DISSERTATION

Titel der Dissertation

Rigorous Techniques for Continuous Constraint Satisfaction Problems

Verfasser

Mag. Ferenc Domes

angestrebter akademischer Grad

Doktor der Naturwissenschaften (Dr.rer.nat)

Wien, 2010

Studienkennzahl lt. Studienblatt:	A 091 405
Dissertationsgebiet lt. Studienblatt:	Mathematik
Betreuer:	Prof. Dr. Arnold Neumaier

Declaration of Authorship

I, Ferenc Domes, declare that this thesis titled, Rigorous techniques for continuous constraint satisfaction problems and the work presented in it are my own. I confirm that:

- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:
Ferenc Domes, 2010, Wien

"There are no shortcuts to any place worth going"

Helen Keller

Abstract

This thesis contributes rigorous techniques for solving continuous constraint satisfaction problems, i.e., finding one or all points (called feasible points) satisfying a given family of equations and/or inequalities (called constraints). Many real word problems are continuous constraint satisfaction problems. New and old state of the art methods are presented, integrated in GloptLab, a new easy-to-use testing and development platform for solving quadratic constraint satisfaction problems. The basic solution principle is branch and prune and filtering. Filtering techniques tighten a box – the Cartesian product of intervals defined by the bounds on the variables. In order to avoid a loss of feasible points, rigorous error estimation using interval arithmetic and directed rounding are used, to take care that all calculations are valid even though the calculations are performed in floating-point arithmetic only.

Discussed are the mathematical background, algorithms and tests of constraint propagation, strictly convex enclosures, linear relaxations, finding, using and verifying approximately feasible points, optimal scaling and other auxiliary techniques. In particular:

• Constraint propagation is based on a sequence of steps, each using a single constraint only. Traditional techniques are extended by special quadratic constraint propagation methods using new techniques for eliminating bilinear entries and finding optimal enclosures for separable quadratic expressions.

• A quadratic inequality constraint with a positive definite Hessian defines an ellipsoid. A rounding error controlled version of the Cholesky factorization is used to transform a strictly convex quadratic constraint into a norm inequality, for which the interval hull is easy to compute analytically.

• Different methods for computing linear relaxations are discussed, combined and extended. Partially improved and existing methods, as well as new approaches for rigorously enclosing the solution set of linear systems of inequalities are presented.

• Numerous examples show that the above methods complement each other and how to create solution strategies that solve constraint satisfaction problems globally and efficiently.

Abstract

Diese Arbeit beschäftigt sich mit rigorosen Techniken für das Lösen kontinuierlicher Zulässigkeitsprobleme. Das heißt, wir suchen nach einem oder allen Punkte, genannt zulässige Punkte, die eine Familie von Gleichungen und/oder Ungleichungen erfüllen, die wir im Weiteren Nebenbedingungen nennen werden. Zahlreiche Anwendungen führen auf kontinuierliche Zulässigkeitsprobleme. Neue und bereits existierende moderne Methoden werden präsentiert und integriert in GloptLab, eine neue, leicht bedienbare Test- und Entwicklungsplattform zum Lösen quadratischer Zulässigkeitsprobleme. Der Lösungsalgorithmus beruht auf dem Grundprinzip von Branch-and-Prune und auf Filterung. Filterungsmethoden dienen zur Verkleinerung/Reduktion einer Box, definiert als das kartesische Produkt der Intervalle, die die Schranken an die Variablen festlegen. Um den Verlust zulässiger Punkte zu vermeiden, werden alle Fehlerabschätzungen rigoros mittels Intervallarithmetik und gerichteter Rundung durchgeführt. Das stellt sicher, dass alle Rechnungen auch in Gleitkommaarithmetik gültig sind. In der Doktorarbeit werden die folgenden Themen diskutiert: der mathematische Hintergrund, Algorithmen und Tests für Constraint-Propagation, strikt konvexe Einschließungen, lineare Relaxationen, das Berechnen, korrekte Benutzen und Verifizieren approximativ zulässiger Punkte, optimale Skalierung und diverse Hilfsmethoden. Insbesondere:

• Constraint-Propagation basiert auf einer Folge von Schritten, die jeweils eine einzelne Nebenbedingung verwenden. Traditionelle Techniken werden durch eine spezielle quadratische Methode erweitert, die neue Verfahren für die Eliminierung bilinearer Einträge und für das Berechnen optimaler Einschließungen für separable quadratische Ausdrücke verwendet.

• Eine quadratische Ungleichungsnebenbedingung, die eine positiv definite Hesse-Matrix besitzt, definiert ein Ellipsoid. Eine spezielle rundungsfehlerkontrollierte Version der Cholesky-Zerlegung wird verwendet, um die strikt konvexe quadratische Nebenbedingungen in Norm-Ungleichungen zu transformieren. Für diese ist es dann einfach, die Intervall-Hülle analytisch zu bestimmen.

• Diverse Methoden für die Erzeugung linearer Relaxationen werden diskutiert, kombiniert und erweitert. Teilweise verbesserte, existierende und neue Verfahren für das rigorose Einschließen der Lösungsmenge linearer Systeme werden präsentiert.

• Eine Vielzahl von Beispielen demonstrieren, dass die präsentierten Verfahren einander ergänzen. Außerdem zeigen sie, wie man Lösungsstrategien entwickelt, die Zulässigkeitsprobleme global und effizient lösen.

Acknowledgements

I would like to thank my thesis advisor Arnold Neumaier, my parents and my wife Réka for their help and support. This thesis was supported through the research grant FSP 506/003 of the University of Vienna.

Contents

Notation

$\mathbb{N}_0$ denotes the set of natural numbers including zero and $\mathbb{R}_+$ the set of nonnegative reals. The n-dimensional identity matrix is denoted by I_n and the n-dimensional zero matrix is denoted by 0_n. The jth row vector of a matrix A is denoted by $A_{j:}$, the kth column vector by $A_{:k}$ and the number of nonzero entries by $\text{nnz}(A)$. The set $\neg N$ denotes the complement of a set N. The number of elements of a set N is denoted by $|N|$. Let $I \subseteq \{1, \ldots, n\}$ and $J \subseteq \{1, \ldots, m\}$ be index sets and let $n_I := |I|$, $n_J := |J|$. Let x be an n-dimensional vector, then x_I denotes the n_I-dimensional vector built from the components of x selected by the index set I. Let A be an $m \times n$ matrix, then $A_{:I}$ denotes the $m \times n_I$ matrix built from the rows of A selected by the index sets I. Similarly, $A_{J:}$ denotes the $n_J \times n$ matrix built from the columns of A selected by the index sets J. $(A^T)^{-1}$ is denoted by A^{-T}. For vectors and matrices the comparison operators $=$, $\neq$, $<$, $>$, $\leq$, $\geq$ and the absolute value $|A|$ of a matrix A are interpreted component-wise. For an $n \times n$ matrix A, $\text{diag}(A)$ denotes the n-dimensional vector with $\text{diag}(A)_i = A_{ii}$. For an $n \times n$ matrix A, $\text{Diag}(A)$ denotes the $n \times n$ diagonal matrix with

$$\text{Diag}(A)_{ij} := \begin{cases} A_{ii} & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

For an n-dimensional vector x, $\text{Diag}(x)$ denotes the $n \times n$ diagonal matrix with

$$\text{Diag}(A)_{ij} := \begin{cases} x_i & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

A matrix A is called as *very small with respect to a matrix* B if $|A| < |B|$ and for a fixed, given tolerance $0 < \kappa \ll 1$ (chosen by the implementation of the method)

$$\max_{i,j}(D_{ij}) \leq \kappa \text{ where } D_{ij} := \begin{cases} |A_{ij}|/|B_{ij}| & \text{if } |A_{ij}| \geq 1 \\ |A_{ij}| & \text{otherwise.} \end{cases} \tag{1}$$

For example if

$$A = \begin{pmatrix} 0.001 & 10 \\ 0.002 & 0.004 \end{pmatrix},\ B = \begin{pmatrix} 0.1 & 10000 \\ 0.003 & 0.02 \end{pmatrix},\ \text{then } D = \begin{pmatrix} 0.001 & 0.001 \\ 0.002 & 0.004 \end{pmatrix}$$

and $|A| < |B|$, the matrix A is very small with respect to B if the tolerance κ satisfies $\kappa \geq \max_{i,j}(D_{ij}) = 10^{-3}$. The expression $\|A\|_2 := \sup\{\|Ax\|_2 \mid \|x\|_2 = 1\}$ denotes the *Euclidean norm* of a matrix A. The expression $\lambda(A)$ denotes the set of all *eigenvalues* of a square matrix A. Furthermore, $\lambda_{max}(A)$ denotes the largest and $\lambda_{min}(A)$ the smallest eigenvalue of A.

$\mathbf{a} = [\underline{a}, \overline{a}]$ with $\underline{a} \leq \overline{a}$ denotes a real interval with a possibly infinite lower bound $\underline{a}$ and a possibly infinite upper bound $\overline{a}$. A bound is *large*, if its absolute value is greater than a configurable constant μ (whose default value is 10^6 in GloptLab). Decisions are often based on "if a bound is large" rather than on "if a bound is infinite". An interval is *large* if both of its bounds are large. The expressions

$$\operatorname{wid}(\mathbf{a}) := \overline{a} - \underline{a}$$

denotes the *width*,

$$\operatorname{mid}(\mathbf{a}) := (\underline{a} + \overline{a})/2$$

denotes the *midpoint*,

$$\langle \mathbf{a} \rangle := \begin{cases} \min(|\underline{a}|, |\overline{a}|) & \text{if } 0 \notin [\underline{a}, \overline{a}], \\ 0 & \text{otherwise,} \end{cases}$$

denotes the *mignitude* and

$$|\,\mathbf{a}| := \max(|\underline{a}|, |\overline{a}|)$$

denotes the *magnitude* of an interval $\mathbf{a}$. An interval is called *thin* or *degenerate* if its width is zero. An interval is called *narrow* if its width is less than a configurable constant η (whose default value is 10^{-6} in GloptLab). Decisions are often based on "if an interval is narrow" rather than on "if an interval is thin". The *sign of the interval* $\mathbf{a}$ is defined by

$$\operatorname{sign} \mathbf{a} = \begin{cases} 0 & \text{if } \underline{a} = \overline{a} = 0, \\ 1 & \text{if } \underline{a} \geq 0, \\ -1 & \text{if } \overline{a} \leq 0, \\ [\,-1, 1] & \text{if } \underline{a} < 0 < \overline{a}. \end{cases}$$

Note that this definition is not standard; for example, the standard MATLAB sign function is different. We also consider a quadratic expression $p(x)$ in $x = (x_1, \ldots, x_n)^T$ such that the evaluation at any $x \in \mathbf{x}$ is a real number. The box $p(\mathbf{x})$ is called an *interval enclosure* of $p(x)$ in the box $\mathbf{x}$ if $p(x) \in p(\mathbf{x})$ holds for all $x \in \mathbf{x}$. There are a number of methods for defining $p(\mathbf{x})$, for example interval evaluation or centered forms (for details, see, e.g., NEUMAIER [70]). If for all $y \in p(\mathbf{x})$ an $x \in \mathbf{x}$ exists such that $p(x) = y$, then $p(\mathbf{x})$ is called the *range*. If this only holds for $y = \inf p(\mathbf{x})$ and $y = \sup p(\mathbf{x})$, then $p(\mathbf{x})$ is the *interval hull* $\square\{p(x) \mid x \in \mathbf{x}\}$. To get rigorous results when using floating point arithmetic, one needs an implementation of interval arithmetic with outward rounding. Another – and somewhat trickier – alternative is is to compute the upper and the lower bound of the range separately, without the use of interval arithmetic, by using monotonicity properties of the operations. To get rigorous results when using floating point arithmetic, one needs here directed rounding. However, not all expressions can be

bounded from below or above using directed rounding only; and detailed considerations are needed in each particular case. For an expression p, $\nabla\{p\}$ denotes the result obtained when first the rounding mode is set to downward rounding, then p is evaluated, and by $\Delta\{p\}$ the result obtained when first the rounding mode is set to upward rounding, then p is evaluated. Assumed is that negating an expression is done without error; thus, e.g., $\Delta\{-(x-y)\} = -\Delta\{x-y\}$ holds. Careful arrangement allows in many cases to replace downward rounded expressions by equivalent upward rounded expressions. For example, $\nabla\{x-y\} = \Delta\{-(y-x)\}$. If this is possible, one can achieve correct results using only upward rounding (thus saving rounding mode switches), while in INTLAB's interval arithmetic (see RUMP [82]), the rounding mode is switched often, slowing down the computations.

An *interval vector* $\mathbf{x} = [\underline{x}, \overline{x}] \in \overline{\mathbb{IR}}^n$ or a *box* is the Cartesian product of the closed real intervals $\mathbf{x}_i := [\underline{x}_i, \overline{x}_i]$, representing a (bounded or unbounded) axiparallel box in $\mathbb{R}^n$. The values $-\infty$ and ∞ are allowed as lower and upper bounds, respectively, to take care of one-sided bounds on variables. $\overline{\mathbb{IR}}^n$ denotes the set of all n-dimensional boxes. A box is large or narrow when all its components are large or narrow. Operations defined for intervals (like width, midpoint, mignitude and magnitude) are interpreted component-wise when applied to boxes. The condition $x \in \mathbf{x}$ is equivalent to the collection of simple bounds

$$\underline{x}_i \le x_i \le \overline{x}_i \quad (i = 1, \ldots, n),$$

or, with inequalities on vectors and matrices interpreted component-wise, to the two-sided vector inequality $\underline{x} \le x \le \overline{x}$. Apart from two-sided constraints, this includes with $\mathbf{x}_i = [a, a]$ variables x_i fixed at a particular value $x_i = a$, with $\mathbf{x}_i = [a, \infty]$ lower bounds $x_i \ge a$, with $\mathbf{x}_i = [-\infty, a]$ upper bounds $x_i \le a$, and with $\mathbf{x}_i = [-\infty, \infty]$ free variables.

To account for inaccuracies in computed entries of a matrix, we use interval matrices, standing for uncertain real matrices whose coefficients are between given lower and upper bounds. The expression $\mathbf{A} := [\underline{A}, \overline{A}] \in \overline{\mathbb{IR}}^{m \times n}$ denotes an $m \times n$ interval matrix with lower bound $\underline{A}$ and upper bound $\overline{A}$. $\mathbf{A} \in \overline{\mathbb{IR}}^{n \times n}$ is symmetric if $\mathbf{A}_{ik} = \mathbf{A}_{ki}$ for all $i, k \in \{1, \ldots, n\}$. The comparison matrix $\langle \mathbf{A} \rangle$ of a square interval matrix $\mathbf{A}$ is defined by

$$\langle \mathbf{A} \rangle_{ij} := \begin{cases} -|\,\mathbf{A}_{ij}| & \text{for } i \ne j, \\ \langle \mathbf{A}_{ij} \rangle & \text{for } i = j. \end{cases}$$

A real matrix A is identified with the narrow interval matrix with $\underline{A} = \overline{A} = A$; in particular, its comparison matrix is

$$\langle A \rangle_{ij} := \begin{cases} -|A_{ij}| & \text{for } i \ne j, \\ |A_{ij}| & \text{for } i = j. \end{cases}$$

The width and the radius of an interval matrix A are the real matrices defined by

$$\operatorname{wid}(\mathbf{A}) := \overline{A} - \underline{A}, \text{ and } \operatorname{rad}(\mathbf{A}) := \operatorname{wid}(\mathbf{A})/2,$$

respectively. A symmetric interval matrix $\mathbf{A} \in \overline{\mathbb{IR}}^{n\times n}$ is called positive definite if all symmetric $A \in \mathbf{A}$ are positive definite, $x^T Ax > 0$ for all nonzero $x \in \mathbb{R}^n$. An interval matrix $\mathbf{A} \in \overline{\mathbb{IR}}^{n\times n}$ is called an H-Matrix iff a vector $u > 0$ with $\langle \mathbf{A} \rangle u > 0$ exists (see, e.g., NEUMAIER [70]).

