet. In der Pivot Technik wird neben der Lösung auch ein stückweise linearer Pfad generiert, der den letzten Iterationspunkt und den Newton Punkt verbindet. Ob dieser Punkt als nächster Iterationspunkt akzeptiert wird hängt von einem nicht-monotonen Stabilisierungsverfahren ab, das eine Watchdog Technik beinhaltet. Außerdem existiert eine glatte "merit" Funktion, basierend auf einer modifizierten Fischer-Burmeister Funktion, die den Erfolg des Fortschritt misst, und bei der Lösung des Komplementaritätsproblems eine Nullstelle besitzt. Gemäß der Verbesserung des Wertes dieser merit Funktion sind gewisse nicht monotone Abstiegskriterien definiert. Diese werden jedoch nicht in jedem Schritt getestet, um die Anzahl der Funktions- und Gradientenauswertungen zu minimieren. Wenn die Lösung aus der Pivot Technik, also der Newton Punkt, diesen Abstiegskriterien genügt, wird er als neuer Iterationspunkt verwendet. Falls nicht, geht der Algorithmus zurück zum letzten "Checkpoint", dem letzten Punkt, der dem Test mit dem Abstiegskriterium erfolgreich unterzogen wurde. Der Pfad zwischen diesem Checkpoint, und dem Newton Punkt nach diesem Checkpoint (der Newton Punkt wird nach jedem Checkpoint gespeichert) wird dann nach einem die Abstiegskriterien erfüllenden Punkt durchsucht. Sollte kein passender Punkt gefunden werden, geht

der Algorithmus zurück zum “Bestpoint”, dem Punkt mit dem bisher niedrigsten Wert der merit Funktion, und macht einen projizierten Gradientenschritt.

Ein globaler Konvergenzbeweis dieser Theorie ist in der Arbeit enthalten. Der Algorithmus wurde im Zuge dieser Arbeit in MATLAB/Octave implementiert, und steht auf http://www.mat.univie.ac.at/~neum/software/compass/COMPASS.html zum Download und zur freien Benutzung zur Verfügung. Eine Simulation wurde anhand von zufällig generierten Problemen durchgeführt und dokumentiert das erfolgreiche Lösen von Problemen des Algorithmus zumindest bis zu einer Größenordnung von 200 Variablen. Eine kurze geschichtliche Einführung über den Zusammenhang zwischen gemischten Komplementaritätsproblemen und ökonomischen Modellen ist in der Arbeit enthalten, sowie eine Anleitung anhand eines Beispiels, wie solche Modelle in der Form von Komplementaritätsproblemen forumliert werden können.

A.2 Curriculum Vitae

Family name:	SCHMELZER
First name:	Stefan
Date of birth:	12-12-1983
Nationality:	Austria

Education

foreseen summer 2012	Diploma in Mathematics
03/2004 - current	University of Vienna, Mathematics
09/2002 - 09/2003	Civil Service, Arbeiter Samariter Bund Österreich
1994 - 2002	Bundesgymnasium Neusiedl/See

Present position Researcher at the Department for Economics and Finance at the Institute for Advanced Studies (since 01/2011)
Research fields General equilibrium modelling, computational economics, environmental and energy economics.
Years with firm 3

Professional Experience Record

Date	Since 01/2011
Location	Vienna
Company	Institute for Advanced Studies in Vienna
Position	Researcher

Date	03/2009 - 12/2010
Location	Vienna
Company	Institute for Advanced Studies in Vienna
Position	Student Assistant

Date	07/2008 - 02/2009
Location	Vienna
Company	Private tutoring school “Lernquadrat”
Position	Mathematics teacher

Selected Publications

- Miess, M., Schmelzer, S. and Schnabl, A. (2011). Evaluierung der regionalen Beschäftigungs- und Wachstumsoffensive 2005/2006 (Evaluation of the Regional Employment and Growth Offensive). *Studie im Auftrag des Bundesministeriums für Wirtschaft, Familie und Jugend (Study commissioned by the Austrian Ministry of Economy, Family and Youth).*

- Balabanov, T., Friedl, B., Miess, M., and Schmelzer, S. (2010). Mehr und qualitätsvollere Green Jobs (Green Jobs for a Sustainable, Low-carbon Austrian Economy). *Studie im Auftrag des Bundesministeriums für Arbeit, Soziales und Konsumentenschutz (Study Commissioned by the Austrian Ministry for Labour, Social Affairs and Consumer Protection).*