Chapter 1

Introduction

1.1 Outline

This thesis contributes rigorous techniques for solving continuous constraint satisfaction problems. A continuous constraint satisfaction problem is the special case of a continuous optimization problem where the objective function is constant. Many real world problems are continuous constraint satisfaction problems. Applications include robotics (Grandon et al. [37], Merlet [66]), localization and map building (Jaulin [49], Jaulin et al. [50], biomedicine Cruz & Barahona [23]), circle packing (Markót & Csendes [64], Szabó et al. [96]), or the protein folding problem (Krippahl & Barahona [58], Neumaier [71]).

Solving a constraint satisfaction problem consist in finding one or all feasible points, i.e., points satisfying a given family of equations and/or inequalities (called constraints). The basic solution principle is branch and prune based on filtering. Filtering techniques tighten a box – the Cartesian product of intervals defined by the bounds on the variables. The box enclosing the set of all feasible points of a constraint satisfaction problem is called bound constraint. Branch and prune is used to partition the bound constraints smaller subboxes followed by pruning methods applied to each of them in order to find a better covering of the set of all feasible points.

In practice, constraint satisfaction problems are solved by a combination of a variety of techniques, often involving constraint propagation, with either some form of stochastic search or a branch and prune scheme for a complete search. These techniques are mainly complemented by filtering or pruning techniques based on techniques borrowed from optimization, such as linear or convex relaxations (see, e.g., Neumaier [73]).

Rigorous methods guarantee that no feasible point is lost if they are used to tighten the bound constraints. In order to avoid a loss of feasible points, rigorous error estimation using interval arithmetic and directed rounding are used, to take care that all calculations are valid even though the calculations are performed in floating-point arithmetic only.

Old, new, and modified state of the art methods are presented and integrated in GLOPTLAB, a configurable framework for the rigorous global solution of quadratic constraint satisfaction problems. Discussed are the mathematical background, algorithms and tests of constraint propagation, strictly convex enclosures, linear relaxations, finding, using and verifying approximately feasible points, optimal scaling and global solution techniques.

The thesis is organized as follows: In Chapter 2 GLOPTLAB is presented, a framework which combines several new or state of the art methods with special features ranging from the building of user-defined solution strategies to interactive solution based on a graphical user interface. The detailed description of the system is already published in [27]. The remaining part of the thesis discusses some of the methods already integrated into the GLOPTLAB environment. The methods are logically divided into the following chapters which are slightly modified versions of joint papers with my thesis advisor, which are under review or already published in a scientific journal. At the end of each chapter the contribution to the research by the authors are discussed in detail.

- Chapter 3 is about constraint propagation on quadratic constraints, a basic and cheap tool for extracting bound information from the constraints. Constraint propagation is based on a sequence of steps, each using a single constraint only. Traditional techniques are extended by special quadratic constraint propagation methods using new techniques for eliminating bilinear entries and finding optimal enclosures for separable quadratic expressions. This chapter is a slightly modified version of the already published paper [30].

- Chapter 4 discusses rigorous enclosures of ellipsoids, a method for computing the interval hull of strict convex constraints. A quadratic inequality constraint with a strictly convex Hessian defines an ellipsoid. A rounding error controlled version of the Cholesky factorization is used to transform a strictly convex quadratic constraint into a norm inequality, for which the interval hull is easy to compute analytically, but the rigorous version with rounding error control is highly nontrivial. This chapter is a slightly modified version of the paper [31] which is under review.

- Chapter 5 is about rigorous filtering using linear relaxations. Different methods for computing linear relaxations are discussed, combined and extended. Partially

improved and existing methods, as well as new approaches for rigorously enclosing the solution set of linear systems of inequalities are presented. The loss of sharpness caused by relaxing the problem is compensated by the ease and efficiency with which the relaxed problems can be solved. This chapter is a slightly modified version of the paper [33] which is under review.

- Various examples illustrate the advantage of the methods discussed. Testing is mainly done using the TEST ENVIRONMENT (DOMES et al. [28]), an interface for comparing and testing of different optimization solvers. This tool for both developers of solver software and practitioners allows us to exploit how the methods complement each other to solve constraint satisfaction problems globally, rigorously and efficiently.

I have written two other, related papers (one already published, the other submitted) discussing additional methods for constraint satisfaction problems:

- Good scaling is an essential requirement for the good behavior of many numerical algorithms. The paper DOMES & NEUMAIER [29] surveys scaling algorithms for systems of polynomials from the literature, and discuss some new ones, applicable to arbitrary polynomial constraint satisfaction problems. This includes systems of polynomial equations; but there may be fewer or more constraints than variables. The technique extends without problems to the scaling of polynomial optimization problems.

- Local search techniques for finding an approximate feasible point are used to find a small box around the approximation and verified to contain a feasible point. Even if no approximate feasible point could be located, the information extracted from the search can often be used to reduce the bounds around the solution set. The paper DOMES & NEUMAIER [32] discusses the feasible point search, present new techniques for pruning using feasible or nearly feasible points and a method for verifying feasible points close to given approximations.

1.2 Problem specification

A continuous, global optimization problem can be posed as

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & C(x) \in \mathbf{F} \\ & x \in \mathbf{x}. \end{array} \tag{1.1}$$

The function $f(x) : \mathbb{R}^n \to \mathbb{R}$ is called the **objective function**, the m **general constraints** $C_i(x) \in \mathbf{F}_i$, $(i = 1, \ldots, m)$ are interpreted as component-wise enclosures. This includes equality constraints if $\mathbf{F}_i = [F_i, F_i]$ is a degenerate interval, and inequality constraints if one of the bounds is infinite and two-sided constraints $\underline{F_i} \leq C_i \leq \overline{F_i}$ if both bounds are finite, and different. The n **bound constraints** $x_j \in \mathbf{x}_j$ with $j = 1, \ldots, n$ are interpreted as enclosures $x_j \in \mathbf{x}_j$ with $j = 1 \ldots n$. Again, fixed variables and one-sided bounds on the variables are included as special cases.

An $x \in \mathbf{x}$ is called a **feasible point** if $C(x) \in \mathbf{F}$ is satisfied. The problem is called **infeasible** if there are no feasible points. A **constraint satisfaction problem** is an optimization problem with a constant objective function $f(x) = c$. In this case, the task is either to find a single feasible point or to find a finite covering or bounding box of the set of all feasible points by n-dimensional boxes of small volume.

This thesis considers **rigorous methods** for bounding the feasible domain. Rigorous methods **reduce the search space** $(\mathbf{x}, \mathbf{F})$ while **guaranteeing** that no feasible points are lost. Formally, using the problem specification (1.1), rigorous means that each method $\Gamma : (\mathbf{x}, \mathbf{F}) \to (\tilde{\mathbf{x}}, \tilde{\mathbf{F}})$ has the property

$$\{x \in \mathbf{x} \mid C(x) \in \mathbf{F}\} = \{x \in \tilde{\mathbf{x}} \mid C(x) \in \tilde{\mathbf{F}}\}.$$

The reduction property is ensured by requiring that $\tilde{\mathbf{x}} \subseteq \mathbf{x}$ and $\tilde{\mathbf{F}} \subseteq \mathbf{F}$.

Polynomial expressions in standard form are represented as a linear combination of monomials. All monomials occurring in some general constraints are collected together in a vector-valued function $p(x) : \mathbb{R}^n \to \mathbb{R}^{n_p}$ with components

$$p(x)_k = \prod_{j=1}^{n} x_j^{E_{kj}} \quad \text{for } k = 1 \ldots p.$$

Here $E \in \mathbb{N}_0^{s \times n}$ is a sparse matrix encoding the powers with which the variables appear in the monomials used. The corresponding polynomial coefficients are collected in a sparse matrix $A \in \mathbb{R}^{m \times n_p}$. Thus the general **polynomial constraint satisfaction problem** with n variables and m constraints takes the form

$$x \in \mathbf{x}, \quad Ap(x) \in \mathbf{F} \tag{1.2}$$

with $p(x)$ as above, $A \in \mathbb{R}^{m \times n_p}$, $\mathbf{x} \in \mathbb{IR}^n$, and $\mathbf{F} \in \mathbb{IR}^m$.

To be able to rigorously account for uncertainties due to measurements of limited accuracy, conversion errors from an original representation to our normal form, and rounding

errors when creating new constraints by relaxation techniques, the coefficients in the constraints are allowed to vary in (narrow) intervals. Thus the coefficient matrix A can vary arbitrarily within some interval matrix $\mathbf{A}$. The $m \times n_p$ interval matrix $\mathbf{A}$ with closed and bounded interval components $\mathbf{A}_{ik} = [\underline{A}_{ik}, \overline{A}_{ik}]$, is interpreted as the set of all $A \in \mathbb{R}^{m \times n_p}$ such that $\underline{A} \leq A \leq \overline{A}$, where $\underline{A}$ and $\overline{A}$ are the matrices containing the lower and upper bounds of the components of $\mathbf{A}$.

As observed by [60] polynomial constraints can be decomposed in many different ways into quadratic constraints by introducing additional variables, many of the presented methods assume that all constraints are quadratic. Therefore the **quadratic constraint satisfaction problem with uncertain constraint coefficients** is written in the form

$$Aq(x) \in \mathbf{F}, \quad x \in \mathbf{x}, \quad A \in \mathbf{A}, \tag{1.3}$$

with the $n(n+1)$-dimensional, redundant, quadratic monomial vector

$$q(x) = (x_1, \ldots, x_n, x_1^2, \ldots, x_1 x_n, \ldots, x_n x_1, \ldots, x_n^2)^T.$$

If additional n_s slack variables x^s are introduce in addition to the n_o original variables x^o, then the quadratic constraint satisfaction problem in the equality form is given by

$$Aq(x) = 0, \quad x \in \mathbf{x}, \quad A \in \mathbf{A}, \tag{1.4}$$

where $n_e = n_o + n_s$ is the number of the variables $x = (x^o \;\; x^s)^T$ and

$$q(x) = (1, x_1, \ldots, x_{n_e}, x_1^2, \ldots, x_1 x_{n_e}, \ldots, x_{n_e} x_1, \ldots, x_{n_e}^2)^T.$$

A special representation of general optimization problems is also used, in which (1.3) and (1.4) are special cases, where the objective function is constant and no user-defined univariate functions (see below) occur: Let $I, J \subseteq \{1, \ldots, n\}$ be index sets then

$$x_j := \phi_k(x_i) \text{ with } j \in J, \quad k \in \{1, \ldots, n_u\}, \quad i \in I \tag{1.5}$$

assigns the univariate function ϕ_k depending on the variable x_i to the variable x_j. For example, if $(i, j, k) = (3, 4, 2)$ and $\phi_2(z) = \sin(z + \pi/3)$ then $x_4 := \phi_2(x_3) = \sin(x_3 + \pi/3)$ is an additional univariate non-quadratic constraint definition. The term $x_J := \phi(x_I)$ in (1.6) represents all n_u univariate function definitions. An *objective function* is also defined, allowing to handle continuous optimization problems in future versions of GloptLab.

The optimization problem

$$\begin{array}{ll} \min & A_{o:}q(x) \in \mathbf{F}_o \\ \text{s.t.} & Aq(x) \in \mathbf{F} \ \text{ for some } A \in \mathbf{A}, \\ & x \in \mathbf{x},\ x_J = \phi(x_I). \end{array} \tag{1.6}$$

with

$$x \in \mathbb{R}^n, \quad q(x) \in \mathbb{R}^{n_q}, \quad A \in \mathbb{R}^{m \times n_q}, \quad i \leq n,\ |I| = |J| = n_u$$

and $\phi : \mathbb{R}^{n_u} \to \mathbb{R}^{n_u}$ is called the **internal inequality representation of** GloptLab.

Since the above representation is often obtained from converting non-quadratic problems by introducing additional intermediate variables, the variables are categorized as the n_o original variables x^o, n_i intermediate variables x^i (e.g., variables used substituting univariate functions) and the n_s slack variables x^s by writing $x = (x^o \ \ x^i \ \ x^s)^T$ with $n = n_o + n_i + n_s$. There are no slack variables in the internal inequality representation (1.6) but slack variables may occur in the **internal equality representation of** GloptLab:

$$\begin{array}{ll} \min & A_{o:}q(x) \in \mathbf{F}_o \\ \text{s.t.} & Aq(x) = 0 \ \text{ for some } A \in \mathbf{A}, \\ & x \in \mathbf{x},\ x_J := \phi(x_I). \end{array} \tag{1.7}$$

where

$$i \in \{1, \ldots, n\}, \quad x \in \mathbb{R}^n, q(x) \in \mathbb{R}^{n_q+1}, \quad A \in \mathbb{R}^{m \times (n_q+1)}, \quad |I| = |J| = n_u,$$

and $\phi : \mathbb{R}^{n_u} \to \mathbb{R}^{n_u}$.

The conversion from the AMPL format to the internal problem representation of GloptLab is done by AMPL in connection with the Coconut Environment [88], while the parsing and conversion from a simplified AMPL format is done by the SMPL parser [63]. Converting the GloptLab problem representation (.def, .glb files) to AMPL or SMPL formats is also possible. More information about the conversion possibilities can be found in Figure 2.1 (Subsection 2.4.1).

By computing a linear relaxation of nonlinear problem (see Chapter 5 for details) a **linear system of inequalities** is obtained and can be written as

$$Ex \in \mathbf{b},\ x \in \mathbf{x}, \tag{1.8}$$

where E is an $m \times n$ real matrix, $\mathbf{b} := [\underline{b}, \overline{b}]$ is an m–dimensional and $\mathbf{x}$ is an n–dimensional box. The linear expressions comprise component-wise linear enclosures $E_{i:}x \in \mathbf{b}_i$. This includes linear equalities if $\mathbf{b}_i$ is a degenerate interval with $\underline{b}_i = \overline{b}_i$, linear inequality

constraints if one bound of $\mathbf{b}_i$ is infinite, and two-sided linear inequalities if both bounds are finite.

Chapter 2

GloptLab - a configurable framework for the rigorous global solution of quadratic constraint satisfaction problems

Abstract.

GloptLab is an easy-to-use testing and development platform for solving quadratic constraint satisfaction problems, written in Matlab.

The algorithms implemented in GloptLab are used to reduce the search space: scaling, constraint propagation, linear relaxations, strictly convex enclosures, conic methods, and branch and bound. All these methods are rigorous, hence it is guaranteed that no feasible point is lost. Finding and verifying feasible points complement the reduction methods. From the method repertoire custom made strategies can be built, with a user-friendly graphical interface.

GloptLab was tested on a large test set of constraint satisfaction problems, and the results show the importance of compose a clever strategy.

2.1 Introduction

GloptLab is a *testing and development platform*, implemented in Matlab, for rigorously solving *quadratic constraint satisfaction problems*, i.e., for finding multivariate

points satisfying a given list of quadratic equations and inequalities, in a way that ensures that no possible solution is lost during the solution process. GloptLab supports rigorous input, converting decimal numbers that are not exactly representable into narrow interval coefficients accounting for the conversion errors. Coefficients may also be specified directly as narrow intervals.

GloptLab can also solve other algebraic constraint satisfaction problems, using the `dag2gloptlab` converter of the Coconut Environment, which introduces intermediate variables to transform algebraic problems to quadratic ones.

All methods implemented in GloptLab are rigorous since they meet the above property. The methods in GloptLab return a certificate which can be used to verify the solution process and to automatically generate human-readable computer assisted proofs.

The reduction methods implemented in GloptLab were jointly developed with Arnold Neumaier, and are described in separate publications quoted in Section 2.2, where a short introduction to each of them is given. The input format is analyzed and preprocessed by the *problem simplification.Constraint propagation* is a fast and effective method which is also used as a part of other more complicated methods (see Chapter 3). We use different *linear relaxation* techniques to get finite bounds or decrease the size of the search space (see Chapter 5). *Strict convex enclosures* compute a nearly optimal interval hull of strictly convex constraints (see Chapter 4). *Conic methods* may lead to spectacular reductions of the search domain, but require a great deal of computation time (see Domes & Neumaier [34]). *Branch and bound* divides the search space into smaller subdomains and applies some of the above methods to reduce their size or even eliminate them when they do not contain feasible points. The boxes which remain after the branching can be merged to a single or fewer ones by computing their *interval hull* or finding and bounding the *clusters* of them. *Finding and verifying feasible points* are important if we search only for a single solution of the constraint satisfaction problem. Different *scaling algorithms* guarantee that the methods, which are not scaling invariant, do not run into difficulties due to bad scaling (see Domes & Neumaier [29]).

Some of the methods mentioned above make use of *external toolboxes*. Since the verification is often based on interval techniques, INTLAB (Rump [82]) is probably the most important toolbox, although some methods avoid using it to speed up the computation, it is always needed to run GloptLab. Unverified solutions of linear programs are obtained by using LPSolve (Berkelaar et al. [16]), SeDuMi (Sturm et al. [95]) or SDPT3 (Toh et al. [98]). The latter two can be also used to optimize over symmetric cones which make them an essential part of the rigorous conic methods. In general, the nonrigorous parts are only used for generating approximations which are needed in subsequent rigorous computation steps. The algorithms for finding and verifying feasible

points make use of local solvers like projected BFGS and conjugate gradient methods from Kelley [56]. AMPL (Fourer et al. [35]), the Coconut Environment (Schichl [87]) and the SMPL parser (Markót [63]) are also used to convert to the internal representation of the problem.

There is a number of software packages for solving constraint satisfaction problems. The Numerica software by Hentenryck et al. [43] uses branch and prune methods and interval constraint programming to solve constraint satisfaction problems. The ICOS solver by Lebbah [59] is a software package for the rigorous solution of nonlinear and continuous constraints, based on constraint programming and interval analysis techniques. The PaLM system by Jussien & Barichard [52] uses explanation-based constraint programming, and propagates the constraints of the problem, learning from the failures of the solver. The prize-winning solver Baron by Sahinidis & Tawarmalani [86] can also solve constraint satisfaction problems. Initiated by the development of interval analysis on directed acyclic graphs by Schichl & Neumaier [89], the Coconut Environment [87, 88] has been developed as a global optimization software platform.

Typically, the solvers quoted require finite and not too large two-sided bound constraints to ensure the efficiency of the interval techniques. Formally, unbounded problems are often tightened (e.g., by Baron) by adding artificial bound constraints, with the resulting danger of excluding feasible points. Some of the best solvers (e.g., Baron) use unverified methods and return unverified results. The reason is that verifying the results or error control is often considered to be an unnecessary extra effort. However there is a number of cases where serious safety problems can arise from unverified results. This has motivated research in robotics (e.g., Merlet [67]) and more generally in safe computation techniques (Jansson [46], Keil [55], Lebbah et al. [61]). Uncertainties in the input data are even more often ignored. In general the modeling languages (e.g., AMPL [35] or GAMS by Brooke et al. [18]) do not support an exact treatment of rational or interval constraint coefficients.

Section 2.4 contains the novel contribution as we discuss the integration of the above methods in the GloptLab environment, features ranging from the building of user defined solution strategies with the graphical user interface to the possibilities of extending the method repertoire. An example can be found in Section 2.5, while in Section 2.6 we present some test results of GloptLab analyzed by the Test Environment (see Neumaier et al. [75] and Domes et al. [28] for the current version). In the final section we summarize the most important features of GloptLab, give some perspectives and talk about future work.
More information is available at the GloptLab homepage:

`http://www.mat.univie.ac.at/~dferi/gloptlab.html`

The public version of GloptLab is available at:

`http://www.mat.univie.ac.at/~dferi/gloptlab/download.html`

2.2 The implemented methods

There is a number of different rigorous methods developed for and integrated in GloptLab. In this chapter we give a brief description of the most important methods of this constantly expanding repertoire.

2.2.1 Problem simplification and scaling

This is usually the first step after reading a problem. In the problem simplification phase several preprocessing steps are done: We first identify and *remove bound constraints* from the general constraints, and store their bounds in the box $\mathbf{x}$. *Unbounded constraints* – where the corresponding interval F_i in (1.6) is unbounded – *are removed*. Possibly *redundant constraints are identified* and can be optionally removed. The *problem can be transformed into the equality representation* (1.4) by introducing additional slack variables. *Additional structural characteristics* like sparsity pattern are also derived.

The polynomial scaling problem consists of *finding* a *constraint scaling vector* $r \in \mathbb{R}^m_+$ and a *variable scaling vector* $c \in \mathbb{R}^n_+$ such that the scaled problem

$$x \in \mathbf{x}, \quad A^s q(x) \in \mathbf{F}^s \quad \text{with} \quad A^s_{ik} := r_i |A_{ik}| q(c)_k, \quad \mathbf{F}^s_i := r_i \mathbf{F}_i \tag{2.1}$$

is well-scaled in an appropriate sense. Which properties constitute a well-scaled problem is a somewhat ill-defined matter, because it highly depends on the applications and is not easily quantifiable. Intuitively, a scaling algorithm should somehow decrease large variations between appropriately weighted sums of logarithms of the coefficients of the matrix A; the weights should reflect the expected size of the values of the monomials. In GloptLab we can choose between the Hompack (Watson & Terry [103]) algorithm, Morgan's algorithm (Morgan [68], Chapter 5), and the methods LP and ScaleIT described in Domes & Neumaier [29]. The computed scaling vectors are then stored and later used by different methods.

The task Simplify simplifies the problem, and computes the scaling vectors. The task parameters are:

Parameters of the task SIMPLIFY		
Parameter	Type	Description
OBJECTIVE	selection	what to do with the objective? (remove, etc.)
SCALING	selection	select the used scaling method.
LINEAR SOLVER	selection	select a possible linear solver for the scaling.

2.2.2 Constraint propagation

Filtering techniques which tighten a box are called *constraint propagation* if they are based on single constraints only. *Forward propagation* uses the bound constraints to improve the bounds on the general constraints; *backward propagation* uses the bounds on the general constraints to improve the bounds on the variables.

Since (1.3) only consists of quadratic expressions, we can write each constraint without loss of generality in the form

$$\sum_k (a_k x_k^2 + b_k x_k) + \sum_{\top j,kj>k} b_{jk} x_j x_k \geq c, \quad x \in \mathbf{x},$$

where the $a_k x_k^2$ are the quadratic, the $b_k x_k$ the linear and the $b_{jk} x_j x_k$ the bilinear terms.

We first separate the constraint by approximating or bounding the bilinear terms, then we apply the forward propagation step: we compute the enclosure $\mathbf{p}_k$ of each univariate quadratic term $p_k(x_k) := a_k x_k^2 + b_k x_k$, where the uncertainties $\mathbf{a}_k$ and $\mathbf{b}_k$ of the constraint coefficients are also taken into account. Then we use the $\mathbf{p}_k$ to verify that the constraint is feasible, to get a new bound on each $p_k(x_k)$ and to find a new lower bound for the constraint. If the constraint has been found feasible, we can apply the backward propagation step and find the set of all x_k with $a_k x_k^2 + b_k x_k \in \mathbf{p}_k$. Finally, if we cut the bounds found with the original bound on the variables, we may obtain tighter bound constraints.

The method is cheap, rigorous, and does not require interval arithmetic since only directed rounding is used. It is often used in other methods for verifying approximate solutions. In general, if used as a stand alone technique, more than one step of constraint propagation is done successively, until no further significant reduction takes place.

A more detailed description of our constraint propagation can be found in Chapter 3.

The task PROPAGATE completes one step of constraint propagation. The task parameters are:

Parameters of the task PROPAGATE		
Parameter	Type	Description
METHOD	selection	select separable or linear method.
FULL MODE	decision	select only the forward mode or the method.

2.2.3 Linear relaxations

Linear constraints of the form

$$Ex \geq b,\ x \in \mathbf{x}. \tag{2.2}$$

may be obtained by relaxing the constraints of (1.3). Every feasible point of the constraint satisfaction problem (1.3) satisfies (2.2) iff for all $x \in \mathbf{x}$ and $A \in \mathbf{A}$ the inequalities

$$Aq(x) + \underline{b} - \underline{F} \leq Ex \leq Aq(x) + \overline{b} - \overline{F}$$

hold. In this case the linear system (2.2) is called a linear relaxation of (1.3) (proof can be found in Chapter 5). The relaxation (2.2) is found by computing interval enclosures, by using constraint propagation from Subsection 2.2.2 and by finding linear under and overestimators: the function $u(x)$ is called a linear underestimator of $p(x)$ in the box $\mathbf{x}$, if for all $x \in \mathbf{x}$, $u(x) \leq p(x)$ holds. Similarly, the function $v(x)$ is called a linear overestimator of $p(x)$ in the box $\mathbf{x}$, if for all $x \in \mathbf{x}$, $p(x) \leq v(x)$ holds.

After linearizing the constraints we apply different methods to improve the bound constraints $x \in \mathbf{x}$. These methods are explained in detail in Chapter 5.

If some bounds in $\mathbf{x}$ are infinite and the feasible domain is bounded, the *linear bounding* method is used to get finite bound constraints. This requires the approximate solution of a *single* linear program and a single constraint propagation step to generate new finite and rigorous bounds. The only purpose of this method is to bound the feasible domain, and leads to no further improvements if applied more than one time.

In *linear contraction* we first compute new bounds on the constraints, then cut them with the original ones. Then a modified Gauss-Jordan elimination is used to precondition the system, then either a direct interval evaluation or a single constraint propagation step is used to get new bounds on some or all of the variables.

Among the methods based on linear relaxations, the *LP contraction* is the method which requires the most computational time since in each step we solve more than one linear program. We find the d most promising directions (usually $d = 3$) and minimize the upper and lower bound from these directions. This requires the approximate solution

of $2d$ linear programs, of which the dual solutions are used to generate new constraints. Propagating the new constraints may improve the bound on the selected variables.

The task LINEAR applies a linear method to the problem. The task parameters are:

Parameters of the task LINEAR		
Parameter	Type	Description
METHOD	selection	select the method bound, contract or solve.
LINEAR SOLVER	selection	select an external linear solver.
EQU SOLVER	selection	select a method for solving equalities.

2.2.4 Strictly convex enclosures

A quadratic inequality constraint with a positive definite Hessian matrix defines an ellipsoid whose interval hull is easy to compute analytically. However, to cope efficiently with rounding errors is nontrivial.

For a real, symmetric matrix A we compute the *directed Cholesky factorization*; an approximate factorization $A \approx R^T R$ with nonsingular upper triangular R such that the error matrix $A - R^T R$ of the factorization is tiny and guaranteed positive semidefinite. Clearly, this implies that A is positive definite; conversely (in the absence of overflow), any sufficiently positive definite symmetric matrix has such a factorization with R representable in floating point arithmetic. In Chapter 4 we find such a representation which makes the error as small as possible and works even for nearly singular matrices.

We use the directed Cholesky factorization to transform a strictly convex quadratic constraint of the constraint satisfaction problem (1.3) into an ellipsoid defined by a Euclidean norm constraint

$$||Rx||_2^2 + 2a^T x \leq \alpha. \tag{2.3}$$

There is also need for scaling when factoring ill-conditioned matrices before applying the factorization. Therefore the scaling computed in the simplification is used before the directed Cholesky factorization is applied.

We derive the optimal box enclosure of this ellipsoid; we find constants β, γ, Δ and a vector $d > 0$ such that if $\Delta \geq 0$ then (2.3) implies

$$\|R(x - \tilde{x})\|_2 \leq \delta := \gamma + \sqrt{\Delta}, \quad |x - \tilde{x}|_2 \leq \frac{\delta}{\beta} d. \tag{2.4}$$

If $\Delta < 0$ then (2.3) has no solution $x \in \mathbb{R}$. For suitably chosen $\tilde{x}$ the bounds in (2.4) are optimal (for details and proof see Section 6,7, of Chapter 4).

By the second inequality of (2.4) we get rigorous bounds

$$\mathbf{u} := \left[(\delta/\underline{\beta})d - \tilde{x}, (\delta/\overline{\beta})d + \tilde{x}\right]$$

on the variables x. If we do this for each strictly convex quadratic constraint of the constraint satisfaction problem (1.3) and cut the resulting bounds with the original ones we may get tighter bound constraints.

By this method we get rigorous bounds on all n variables, obtainable with $O(n^3)$ operations. This should be used only once per problem, since successive application gives no further improvement of the bounds.

The task EHULL finds the ellipsoid hull of strict convex constraints. The task parameters are:

Parameters of the task EHULL		
Parameter	Type	Description
SCALING	decision	apply the scaling factors found by the task SIMPLIFY

2.2.5 Conic methods

Conic methods approximate the general constraints by hyperplanes, balls or hyperellipsoids, using semidefinite or conic programming in order to find sharp bounds on the feasible set of a quadratic constraint satisfaction problem. The conic methods use the internal equality form (1.4) and are based on the following proposition, improved by the techniques of SCHICHL & NEUMAIER [90]:

Proposition 2.1. *If G is positive semidefinite and $Z \leq 0$, than for any $x \in \mathbf{x}$ with $Eq(x) = 0$, we have*

$$0 \leq \begin{pmatrix} 1 \\ x \end{pmatrix}^T G \begin{pmatrix} 1 \\ x \end{pmatrix} - \begin{pmatrix} 1 \\ \overline{x} - x \\ x - \underline{x} \end{pmatrix}^T Z \begin{pmatrix} 1 \\ \overline{x} - x \\ x - \underline{x} \end{pmatrix} - z^T Aq(x). \tag{2.5}$$

Proof. Since by (1.4) the equality $Aq(x) = 0$ holds for all $x \in \mathbf{x}$ and by the definition of positive definiteness, all terms on the right hand side of (2.5) are greater or equal to zero. □

Now if G is positive semidefinite and $Z \leq 0$ the equation

$$\begin{pmatrix} 1 \\ x \end{pmatrix}^T G \begin{pmatrix} 1 \\ x \end{pmatrix} \leq \begin{pmatrix} 1 \\ \overline{x} - x \\ x - \underline{x} \end{pmatrix}^T Z \begin{pmatrix} 1 \\ \overline{x} - x \\ x - \underline{x} \end{pmatrix} + z^T Eq(x) + p(x)$$

implies that $0 \leq p(x)$. To find the positive semidefinite matrix G, the matrix Z, the vector z and free parameters in $p(x)$ we solve the conic program

$$\begin{aligned} \min \quad & c^T y \\ \text{s.t.} \quad & y_i \geq 0, \\ & \|r(y)\| \leq y_k, \\ & \tfrac{1}{2}\|s(y)\|^2 \leq y_j y_k, \\ & G \text{ symmetric and positive semidefinite,} \end{aligned} \tag{2.6}$$

with suitably chosen objective, non-negativity constraints $y_i \geq 0$, norm constraints $\|r(y)\| \leq y_k$, rotated conic constraints $\frac{1}{2}\|s(y)\|^2 \leq y_j y_k$ and the semidefiniteness constraint for the matrix G. Choosing one of the quadratic expressions

- $p(x) = \pm x_i + \zeta$ and minimizing ζ,
- $p(x) = -\sum_{i=1}^n x_i^2 + \zeta$ and minimizing ζ,
- $p(x) = -1$ and minimizing 0,
- $p(x) = -\|\omega \circ x\|^2 + 2\xi^T(\omega \circ x) + \delta$ with $\|\xi\| \leq \zeta$ and minimizing $\zeta + \delta$,

results in interesting enclosures of the feasible domain. Since the conic program (2.6) is solved by an approximate solver we get the approximate solutions $\hat{G}$, $\hat{Z}$ and $\hat{z}$, and we need to verify the results by computing

$$\hat{p}(x) := \begin{pmatrix} 1 \\ x \end{pmatrix}^T \hat{G} \begin{pmatrix} 1 \\ x \end{pmatrix} - \begin{pmatrix} 1 \\ \overline{x} - x \\ x - \underline{x} \end{pmatrix}^T \hat{Z} \begin{pmatrix} 1 \\ \overline{x} - x \\ x - \underline{x} \end{pmatrix} - \hat{z}^T Eq(x),$$

using interval arithmetic. Since $\hat{p}(x)$ is a rigorous enclosure of the feasible domain and a quadratic expression with narrow interval coefficients, we can use constraint propagation on it and may obtain tighter bound constraints.

Since the solution of the conic programs is rather costly, the maximal dimension of problems solved by this method is limited, and the number of iterative steps should be rather low. For details on the conic methods used in GloptLab (see Chapter 4).

The task Conic applies a conic method to the problem. The task parameters are:

Parameters of the task CONIC		
Parameter	Type	Description
METHOD	selection	bound, ellipsoid, feasibility or fixellipsoid.
CONIC SOLVER	selection	select an external conic solver.
REDUCE IN X ...	numeric	the reduction directions for the bound method.
Z SETTING	selection	select strategy for setting the matrix Z.
WEIGHT	decision	decide weighting the slack variables or not.
VERIFY	decision	verify the results or compute approximately.
INTROUND	decision	transform all coefficients into rational numbers.

2.2.6 Branch and bound

Using branch and bound on the constraint satisfaction problem (1.3) means that we partition the bound constraints $\mathbf{x}$ into s smaller subboxes, x^k $(k = 1, \ldots, s)$ such that $\mathbf{x} = \mathbf{x}^1 \cup \cdots \cup \mathbf{x}^s$ and use rigorous methods $\Gamma_i(\mathbf{x}^k, \mathbf{F})$ on each $\mathbf{x}^k$ separately. The methods applied to a subbox may reduce its width and even eliminate it if it contains no feasible points. There are different branching strategies, but in general they can be classified by the amount of memory they need. Recursive splitting selects a variable and splits the original box in this variable into two new boxes. The rigorous methods are applied to the first one, while the second one is stored on a stack. If the first box is reduced but not eliminated by the methods, it is split again, whereby the second part is again stored on the stack. This is done until the actual box is empty, a minimal width of the current box is reached, or the maximal number of elements on allowed the stack is exceeded. Then the last box is popped from the stack, reduced and split by using the same procedure. This is the *depth-first* split method. Since the maximal memory needed by the depth-first split is low this is the branching method which is currently implemented in GLOPTLAB. Choosing the ith direction in which a box is split is a critical issues; in GLOPTLAB either the one where $\mathbf{x}_i$ has maximal width or the one where the constraints have maximal range $\sum_k \mathrm{wid}(A_{k:}q(\mathbf{x}_i))$ can be chosen. Different variable selection methods and splitting strategies may be included in the future.

Recursive splitting results in a finite cover of the feasible domain by nonempty subboxes of a given maximum size. We can either return all boxes found or create the *interval hull* of them. Connected components of the union of the subboxes define *clusters*, which can be separately bounded by their interval hull. Since returning all boxes found often results in an unnecessary large amount of output and computing a single interval hull for distinct connected components is a crude approximation, therefore in most cases computing interval hull of clusters is the method of selection.

The task SPLIT divides the current box into sub-boxes and applies a solution strategy to each of them. The task parameters are:

Parameters of the task SPLIT		
Parameter	Type	Description
METHOD	selection	currently only the depthfirst is available.
DIR CHOOSER	selection	this criteria sets the split variable index.
SPLIT IN X	numeric	select the components of x which can be split.
ABSOLUTE SMALL	numeric	a small box has a small param. less than this.
MARGIN	numeric	margin between absolute and relative small.
RELATIVE SMALL	numeric	a small box has rel. small par. less than this.
SMALL BOX CRIT	selection	the criteria classifying the small parameters.
MAX DEPTH	numeric	the maximum allowed depth of the split.

The task MERGE BOXES merges the boxes found by the task SPLIT. The task parameters are:

Parameters of the task MERGE BOXES		
Parameter	Type	Description
METHOD	selection	select the interval hull or the cluster method.
TOLERANCE	numeric	tolerance between neighboring clusters.
DROP BOXES	decision	drop or leave the boxes found by the split.

2.3 Finding and verifying feasible points

An important step toward the rigorous solving of optimization problems is to find and verify feasible points of a constraint satisfaction problem.

To find a feasible point of the constraint satisfaction problem (1.3) we construct a smooth *feasibility distance function* $d(x) : \mathbb{R}^n \to \mathbb{R}$, which we minimize in the box $\mathbf{x}$ by using a local solver. The current selection of local solvers integrated in GLOPTLAB consists of BFGS, GRADIENT PROJECTION (both of them are MATLAB versions by KELLEY [56]), LBFGS-B (ZHU et al. [104]) and FMINCON (contained in the optimization toolbox for MATLAB). Note that most of these solvers require the gradient of the function $d(x)$, but since the feasibility distance function d is smooth this can be computed explicitly.

If an approximately feasible point x_f has been found, we try to find a box $\mathbf{b} \subseteq \mathbf{x}$ around x_f such that the existence of a feasible point inside of $\mathbf{b}$ is guaranteed by a mathematical existence theorem. This is called the *verification of feasible points*. For details on finding and verifying feasible points used in GLOPTLAB (see DOMES & NEUMAIER [32]).

In Section 2.6 we make use of finding feasible points to test our solver on a large test set of constraint satisfaction problems. The task FIND FEAS POINT can find and can verify a feasible point inside the current box. The task parameters are:

Parameter	Type	Description
LOCAL SOLVER	selection	select: bfgs, gradproj, lbfgs-b or fmincon.
MAX ITERATIONS	numeric	maximum number of solver iterations.
VERIFICATION	decision	try to verify the found point or not.
DELTA	numeric	δ constant used in the verification process.
SOLVER	selection	linear solver used for verification.

2.4 Integration of methods

The development of GLOPTLAB started in 2005 by Prof. Arnold Neumaier and myself. In the beginning, we experimented with constraint propagation techniques in order to reduce the search space of quadratic problems. Since GLOPTLAB was primary designed as a testing and development platform, we used the interpreted language MATLAB because of its ease of use and its graphical capabilities. In order to aid the development we developed a graphical interface providing a visual representation of the constraints and the current bound constraints during the reduction process. This was very useful for the debugging and testing of our programs. Since we intended later to extend the program to solve non-quadratic constraint satisfaction and optimization problems, we developed a general internal format in an early stage, and made only minor changes to it later.

As we added new methods like the ellipsoid hull and branch and bound to the method repertoire, we needed to decide which of the methods should be used in order to find a fast and reliable solution procedure, and how often and in which order they should be applied. Instead of making a fixed choice, we decided to create a task processor, and a strategy builder. To have an easy way to create, save, load and modify strategies, we added this features to the graphical user interface. We designed the GUI as a layer which is clearly separated from the solver engine, so that a batch solution of the problems is also possible.

Whenever we found a problem where our existing methods seemed to perform poorly we added new functionality, tasks, and parameters in order to improve the performance. For example, we added linear relaxations and conic programs. These lead to the development of a user extensible method repertoire. In order to obtain useful answers in case the complete search could not finish in the given time limit, we developed a method of finding

feasible points. To be competitive with other rigorous solvers we created a method for verifying feasible points close to a near-feasible approximation.

Now GloptLab has a rich selection of rigorous methods that we can use to build strategies and then apply it to solve quadratic constraints satisfaction problems.

We summarize the most interesting features of GloptLab:

- There is a well structured input format representing global optimization problems (already presented at the beginning of Section 2.2).
- At present only quadratic constraints are solved. The solution of non-quadratic, algebraic problems is possible by using the AMPL to GloptLab converter from the Coconut Environment, which automatically transforms algebraic terms to quadratic ones by introducing intermediate variables.
- The whole environment is implemented in a completely modular way, allowing easy portability of individual methods to other solvers and languages (see Subsection 2.4.1).
- Easy to use for prototyping and for development of new techniques in the context of other methods (Subsection 2.4.2).
- The strategy builder allows to test different strategies for different problem classes (Subsection 2.4.2).
- Interactive solution of a particular problem: it is possible to stop the execution of the strategy, remove and add new tasks to it and then resume the solution process. This approach can greatly reduce the solution time (Subsection 2.4.2).
- Contributors can add their own method with only minimal knowledge of the other parts of the software (see Subsection 2.4.3).
- The graphical user interface (Subsection 2.4.4) supports both the easy building of solution strategies and the visualization of the solution process.
- Using the batch mode of GloptLab, it is possible to run solution strategies in the Test Environment [28], or on processors without graphical support (Subsection 2.4.5).

In the following subsections we discuss the above items, however because of the extent of the topics we omit some details. More information can be found in the documentation files which are part of the GloptLab environment.

2.4.1 GloptLab structure diagram

An overview of the structure of GloptLab is given in Figure 2.1. This diagram emphasizes how the software is structured, and gives some overview over currently implemented features. These consist of the dependency of the methods of the external solvers, building strategies from different tasks, conversion possibilities from other input formats, measuring performance, or saving statistical information about all solved problems. The small table in the corner of Figure 2.1 shows which parts of GloptLab are internal, external, or GUI components. All external packages integrated into GloptLab are free. The software packages used are listed in the following table, and are packaged with the current GloptLab version. They can be downloaded and installed separately but in this case the corresponding path variables have to be set manually by editing the Gloptlab.cfg file or by using the editor of the graphical user interface.

Solver	required?	Function
Intlab	necessary	interval arithmetic
Coconut Environment	optional	converter of non-quadratic problems
Sedumi	optional	solver for conic and linear programs
SDPT3	optional	solver for conic and linear programs
lpsolve	optional	solver for linear programs

When Intlab and at least one package from the above selection which is capable to solve both conic and linear programs is installed, all current features of GloptLab can be used. Since the external solvers are connected to the GloptLab solver engine through an interface, adding new linear or conic solvers is not difficult. The new solvers are automatically recognized by the strategy builder opening new options in the task selection process.

2.4.2 Solution strategies

As shown in Figure 2.1, to solve a problem or a list of problems we need a strategy. A *solution strategy* or simply a *strategy* is a list of *tasks* used to solve a problem. A task could be the implementation of one of the methods described in Chapter 2.2, but there are other tasks like loops, conditions and breaks to extend the functionality and ensure the versatility of a strategy. Strategies are built comfortably by using the graphical strategy builder of the user interface (also see 1 in Figure 2.2), automatically ensuring a correct strategy syntax. New methods are automatically recognized by the strategy builder (for details see Subsection 2.4.3). The strategies can be applied to the problems

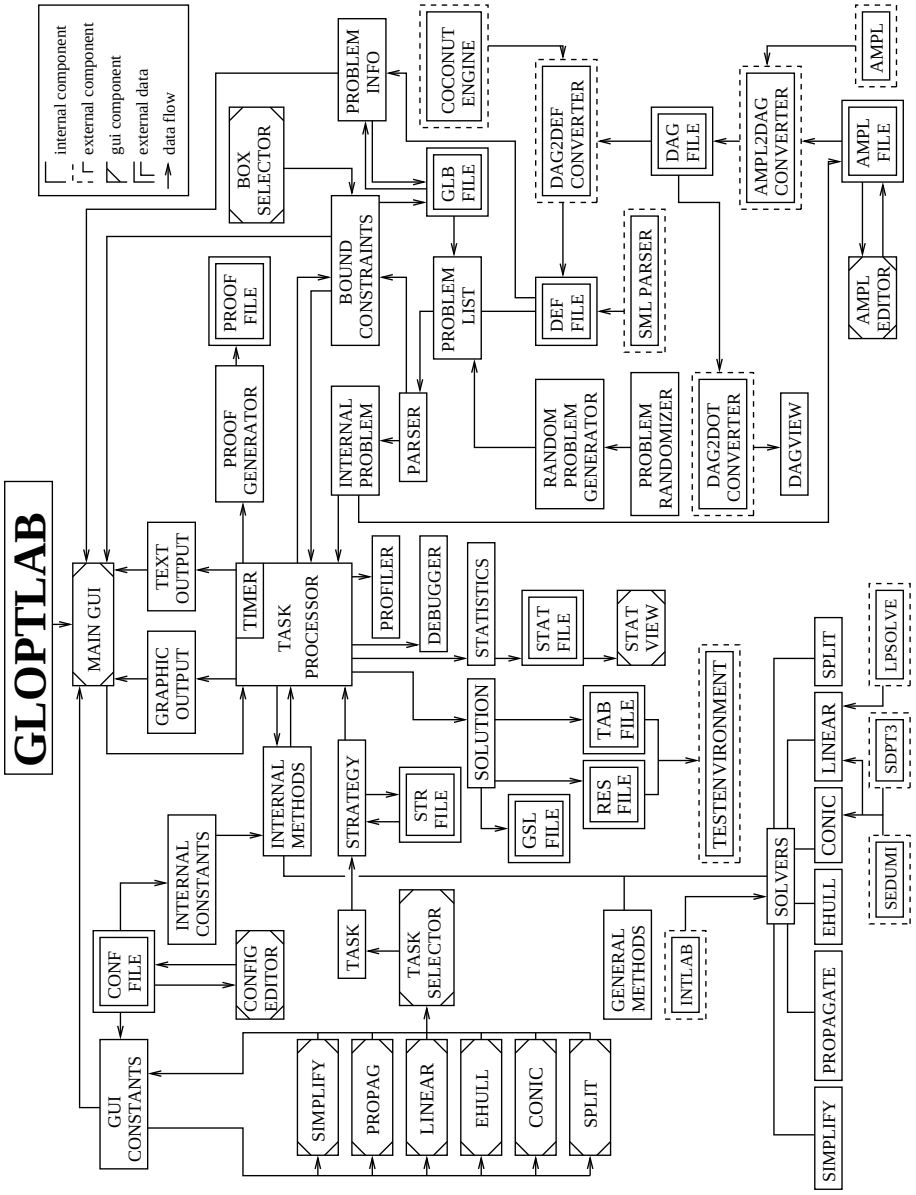

FIGURE 2.1: GloptLab structure

either directly using the GUI, or they can be saved for later use and for the execution of batch solution jobs.

A simple solution strategy, where all tasks have automatically generated default parameters, looks like:

Strategy 2.1 (simple sample)**.**

```
1: Read Problem
2: Simplify
3: Feasibility
4: Begin Condition
5:    Break
6: End Condition
7: Begin While
8:    Propagate
9:    Feasibility
10:    Begin Condition
11:       Break
12:    End Condition
13: End While
14: Begin Split
15:    Propagate
16:    Feasibility
17:    Begin Condition
18:       Break
19:    End Condition
20: End Split
21: Merge
22: Begin Postprocess
23:    Merge
24:    Feasibility
25: End Postprocess
26: Pause
27: Finish
```

while a more sophisticated one is:

Strategy 2.2 (complex sample)**.**

```
1: Read Problem
2: Simplify
3: Ehull
4: Linear
5: Feasibility
6: Begin Condition
7:    Break
8: End Condition
9: Conic
10: Begin While
11:    Propagate
12:    Linear
13:    Feasibility
14:    Begin Condition
15:       Break
16:    End Condition
17: End While
18: Begin Split
19:    Propagate
20:    Linear
21:    Feasibility
22:    Begin Condition
23:       Break
24:    End Condition
25: End Split
26: Merge
27: Begin Split
28:    Propagate
29:    Linear
30:    Feasibility
31:    Begin Condition
32:       Break
33:    End Condition
34: End Split
35: Begin Postprocess
36:    Merge
37:    Feasibility
38: End Postprocess
39: Pause
40: Finish
```

Each method may have several input parameters (which are omitted in the above strategies), all of which have default values. For example, the while loop starts with BEGIN WHILE and ends with END WHILE and has, as parameters, the minimal gain percentage MINGAIN, the maximum number of iteration MAXITER and the width of a small box SMALL. The special parameters PRT and DEB can be set for every method and determine the level of the text and the debugging output. For more details on the various parameters see the tables at the end of the sections describing the different methods.
The input parameters depend on the implementation of the corresponding task, the definition of which must include their documentations. Therefore older strategies may become invalid if a task description has been changed. If strategies are built and updated using the GLOPTLAB graphical user interface, this is automatically recognized and the invalid lines are flagged for correction.

2.4.3 User defined methods

The different methods are integrated into GLOPTLAB in a uniform way such that the repertoire of methods can be extended easily. New functions can be written for each task, including those presented in Section 2.2 (constraint propagation, linear methods, conic methods, branch and bound), without knowledge of the GLOPTLAB code. For example if someone creates a new linear method it is automatically recognized by GLOPTLAB as such if it is placed into the `Gloptlab/Source/UserDefined/` directory as an `m`-file called `gllinear_*.m` and can be selected in the strategy generator as one of the options for linear methods. Samples for user defined methods in each class can be found in the `Gloptlab/Source/UserDefined/` directory.

Each method accesses the problem in either the inequality representation (1.6) or the equality representation (1.7) (which are easily converted into each other) together with a number of additional control parameters (maximal depth, linear solver name, etc.) specific for each category. The results returned by the methods may consist of new bound constraints, found feasible points, linear relaxations, new general constraints, etc. The writer of the methods must ensure for all rigorous tasks that the results are indeed rigorous.

2.4.4 Graphical user interface

The *graphical user interface* of GLOPTLAB consists of areas for entering problems, for defining strategies, for displaying the solver progress and for configuring GLOPTLAB (see Figure 2.2).

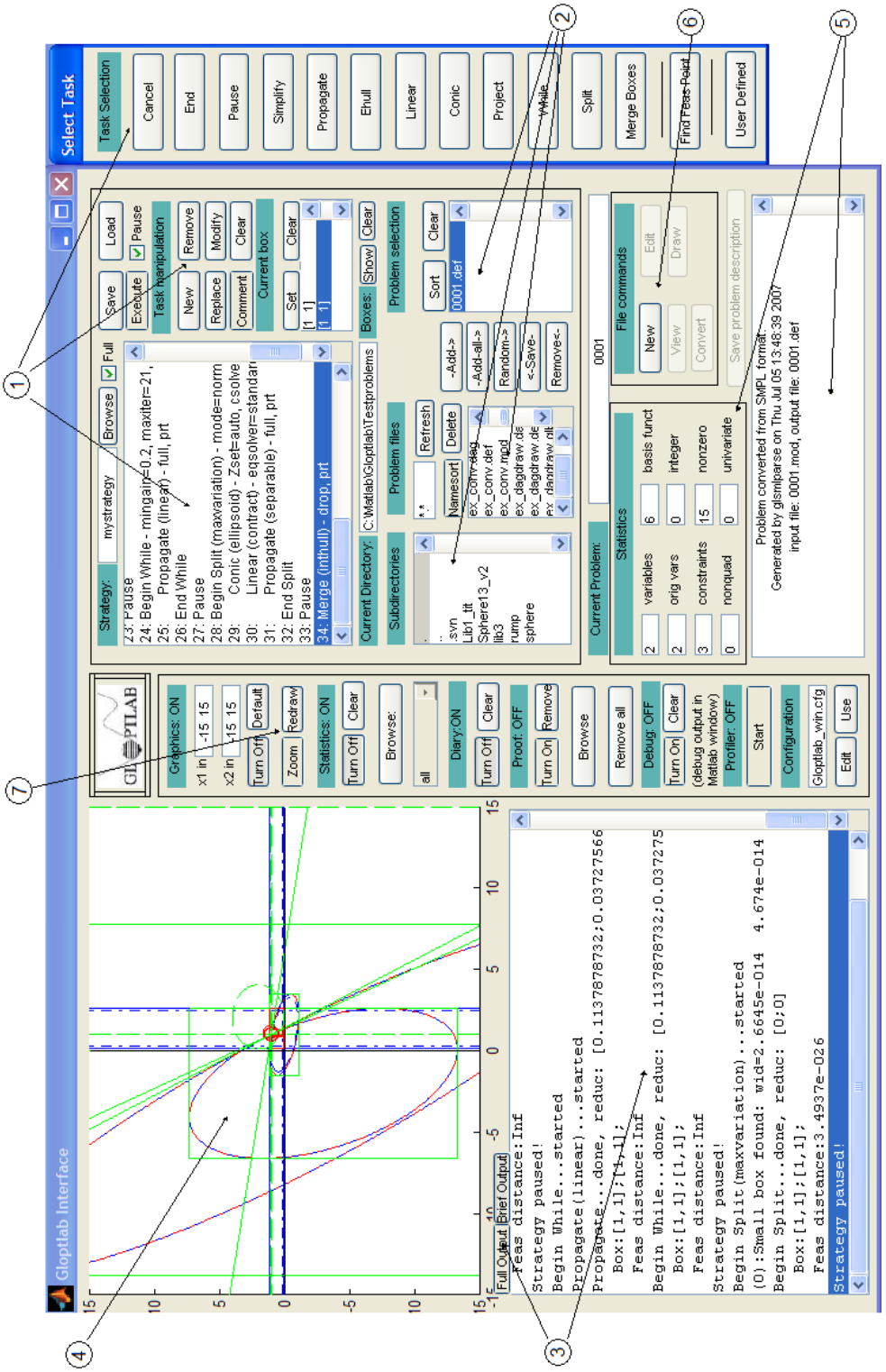

FIGURE 2.2: Gloptlab GUI; for explanations screenshot the text.

Building a strategy in the graphical user interface is done by inserting, editing or removing tasks using the strategy builder (marked 1 in Figure 2.2). In the graphical user

interface not only the strategies can be edited but a single problem or a *problem list* (marked 2 in Figure 2.2) can be solved by executing a strategy using the EXECUTE button. The text output of the solution procedure can be found in the text output window (marked 3 in Figure 2.2), while the graphical output for problems of every dimension is found in the graphical output window (marked 4 in Figure 2.2). Important information for the currently selected problem (name, number of variables and constraints etc.) can be viewed in the right lower part (marked 5 in Figure 2.2). Creating new problems or converting existing ones into the GLOPTLAB format is also possible with the conversion tools and by using the internal GLOPTLAB editor. They can be accessed from the panel marked with 6 in Figure 2.2. In the central long panel (marked 7 in Figure 2.2) the parameters for the graphical output, the statistical database, the automatically generated proofs, the profiler and the general configuration can be accessed and modified. The GLOPTLAB configuration consists of several global parameters, like the path of the external solvers or the width of a box which is assumed as tiny, and all the default values of the parameters used in the different task. There can be different configuration files, and the parameters contained in them can be edited by the user.

2.4.5 Batch solution

Although GLOPTLAB can be completely controlled by using the graphical user interface, the latter is only an additional layer built on the GLOPTLAB core and not essential for using the software. Alternatively, it is possible to solve one or more problems with a selected strategy by using the UNIX `GloptSolve` or the MATLAB `GloptSolve.m` scripts.

GLOPTLAB can generate autosave files (`.sav`), solution files in the GLOPTLAB format (`.gls`) and `.res` files as well. The latter is needed for the TEST ENVIRONMENT [28], which allows one to compare the results and the performance of GLOPTLAB with other solvers.

2.4.6 Notes

The current version of GLOPTLAB can be obtained at the official GLOPTLAB homepage: `http://www.mat.univie.ac.at/~dferi/gloptlab.html`.

2.5 Examples

The comparison of the performance to non-rigorous solvers which are implemented in a compiled language like C++ is difficult. We also emphasize that the strength of

GloptLab lies in the easy use, extendability, the interactive solution and finding all solutions of a problem and not in outperforming non rigorous solvers by solving a whole test set of problems using a single default strategy. However we use the following two dimensional example to demonstrate one of the advantages of GloptLab: the quadratic constraint satisfaction problem

$$\begin{aligned} -3x_1^2 + x_2x_1 + x_2^2 &= -2 \\ x_1^2 + 3x_1x_2 - 3x_2^2 &= 10 \end{aligned} \tag{2.7}$$

has no solution. The graph of (2.7) generated by the graphical user interface of GloptLab can be found in Figure 2.3.

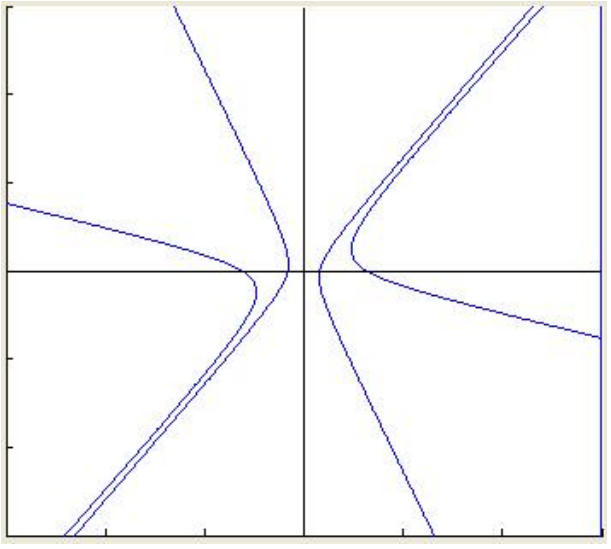

Figure 2.3: Two dimensional example consisting of two equality constraints.

We tested some state of the art solvers by using the NEOS Server (see Czyzyk et al. [24]) and obtained following results:

- The global solver Baron found the problem infeasible after completing 41 iteration steps in approximately 0.3 seconds. However the message

  ```
  User did not provide appropriate variable bounds.
  We may not be able to guarantee globality.
  ```

  is hidden in the log file returned by the solver. Thus, we tried to set artificial bounds, and when we used $-10^4 \leq x_1, x_2 \leq 10^4$ this message disappeared, showing that Baron cannot cope with unbounded bound constraints.

- The local solver Knitro returned after 35 major iterations and 178 function evaluations the message:

```
EXIT: Convergence to an infeasible point.
Problem appears to be locally infeasible.
If problem is believed to be feasible, try multistart to search
for feasible points.
```

- The rigorous global solver ICOS modified the problem by adding the artificially set bounds of $-10^8 \leq x_1, x_2 \leq 10^8$. This happened without additional warning. It found the modified problem infeasible after 145 splits. The execution time was 4.38 seconds.
- We used GloptLab with the solution strategy in Subsection 2.4.2, and verified infeasiblity in 0.860 seconds. GloptLab did not set any artificial bounds on the variables, and needed no branching since the conic ellipsoid enclosure verified that the problem is infeasible.

2.6 Some test results

In this section we present some promising test results of GloptLab. The LaTeX tables containing the results are automatically generated by the Test Environment [28], which we used for checking the solutions for correctness. We tested GloptLab on the library Lib3 of the Coconut Environment Testset (see Shcherbina et al. [91]), containing 308 constraint satisfaction problems. We solved the problems by using the sample strategies 2.1 and 2.2. These strategies are configured not to accept problems containing non-algebraic functions or more than 100 variables. The maximal time allowed for the solution of a single problem was 120 seconds.

Gloptlab on `Lib3` using the strategy (2.1)								
stat	all	wr	easy location			hard location		
			+G	-G	I	+G	-G	I
all	308	0	121	124	0	14	29	20
G	125	0	111	0	0	14	0	0
X	76	0	0	57	0	0	8	11
TU	95	0	8	59	0	0	21	7
U	12	0	2	8	0	0	0	2

Gloptlab summary statistics						
lib	all	accept	+G	G!	G?	I?
Lib3	308	232	135	125	0	0

Gloptlab on `Lib3` using the strategy (2.2)								
stat	all	wr	easy location			hard location		
			+G	-G	I	+G	-G	I
all	308	0	130	115	0	19	24	20
G	139	0	120	0	0	19	0	0
X	76	0	0	57	0	0	8	11
TU	85	0	10	52	0	0	16	7
U	8	0	0	6	0	0	0	2

Gloptlab summary statistics						
lib	all	accept	+G	G!	G?	I?
Lib3	308	232	149	139	0	0

Table legend: `stat` - solution status; `all` - the number of problems given to the solver; `accept` - problems accepted by the solver; `wr` - number of wrong claims (the sum of `G?` and `I?`); `easy location` - problems which have been classified as easy; `hard location` - problems which have been classified as hard. Status codes: `G` - the result claimed to be a global optimizer; `+G` - a global solution was found; `-G` - no global solution was found; `G!` - correctly claimed global solution; `G?` - wrongly claimed global solution, `I` - infeasible problem; `I?` - wrongly claimed infeasibility, `L` - local solution found; `TL` - timeout reached and a local solution was found; `U` - unresolved (no solution found or error message); `X` - model not accepted by the solver.

The tables show that from the 232 accepted problems we have found 135 correct solution (125 of them was claimed as correct) by using the first strategy and 149 correct solution (139 of them was claimed as correct) by using the second one. Within the same allowed solution time we solved 14 more problems with the second strategy as with the first one. This is approximately 10 percent of the accepted problems, and one third of them was a problem which is classified as a hard one. Indeed; 35 percent more of the hard problems was solved with using the second strategy. This significant difference was caused by the more sophisticated methods and the clever structure of the second strategy. The results show the importance of building a good strategy, as well as the process of testing different methods as the part of a strategy.

2.7 Conclusion and perspectives

Apart from the actual methods implemented in GloptLab, the major innovation is the ease with which it is possible to write strategies, to extend the method repertoire, and

to test selected methods on selected test sets as part of a strategy. In GloptLab users can build, test and optimize their own strategies, they can store and easily share them with other people. Moreover, users can implement and test their own methods without the need of extensive knowledge of the GloptLab implementation itself. The graphical representation of the solution process greatly simplifies the identification of the weak points of a method or a strategy.

Future work on GloptLab in our research group in Vienna includes porting the most useful methods and strategies to the Coconut Environment to increase the execution speed. Since the graphical layer is separated from the main solution engine, exporting parts of the implementation to other programs and the conversion to other programming languages should be easy. We also plan to add further methods to the method repertoire, and to search for optimal solution strategies. One of our goals is to develop an automatic strategy selection, which adapts the strategy to the problem solved. Numerous other features like generating human readable proofs and automatically building statistical databases, already available in a rudimentary form, will be fully developed in the future.

We also intend to extend GloptLab to rigorously solve non-quadratic optimization problems. As discussed in the problem specification, the internal problem representation of GloptLab allows non-quadratic, univariate functions. A converter from a general optimization problem in AMPL format to the internal format is already implemented.

External contributors are welcome to join the project by implementing and testing their own user-defined methods. User-defined methods submitted to us will be permanently added to the method repertoire of future versions of GloptLab if they are promising enough.

Remark: The idea of creating GloptLab came from Arnold Neumaier, the structure, features and implementation is my own work.

Chapter 3

Constraint propagation on quadratic constraints

Abstract. This chapter considers constraint propagation methods for continuous constraint satisfaction problems consisting of linear and quadratic constraints. All methods can be applied after suitable preprocessing to arbitrary algebraic constraints.

The basic new techniques consist in eliminating bilinear entries from a quadratic constraint, and solving the resulting separable quadratic constraints by means of a sequence of univariate quadratic problems. Care is taken to ensure that all methods correctly account for rounding errors in the computations.

Various tests and examples illustrate the advantage of the presented method.

Keywords. Constraint propagation, constraint programming, continuous constraints, quadratic constraint satisfaction problems, rounding error control, verified computation, quadratic programming, constrained optimization.

3.1 Introduction

Context. This chapter contributes some new solution techniques for continuous constraint satisfaction problems. A constraint satisfaction problem is the task of finding one or all points satisfying a given family of equations and/or inequalities, called constraints. Many real word problems are continuous constraint satisfaction problems, often high dimensional ones. Typical applications include robotics Grandon et al. [37], Merlet [66], localization and map building Jaulin [49], Jaulin et al. [50], biomedicine Cruz & Barahona [23], or the protein folding problem Krippahl & Barahona [58].

Solving constrained global optimization problems is typically reduced to solving a sequence of constraint satisfaction problems, each obtained by adding a constraint $f(x) \leq f_{\text{best}}$ to the original constraints, where f is the objective function and f_{best} the function value of the best feasible point found. Thus all techniques for solving constraint satisfaction problems have immediate impact on global optimization (see Neumaier [73]).

Constraint satisfaction problems are solved in practice by a combination of a variety of techniques, almost always involving as key components constraint propagation combined with either some form of stochastic search or a branch and prune scheme for a complete search. These techniques are often complemented by filtering or reduction techniques based on techniques borrowed from optimization, such as convex relaxations Lebbah et al. [60]. For filtering, relaxation, branching and other techniques also see Jaulin [48], Sahinidis & Tawarmalani [85], [60].

Filtering techniques that tighten a box – the Cartesian product of intervals defined by the bounds on the variables – are called *constraint propagation* if they are based on a sequence of steps, each using a single constraint only. *Forward propagation* uses the bound constraints to improve the bounds on the general constraints; *backward propagation* uses the bounds on the general constraints to improve the bounds on the variables. In order to avoid a loss of feasible points, constraint propagation methods are usually implemented with rigorous error control, taking care that all reductions are valid even though the calculations are done with floating-point arithmetic only.

In practice, constraint propagation repeats the reduction of a box by means of a suitably chosen constraint, navigating through the network of constraints connected by the variables, until no further significant reduction takes place. In particular, if the initial search box is unbounded but the feasible domain is bounded, constraint propagation methods may be able to find finite bounds on all variables. Since many methods require finite box constraints, this makes constraint propagation a valuable preprocessing tool.

In a stochastic search procedure, constraint propagation on the initial box may result in a much smaller search domain. In a branch and prune procedure, where a tree of subboxes is generated, constraint propagation may result in a quick elimination of subboxes, or a significant reduction before more complex reduction techniques are applied. This shows that constraint propagation has a wide range of applicability, and is a very useful optimization technique.

Prior work. A number of software packages for solving constraint satisfaction problems make extensive use of constraint propagation. The `Numerica` software Hentenryck

[41], Hentenryck et al. [43] uses branch and prune methods and interval constraint programming to solve constraint satisfaction problems. The `ICOS` solver by Lebbah [59] is a software package for solving nonlinear and continuous constraints, based on constraint programming and interval analysis techniques. `Realpaver` by Granvilliers & Benhamou [38] combines interval methods, with constraint satisfaction techniques to solve systems given by sets of equations or inequality constraints over integer and real variables. Ceberio & Granvilliers [19] solves nonlinear systems by using interval extension and constraint inversion.

The price winning solver `Baron` by Sahinidis & Tawarmalani [86] also uses constraint propagation techniques. Initiated by the development of interval analysis on DAGs (Directed Acyclic Graphs) by Schichl & Neumaier [89], advanced constraint propagation techniques for solving numerical constraint satisfaction problems have been given in Vu et al. [101, 102], which is an efficient implementation of basic constraint propagation algorithms for individual operations. It is included in the global optimization software platform COCONUT Environment [87, 88].

Historically, constraint propagation was pioneered in constraint logic, first for discrete constraints by Cleary [22], later for continuous constraints (Older & Vellino [76], see also [10, 14, 15, 21, 25, 39, 41–43, 45, 53]), but has also forerunners in presolve techniques in mathematical programming (Anderson & Anderson [7], Lodwick [62]). They can be modeled by narrowing Benhamou [11] or chaotic iterations Apt [9], i.e., sequences of application of contracting and monotonic functions on domains. The level of work involved and quality obtained in constraint propagation methods may be characterized by local consistency notions; see Benhamou et al. [12, 13], Jermann et al. [51]. An in-depth treatment of continuous constraint propagation from the point of view of constraint programming can be found in the COCONUT report (Bliek et al. [17]). The global optimization survey of Neumaier [73] also discusses continuous constraint propagation without the need to decompose the constraints into single operations.

Contents. In this chapter, we consider constraint propagation methods for continuous constraint satisfaction problems consisting of linear and quadratic constraints. Care is taken to ensure that all methods correctly account for rounding errors in the computations. We only present techniques for improving the inferences from single constraints. These techniques can be easily combined with our new constraint propagation method. In general constraint propagation on single constraints only is not efficient enough for solving constraint satisfaction problems. As is well known (see, e.g., [60]), future significant improvements can be obtained by using the information from several constraints simultaneously. This can be done in various ways, e.g., by using linear relaxations ([59]),

probing ([99]) or other, new methods in our GloptLab environment (Chapter 2) where all constraint propagation methods introduced in this chapter are implemented, too.

All our methods can be applied after suitable preprocessing to arbitrary algebraic constraints. We can always transform a polynomial constraint to a collection of quadratic constraints by introducing explicit intermediate variables. The same holds for constraints involving roots, provided that we also add nonnegativity constraints to the intermediate variables representing the roots. Rewriting an algebraic constraint satisfaction problem as an equivalent problem with linear and quadratic constraints increases the number of variables and can result in loss of structural information which is used by some constraint propagation techniques but makes possible to apply the methods discussed in this chapter. Of course, all techniques can be applied to the subset of quadratic (or algebraic) constraints in an arbitrary constraint satisfaction problem. However, in practice care should be taken that the dimension of the transformed problem does not exceed the reasonable maximum.

In Section 3.2 we derive rigorous bounds for univariate quadratic expressions, relevant for forward propagation, while in Section 3.3 we find bounds on the arguments in a constraint, relevant for backward propagation. The propagation of separable quadratic constraints (containing no bilinear entries) is discussed in Section 3.4, and in Section 3.5, we give an effective method to bound and eliminate bilinear entries from a constraint. This allows the reduction of the nonseparable constraints to separable ones, leading in Section 3.6 to a constraint propagation method for quadratic constraint satisfaction problems.

3.2 Bounds for univariate quadratic expressions

For use in forward propagation, we derive rigorous bounds for univariate quadratic expressions. We analyze the possible extrema of the expression inside a given interval, and derive the general solution of the problem.

Example 3.1. *Let $f(x)$ denote an univariate quadratic expression, with*

$$f(x) = x^2 - 2x, \quad x \in [-7, 5] := \mathbf{x}. \tag{3.1}$$

We look for the best interval $\mathbf{f}$ such that for all $x \in \mathbf{x}$, $f(x) \in \mathbf{f}$ holds: we find that the minimum of f is attained at $x = 1$ which is inside of the interval $\mathbf{x}$. The maximum of f must be attained at the boundary of $\mathbf{x}$. We have $f(1) = -1$ and the function values on the boundary of $\mathbf{x}$ are 63 and 15. Therefore, we find that $\mathbf{f} = [-1, 63]$.

In general, we want to find a rigorous upper bound on

$$u = \sup\{ax^2 + bx \mid x \in \mathbf{x}\}.$$

We note that $u = \max\{\underline{x}(a\underline{x} + b), \overline{x}(a\overline{x} + b)\}$, except in case that $ax^2 + bx$ attains its global maximum in the interior of $\mathbf{x}$. This is the case iff $a < 0$ and $t = -b/(2a)$ is in the interior of $\mathbf{x}$, in which case $u = b^2/(-4a)$ is attained at t.

If $\underline{x} \geq 0$, we get a rigorous upper bound in finite precision arithmetic by computing with upward rounding as follows (`xl` $= \underline{x}$, `xu` $= \overline{x}$):

Algorithm 3.1 (Rigorous upper bound for a univariate quadratic expression)**.**

```
roundup;
if a == 0,
  u=max(xl*b,xu*b);
else
  u=max(xl*(a*xl+b),xu*(a*xu+b));
  s=b/2; t=s/(-a);
  if t>xl,
    r=(-2*a)*xu;
    if r>b, u=max(u,s*t); end;
  end;
end;
```

With some extra analysis, it could be determined in most cases which of the three cases is the worst case; however, if the unconstrained maximum of the quadratic is very close to a bound (or to both bounds), two (or three) of the cases might apply due to uncertainty caused by rounding errors.

Finding a rigorous enclosure for the interval

$$\mathbf{c} = \{\sup\{ax^2 + bx \mid x \in \mathbf{x}\} \mid a \in \mathbf{a}, b \in \mathbf{b}\}$$

can be reduced to the above for $\underline{x} \geq 0$, using

$$\overline{c} = \sup\{\overline{a}x^2 + \overline{b}x \mid x \in \mathbf{x}\}, \quad \underline{c} = -\sup\{-\underline{a}x^2 - \underline{b}x \mid x \in \mathbf{x}\}.$$

The case $\overline{x} \leq 0$ can be reduced to this by changing the sign of x, and the general case by splitting $\mathbf{x}$ at zero if necessary.

Essentially the same analysis holds for rigorous upper bounds on

$$u = \sup\left\{ \sum_{i=1}^{n} a_i x^i \;\middle|\; x \in \mathbf{x} \right\}$$

and for rigorous enclosures of

$$\mathbf{c} = \sup \left\{ \sum_{i=1}^{n} a_i x^i \;\middle|\; x \in \mathbf{x},\ a \in \mathbf{a} \right\},$$

except that finding the interior extrema is more involved. It can be done with closed formulas for $n \leq 5$ (though already $n = 4$ is quite cumbersome and not recommended). In general, for $n > 3$ we recommend to use a root enclosure algorithm for the derivative, such as that in NEUMAIER [72].

3.3 Solving univariate quadratic expressions

If we have bounds on an univariate quadratic expression, we can find the range for the variable, for which the expression satisfies the given bounds. An in depth analysis of quadratic equations leads to the general solution, relevant for backward propagation.

The present approach, based on directed rounding only, provides an efficient alternative to the interval arithmetic based procedures discussed by DIMITROVA & MARKOV [26, Section 4] and later by HANSEN & WALSTER [40] (who only treat the solution of a quadratic equation with interval coefficients).

Example 3.2. *Let $f(x)$ denote an univariate quadratic expression, with*

$$f(x) = x^2 - 2x \in [-1, 8]. \tag{3.2}$$

We look for the best interval $\mathbf{x}$ such that for all $x \in \mathbf{x}$, (3.2) holds. The inequality

$$x^2 - 2x + 1 = (x - 1)^2 \geq 0,$$

arising from the lower bound, always holds, while the inequality

$$x^2 - 2x - 8 \leq 0,$$

arises from the upper bound. Therefore we find that $\mathbf{x} = [-2, 4]$. We note that, while $\mathbf{x}$ is by definition always an interval, sometimes the set of all x satisfying the constraint may be strictly smaller, containing an interior gap.

In general, we want to find the set

$$X = \{x \geq 0 \mid ax^2 + 2bx \geq c\},$$

and we proceed as follows. If $a = 0$, the constraint is in fact linear, and we have

$$X = \begin{cases} \emptyset & \text{if } b \le 0, c > 0, \\ [0, 0.5c/b] & \text{if } b < 0, c \le 0, \\ [0.5c/b, \infty] & \text{if } b > 0, c \ge 0, \\ [0, \infty] & \text{if } b \ge 0, c \le 0, \end{cases}$$

which can be nested such that only two comparisons are needed in any particular case. For a rigorous enclosure in finite precision arithmetic, rounding must be downwards in the second case, and upwards in the third case.

If $a \neq 0$, the behavior is governed by the zeros of the quadratic equation $ax^2+2bx-c = 0$, given by

$$t_1 = \frac{-b - \sqrt{\Delta}}{a} = \frac{c}{b - \sqrt{\Delta}}, \qquad t_2 = \frac{-b + \sqrt{\Delta}}{a} = \frac{c}{b + \sqrt{\Delta}},$$

where $\Delta := b^2 + ac$. If $\Delta \ge 0$, the zeros are real, and the nonnegative zeros determine

$$X = \begin{cases} [0, \infty] \setminus \,]t_1, t_2[& \text{if } a > 0, \\ [0, \infty] \cap [t_2, t_1] & \text{if } a < 0. \end{cases}$$

Depending on the signs of a, b and c we find

$$X = \begin{cases} \emptyset & \text{if} \quad a < 0, b \le 0, c > 0, & \text{(case 1)} \\ [z/a, \infty] & \text{if} \quad a > 0, b \le 0, c > 0, & \text{(case 2)} \\ [0, -(c/z)] & \text{if} \quad a < 0, b \le 0, c \le 0, & \text{(case 3)} \\ [0, -(c/z)][\cup [z/a, \infty] & \text{if} \quad a > 0, b \le 0, c \le 0, & \text{(case 4)} \\ [-((-c)/z), z/(-a)] & \text{if} \quad a < 0, b \ge 0, c > 0, & \text{(case 5)} \\ [-((-c)/z), \infty] & \text{if} \quad a > 0, b \ge 0, c > 0, & \text{(case 6)} \\ [0, z/(-a)] & \text{if} \quad a < 0, b \ge 0, c \le 0, & \text{(case 7)} \\ [0, \infty] & \text{if} \quad a > 0, b \ge 0, c \le 0, & \text{(case 8)} \end{cases}$$

where

$$z = |b| + \sqrt{\Delta}.$$

These formulas are numerically stable, and can be nested such that only three comparisons are needed in any particular case. (There are avoidable overflow problems for huge $|b|$, which can be cured by using for huge $|b|$ instead of $\sqrt{b^2 + ac}$ the formula $|b|\sqrt{1 + ac/b^2}$.)

Rigorous results in the presence of rounding errors are obtained if lower bounds are rounded downwards, and upper bounds are rounded upwards. With the bracketing as given, this happens if all computations (including those of $\Delta = \sqrt{b^2 + ac}$ and $z =$

$|b| + \sqrt{\Delta}$) are done with rounding upwards if $b \geq 0$, and with rounding downwards if $b \leq 0$. (However, this does *not* hold for the version guarded against overflow, where further care is needed for the directed rounding of $\sqrt{\Delta} = |b|\sqrt{1 + ac/b^2}$.)

If (the exact) Δ is negative, there is no real solution, and X is empty if $c > 0$ and $[0, \infty]$ otherwise. The case when the sign of Δ cannot be determined due to rounding errors needs special consideration. In the first and last case, the conclusion holds independent of the sign of Δ, so that the latter need only be computed for cases 2–7 in the definition of X. In the cases 2, 3, 6, and 7 we have $ac \geq 0$, so that $\Delta \geq 0$ automatically. This leaves cases 4 and 5. Now it is easily checked that with the recommended rounding and, in place of cases 4 and 5,

$$X = \begin{cases} [0, -(c/z)] \cup [z/a, \infty] & \text{if } a > 0,\ b \leq 0,\ c \leq 0,\ \Delta \geq 0, \\ [0, \infty] & \text{if } a > 0,\ b \leq 0,\ c \leq 0,\ \Delta < 0, \\ \emptyset & \text{if } a < 0,\ b \geq 0,\ c > 0,\ \Delta < 0, \\ [-((-c)/z), z/(-a)] & \text{if } a < 0,\ b \geq 0,\ c > 0,\ \Delta \geq 0, \end{cases} \tag{3.3}$$

a rigorous enclosure is computed in all cases. Finding the set

$$X' = \{x \geq 0 \mid ax^2 + 2bx \in \mathbf{c} \text{ for any } a \in \mathbf{a}, b \in \mathbf{b}\}$$

can be reduced to the previous task since

$$X' = \{x \geq 0 \mid \underline{a}x^2 + 2\underline{b}x \leq \overline{c}\} \cap \{x \geq 0 \mid \overline{a}x^2 + 2\overline{b}x \geq \underline{c}\}.$$

The sets

$$X'' = \{x \in \mathbf{x}_0 \mid ax^2 + 2bx \geq c\}$$

and

$$X''' = \{x \in \mathbf{x}_0 \mid ax^2 + 2bx \in \mathbf{c} \text{ for some } a \in \mathbf{a}, b \in \mathbf{b}\}$$

can be obtained by intersecting the result of the above tasks with $\mathbf{x}_0$ if $\underline{x}_0 \geq 0$, by negating x, $\mathbf{x}_0$, and $\mathbf{b}$ if $\overline{x}_0 \leq 0$, and by splitting $\mathbf{x}_0$ at zero if 0 is in the interior of $\mathbf{x}_0$. By modifying the code appropriately, one can also avoid computing roots which can be seen to lie outside $\mathbf{x}_0$.

With minor changes, these formulas also apply for strict inequalities and interior enclosures. Also, it is clear that polynomial inequalities and inclusions of interval polynomials can be solved by a straightforward adaptation of the above arguments.

We end the section with MATLAB code for computing the enclosure

$$X = [\texttt{x1}, \texttt{xu}] \cup [\texttt{x2l}, \texttt{x2u}] \setminus \{\infty\}$$

according to (3.3).

Algorithm 3.2 (Solving an univariate quadratic expression)**.**

```
xl=0;xu=inf;
x2l=inf;x2u=inf;
if b>=0,
  roundup;
  if c>0,
    % case b>=0, c>0
    Delta=b^2+a*c;
    if Delta<0,
      xl=inf;
    elseif a==0 & b==0,
      xl=inf;
    else
      z=b+sqrt(Delta);
      xl=-((-c)/z);
      if a<0, xu=z/(-a); end;
    end
  else
    % case b>=0, c<=0
    if a<0,
      Delta=b^2+a*c;
      z=b+sqrt(Delta);
      xu=z/(-a);
    end;
  end;
else
  rounddn;
  if c>0,
    % case b<0, c>0
    if a>0,
      Delta=b^2+a*c;
      z=-b+sqrt(Delta);
      xl=z/a;
    else
      xl=inf;
    end
  else
    % case b<0, c<=0
    Delta=b^2+a*c;
    if Delta>=0,
      z=-b+sqrt(Delta);
      xu=-(c/z);
      if a>0, x2l=z/a; end;
    end;
  end;
end;
```

3.4 Propagating separable quadratic constraints

We now combine the results of the previous two sections.

Example 3.3. *Let $f(x)$ denote an univariate quadratic expression, with*

$$f(x) := x^2 - 2x \in [-10, 8], \quad x \in [-7, 5]. \tag{3.4}$$

Example 3.1 produced from the bound on x the new bound $f(x) \in [-1, 63]$, hence $f(x) \in [-1, 8]$. Example 3.2 produced from these bounds on f the new bounds $x \in [-2, 4]$. Thus we end up with the new problem

$$f(x) := x^2 - 2x \in [-1, 8], \quad x \in [-2, 4]$$

where the bounds on f and x are tighter than in the original problem (3.4).

This combination of forward and backward propagation for a univariate quadratic expression can be extended without difficulties to a method of constraint propagation for separable quadratic constraints in several variables,

$$\sum_{k=1}^{n} p_k(x_k) \geq c, \quad x \in \mathbf{x}, \tag{3.5}$$

where each term

$$p_k(x_k) := a_k x_k^2 + b_k x_k \tag{3.6}$$

depends on a single variable x_k and may have uncertain coefficients,

$$a_k \in \mathbf{a}_k, \quad b_k \in \mathbf{b}_k.$$

Example 3.4. *We demonstrate separable quadratic constraint propagation on the constraint*

$$-x_1^2 + 2x_1 - x_2 \geq -8 \quad \textit{with } x_1 \in [-7, 5],\ x_2 \in [0, \infty]$$

step by step (see Figure 3.1):

- *We find that for $x_1 \in [-7, 5]$, $x_2 \in [0, \infty]$, $-x_1^2 + 2x_1 \in [-63, 1]$ and $-x_2 \in [-\infty, 0]$ holds.*
- *Therefore, we have $-x_1^2 + 2x_1 - x_2 \leq 1$.*
- *From $0 \geq -x_2$, the inequality $-x_1^2 + 2x_1 \geq -8 - 0 = -8$ follows.*
- *Since $1 \geq -x_1^2 + 2x_1$, the inequality $-x_2 \geq -8 - 1 = -9$ holds.*
- *Then from $-x_1^2 + 2x_1 \geq -8$, $x_1 \in [-2, 4]$, and from $-x_2 \geq -9$, $x_2 \in [-\infty, 9]$ follows.*
- *Finally, we cut the bounds on the variables with the original bounds, and obtain*

$$-x_1^2 + 2x_1 - x_2 \in [-8, 1] \quad \textit{with } x_1 \in [-2, 4],\ x_2 \in [0, 9].$$

For $x_k \in \mathbf{x}_k$ we denote the enclosure of the quadratic univariate term $p_k(x_k)$ by

$$p_k(x_k) \in \mathbf{p}_k = [\underline{p_k}, \overline{p_k}]. \tag{3.7}$$

To find the $\mathbf{p}_k$, if $0 \in \mathbf{x}_k$, we split $\mathbf{x}_k$ at zero into a positive part $\mathbf{x}_k^p$ and a negative part $\mathbf{x}_k^n$, with $\underline{x_k^p} \geq 0$, $\underline{x_k^n} \geq 0$, $\mathbf{x}_k^p \cap -\mathbf{x}_k^n = \{0\}$ and $\mathbf{x}_k^p \cup -\mathbf{x}_k^n = \mathbf{x}_k$. If $\underline{x_k} \geq 0$ then we set $\mathbf{x}_k^p := \mathbf{x}_k$ and $\mathbf{x}_k^n := \emptyset$, while if $\underline{x_k} \geq 0$ then we set $\mathbf{x}_k^p := \emptyset$ and $\mathbf{x}_k^n := -\mathbf{x}_k$. Then we

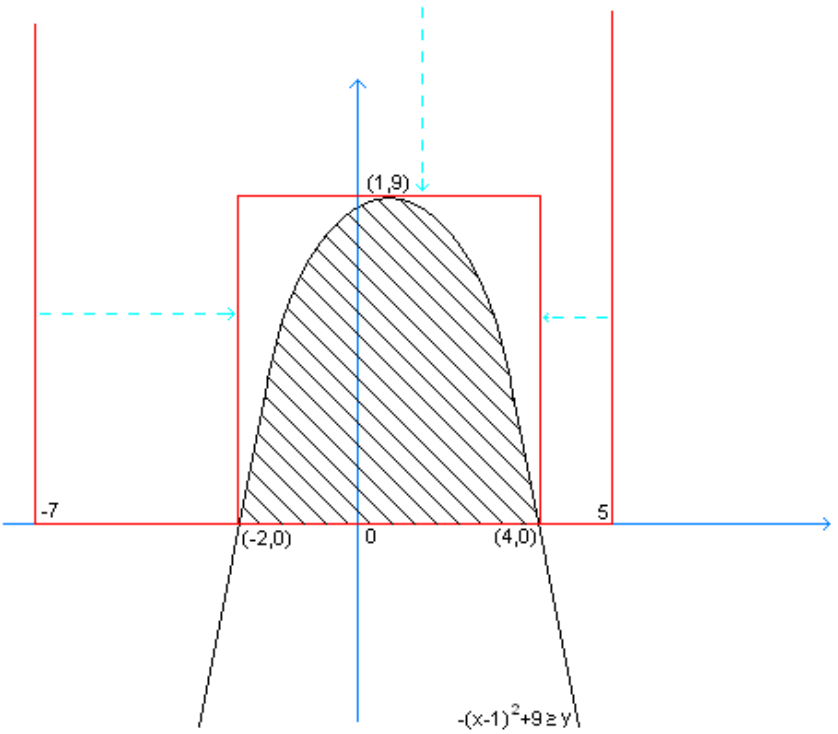

FIGURE 3.1: Improving the bound constraints in Example 3.4.

define the bounds

$$\begin{aligned} \overline{c}_k^p &:= \sup\{\overline{a}x_k^2 + \overline{b}x_k \mid x_k \in \mathbf{x}_k^p\}, \quad \underline{c}_k^p := -\sup\{-\underline{a}x_k^2 - \underline{b}x_k \mid x_k \in \mathbf{x}_k^p\}, \\ \overline{c}_k^n &:= \sup\{\overline{a}x_k^2 - \overline{b}x_k \mid x_k \in \mathbf{x}_k^n\}, \quad \underline{c}_k^n := -\sup\{-\underline{a}x_k^2 + \underline{b}x_k \mid x_k \in \mathbf{x}_k^n\}. \end{aligned} \tag{3.8}$$

Each bound $\overline{c}_k^p$, $\underline{c}_k^p$, $\overline{c}_k^n$, $\underline{c}_k^n$ in (3.8) can be found by applying the results of Section 3.2 to them separately. Then the enclosure of $p_k(x_k)$ is

$$\begin{aligned} \mathbf{p}_k &= [\min(\underline{c}_k^p, \underline{c}_k^n), \max(\overline{c}_k^p, \overline{c}_k^n)] \quad \text{if} \quad 0 \in \mathbf{x}_k, \\ \mathbf{p}_k &= \mathbf{c}_k^p \quad \text{if} \quad \underline{x}_k \geq 0, \\ \mathbf{p}_k &= \mathbf{c}_k^n \quad \text{if} \quad \overline{x}_k < 0. \end{aligned}$$

Then we sum the found intervals and obtain

$$\sum_{k=1}^{n} p_k(x_k) \in \mathbf{e} \text{ with } \mathbf{e} := \sum_{k=1}^{n} \mathbf{p}_k = \Big[\sum_{k=1}^{n} \underline{p}_k, \sum_{k=1}^{n} \overline{p}_k\Big]. \tag{3.9}$$

We can then use the bounds $\mathbf{p}_k$ and $\mathbf{e}$ to check the consistency of the constraint and obtain a new bound for it (forward propagation), and to get better box constraints (backward propagation).

Forward propagation. By (3.5) and (3.9), we have

$$\overline{e} \geq \sum_{k=1}^{n} p_k(x_k) \geq c. \tag{3.10}$$

Therefore, if (3.10) does not hold, the constraint is inconsistent. Following this, we pose the *inconsistency condition*:

$$\text{If } \overline{e} - c < 0 \text{ then the constraint (3.5) is inconsistent.} \tag{3.11}$$

If the constraint is consistent, by (3.5) and (3.9) both

$$\sum_{k=1}^{n} p_k(x_k) \geq c \text{ and } \sum_{k=1}^{n} p_k(x_k) \geq \underline{e}$$

are satisfied, giving us the combined lower bound

$$\sum_{k=1}^{n} p_k(x_k) \geq c' := \max(c, \underline{e}). \tag{3.12}$$

Backward propagation. For all $i \in \{1, \ldots, n\}$ and all $k \neq i$, by (3.7) and (3.12) we obtain

$$\sum_{k=1,\ k\neq i}^{n} \overline{p}_k + p_i(x_i) \geq \sum_{k=1,\ k\neq i}^{n} p_k(x_k) + p_i(x_i) \geq c'.$$

Bringing the upper bounds on the $p_k(x_i)$ to the left hand side and by (3.6) we get

$$a_i x_i^2 + b_i x_i \geq -\gamma_i, \quad \text{where } \gamma_i := \sum_{k\neq i} \overline{p}_k - c'. \tag{3.13}$$

The arrangement of the operations is such that upward rounding still gives correct results. Since the above approximation must be done for each univariate term in the constraint, time can be saved when $n > 2$ by avoiding unnecessary work in the summations. The paper by Dallwig et al. [25] proposes to remove the summations completely, using instead the identity

$$\gamma_i = \overline{e} - c' - \overline{p}_i. \tag{3.14}$$

Again, the arrangement of the operations is such that upward rounding still gives correct results. While fast, this identity must be used with caution: If $\overline{p}_i$ is infinite, $\gamma_i = c' - \infty + \infty$ is undefined. And if $|\overline{p}_i|$ is very large, cancellation (together with the always necessary directed rounding) may lead to unnecessarily pessimistic bounds. Below we give some examples which demonstrate this behavior. To eliminate these problems, we

recommend the use of the formulas

$$\gamma_i = \begin{cases} \gamma' + p_{min} & \text{if} \quad p_i = p_{min} = \infty, \\ \gamma' + p_{min} - e & \text{if} \quad d + e > 0, \\ \gamma' + p_{max} - d & \text{if} \quad d + e \le 0, \end{cases} \tag{3.15}$$

where $\underline{i}, \overline{i}$ are distinct indices with $p_{\underline{i}} = p_{min}, p_{\overline{i}} = p_{max}$,

$$\gamma' := \sum_{k \neq \underline{i}, \overline{i}} \overline{p}_k - c',$$

with nonnegative numbers $d := \overline{p}_i - p_{min}$ and $e := \overline{p}_i - p_{max}$. Again, the arrangement of the operations is such that upward rounding still gives correct results.

Remark 3.1. Alternatively we could rewrite (3.6) as

$$p_k(x_k) = a_k(x_k + b_k/2a_k)^2 - b_k^2/4a_k$$

and use interval arithmetic to enclose ranges (forward propagation) and bounds on x (backward propagation) by using this form. However this would only work if $0 \notin \mathbf{a}_k$ and even in this case it yields non-optimal bounds if $\mathbf{a}_k$ is a proper interval and $\mathbf{b}_k$ is not zero.

Example 3.5. *We denote the sum γ_i from (3.13) as γ_i', the sum from (3.14) as γ_i'' and the sum from (3.15) as γ_i'''. We give three examples, one for each of the above three possibilities and put $c' = 0$. For simplicity, we perform all calculations with 16 digit decimal arithmetic, doing the sums from left to right.*

Case 1: For $\overline{p} = (1, 1, \infty)$, we get

$$\begin{aligned} \gamma_3' &= 1 + 1 = 2, \\ \gamma_3'' &= (1 + 1 + \infty) - \infty = NaN, \\ \gamma_3''' &= 1 + 1 = 2. \end{aligned}$$

Case 2: For $\overline{p} = (8 \cdot 10^{-8}, 9 \cdot 10^{-8}, 10^9)$, we get

$$\begin{aligned} \gamma_3' &= 8 \cdot 10^{-8} + 9 \cdot 10^{-8} = 1.7 \cdot 10^{-7}, \\ \gamma_3'' &= (8 \cdot 10^{-8} + 9 \cdot 10^{-8} + 10^9) - 10^9 = 0, \\ \gamma_3''' &= 9 \cdot 10^{-8} + 8 \cdot 10^{-8} - (10^9 - 10^9) = 1.7 \cdot 10^{-7}. \end{aligned}$$

Case 3: For $\overline{p} = (-10^9, 8 \cdot 10^{-8}, 9 \cdot 10^{-8}, 10^9)$*, we get*

$$\begin{aligned}\gamma_3' &= -10^9 + 8 \cdot 10^{-8} + 10^9 = 10^{-6},\\ \gamma_3'' &= (-10^9 + 8 \cdot 10^{-8} + 9 \cdot 10^{-8} + 10^9) - 9 \cdot 10^{-8} = -9 \cdot 10^{-8},\\ \gamma_3''' &= (8 \cdot 10^{-8} + 9 \cdot 10^{-8}) + 10^9 - (9 \cdot 10^{-8} + 10^9) = 0.\end{aligned}$$

In the two first cases our formula (3.15) *reproduces* (3.13)*, while the formula* (3.14) *fails in the first case, and suffers from severe cancellation in the second case. In the third case all formulas suffer from cancellation.*

Since (3.13) is a univariate, quadratic expression, the results of Section 3.3 can be applied. This may result in an improved bound $\mathbf{x}_i'$ on the variable x_i. If we cut it with the original bound on x_i we obtain $x_i \in \mathbf{x}_i' \cap \mathbf{x}_i$. Since we approximate all univariate expression $p_i(x_i)$, $i \in \{1, \ldots, n\}$, we obtain the new bound constraints

$$x \in \mathbf{x}' \cap \mathbf{x}.$$

In general, separable quadratic constraints can be written as

$$\sum_{k=1}^{n} p_k(x_k) \in \mathbf{c}, \tag{3.16}$$

since they have both a lower bound $\underline{c}$ and an upper bound $\overline{c}$. The inequalities

$$\sum_{k=1}^{n} p_k(x_k) \geq \underline{c} \text{ and } \sum_{k=1}^{n} -p_k(x_k) \geq -\overline{c},$$

represent (3.16), and for them all the results of this section can be applied.

3.5 Nonseparable quadratic constraints

In this section we discuss a method for removing a bilinear term from nonseparable quadratic constraints. This is important since by removing all bilinear terms in turn, the problem is transformed to a separable one, to which the results of the previous sections can be applied.

Example 3.6. *(i) For some positive constant* k*, we consider the quadratic constraint*

$$f(x) := kx_1^2 + kx_2^2 + kx_3^2 + 2x_1x_2 + 2x_1x_3 + 2x_2x_3 \leq 1. \tag{3.17}$$

This constraint defines a bounded ellipsoid when $k > 1$*, while for* $k \leq 1$*, an unbounded domain results. Indeed, the Hessian*

$$G = 2 \begin{pmatrix} k & 1 & 1 \\ 1 & k & 1 \\ 1 & 1 & k \end{pmatrix}$$

has the principal sub–determinants $G_{11} = 2k > 0$*,* $\det G_{1:2,1:2} = 2^2(k^2 - 1)$ *and* $\det G = 8(k^3 - 3k + 2) = 2^3(k-1)^2(k+2)$*; hence* G *is positive definite iff* $k > 1$*.*

If we rewrite $f(x)$ *as*

$$f(x) = (k-2)(x_1^2 + x_2^2 + x_3^2) + (x_1 + x_2)^2 + (x_1 + x_3)^2 + (x_2 + x_3)^2$$

and drop the (nonnegative) quadratic terms, we find the separable quadratic inequality

$$(k-2)(x_1^2 + x_2^2 + x_3^2) \leq 1. \tag{3.18}$$

For $k = 3$*, we find* $x_1^2 + x_2^2 + x_3^2 \leq 1$*, and our separable constraint propagation gives the bounds* $x_i \in [-1, 1]$ *for* $i = 1, \ldots, 3$*. Similarly, any* $k > 2$ *leads to a finite box, which gets arbitrarily large as* k *tends to 2. However, for any value of* $k \leq 2$*, (3.18) is a trivial, non–informative inequality. On the other hand, we have seen that the original constraint (3.18) defines a bounded domain when* $k > 1$*. Thus, for* $k \in]1, 2]$*, the above method of eliminating bilinear terms is not able to exploit the full power of (3.18).*

In Chapter 4, we describe the ellipsoid hull technique, which always yields optimal bounds based on more expensive (and much more difficult to rigorously analyze) linear algebra. For example, when $k = 2$*, we get the finite bounds* $x_i \in [-0.86606, 0.86606]$ *for* $i = 1, \ldots, 3$*.*

(ii) If we add the bound constraints $\mathbf{x}_i = [-1, 5]$*,* $i = 1, \ldots, 3$ *to the constraint (3.17) and set* $k = 2$*, we can approximate the bilinear terms* $x_j x_k$ *by the interval evaluation of* $\mathbf{x}_j \mathbf{x}_k$ *and obtain* $2x_j x_k \in [-10, 50]$*. In this case (3.17) reduces to*

$$2x_1^2 + 2x_2^2 + 2x_3^2 \leq 31. \tag{3.19}$$

Since $x_k \in [-1, 5]$*, we find that* $x_k^2 \in \mathbf{p}_k := [0, 25]$ *for all* $k = 1, \ldots, 3$*. Therefore for all* $i = 1, \ldots, 3$

$$2x_i^2 \leq (31 - \sum_{k \neq i} \underline{p}_k) = 31$$

holds, yielding the bound $x_i \leq \sqrt{31/2} \leq 3.94$*.*

(iii) Suppose we have the same bound constraints as in (ii). We approximate the bilinear terms by linear and constant ones. Since $x_i \in [-1, 5]$ *for each bilinear term* $2x_jx_k$ *the inequality*

$$2x_j + 2x_k - 26 \leq 2x_jx_k,$$

holds. Therefore by (3.17) *we obtain*

$$kx_1^2 + kx_2^2 + kx_3^2 + 4x_1 + 4x_2 + 4x_3 \leq 79. \tag{3.20}$$

Since $x_k \in [-1, 5]$*, we find that* $2x_k^2 + 4x_k \in \mathbf{p}_k := [-2, 70]$ *for all* $k = 1, \ldots, 3$*. Then for all* $i = 1, \ldots, 3$

$$2x_i^2 + 4x_i \leq (79 - \sum_{k \neq i} \underline{p}_k) = 83$$

holds, yielding the bound $x_i \leq -1 + \sqrt{85/2} \leq 5.52$*.*

The example shows that approximating the bilinear entries in different ways can lead to different results.

We now formalize and extend the methods used in the preceding example. We consider an arbitrary multivariate quadratic inequality constraint, which we write without loss of generality in the form

$$\sum_k (a_k x_k^2 + b_k x_k) + \sum_{\substack{j,k \\ j>k}} b_{jk} x_j x_k \geq c, \quad x \in \mathbf{x}, \tag{3.21}$$

where the $a_k x_k^2$ are the quadratic, the $b_k x_k$ the linear and $b_{jk} x_j x_k$ the bilinear terms.

3.5.1 Approximation by constants

As in Example 3.6 (ii), we find rigorous bounds for a bilinear term $b_{jk} x_j x_k$ when x_j and x_k are bounded. We evaluate $\mathbf{x}_i \mathbf{x}_j$ by using the rule for multiplying the intervals $\mathbf{x}_i$ and $\mathbf{x}_j$ (see, e.g., Neumaier [70]) and obtain

$$b_{jk}\mathbf{x}_i\mathbf{x}_j \leq \begin{cases} b_{jk} \min\left(\underline{\mathbf{x}}_i\underline{\mathbf{x}}_j, \underline{\mathbf{x}}_i\overline{\mathbf{x}}_j, \overline{\mathbf{x}}_i\underline{\mathbf{x}}_j, \overline{\mathbf{x}}_i\overline{\mathbf{x}}_j\right) & \text{if } b_{jk} < 0 \\ b_{jk} \max\left(\underline{\mathbf{x}}_i\underline{\mathbf{x}}_j, \underline{\mathbf{x}}_i\overline{\mathbf{x}}_j, \overline{\mathbf{x}}_i\underline{\mathbf{x}}_j, \overline{\mathbf{x}}_i\overline{\mathbf{x}}_j\right) & \text{if } b_{jk} \geq 0. \end{cases} \tag{3.22}$$

Directed rounding when evaluating the right hand side ensures that no feasible points can be lost during this process.

3.5.2 Approximation by linear terms

As in Example 3.6 (iii), we approximate each bilinear term $b_{jk}x_jx_k$ with $x_j \in \mathbf{x}_j$ and $x_k \in \mathbf{x}_k$, by the linear expression

$$q_{jk}(x_{jk}) := g_{jk}x_j + e_{jk}x_k + h_{jk}. \tag{3.23}$$

If for all $x \in \mathbf{x}$ the inequality

$$g_{jk}x_j + e_{jk}x_k + h_{jk} \geq b_{jk}x_jx_k, \tag{3.24}$$

holds, by (3.21) we get

$$\sum_k (a_kx_k^2 + b_kx_k) + \sum_{\substack{j,k \\ j>k}} (g_{jk}x_j + e_{jk}x_k + h_{jk}) \geq \sum_k (a_kx_k^2 + b_kx_k) + \sum_{\substack{j,k \\ j>k}} b_{jk}x_jx_k \geq c,$$

for all $x \in \mathbf{x}$. In order to get an optimal $q_{jk}(x_{jk})$ we set

$$g_{jk} := b_{jk}z_k, \quad e_{jk} := b_{jk}z_j$$

for some $z_k \in \mathbf{x}_k$. By (3.24) term h_{jk} can be obtained by finding the upper bound of

$$f(x_j, x_k) := b_{jk}x_jx_k - (e_{jk}x_k + g_{jk}x_j) = (b_{jk}x_j - e_{jk})x_k - g_{jk}x_j \tag{3.25}$$

for $x_k \in \mathbf{x}_k$ and $x_j \in \mathbf{x}_j$. There, again the direct interval evaluation of (3.25) can be used, but to save computational time we propose:

Proposition 3.2. *Let $f(x, y)$ be monotone in x and y, and suppose that $x \in \mathbf{x}$ and $y \in \mathbf{y}$. Then*

$$\square\{f(x,y) \mid x \in \mathbf{x},\ y \in \mathbf{y}\} = \square\{f(\underline{x},\underline{y}) \cup f(\overline{x},\underline{y}) \cup f(\underline{x},\overline{y}) \cup f(\overline{x},\overline{y})\}$$

holds.

Proof.

$$\begin{aligned}
&\square\{f(x,y) \mid x \in \mathbf{x},\ y \in \mathbf{y}\} = \square\{(\square\{f(x,y) \mid x \in \mathbf{x}\}) \mid y \in \mathbf{y}\} \\
&= \square\{(\square\{f(\underline{x},y) \cup f(\overline{x},y)\}) \mid y \in \mathbf{y}\} = \square\{\square\{f(\underline{x},y) \mid y \in \mathbf{y}\} \cup \square\{f(\overline{x},y) \mid y \in \mathbf{y}\}\} \\
&= \square\{f(\underline{x},\mathbf{y}) \cup f(\overline{x},\mathbf{y})\} = \square\{f(\underline{x},\underline{y}) \cup f(\overline{x},\underline{y}) \cup f(\underline{x},\overline{y}) \cup f(\overline{x},\overline{y})\}
\end{aligned} \tag{3.26}$$

holds. □

In floating point arithmetics, we have to ensure correct upward rounding and therefore we compute

$$\begin{aligned} u_1 &= \Delta f(\underline{x},\underline{y}), \quad u_2 = \Delta f(\overline{x},\underline{y}), \quad u_3 = \Delta f(\underline{x},\overline{y}), \quad u_4 = \Delta f(\overline{x},\overline{y}), \\ l_1 &= \nabla f(\underline{x},\underline{y}), \quad l_2 = \nabla f(\overline{x},\underline{y}), \quad l_3 = \nabla f(\underline{x},\overline{y}), \quad l_4 = \nabla f(\overline{x},\overline{y}). \end{aligned}$$

and obtain

$$\square\{f(x,y) \mid x \in \mathbf{x},\ y \in \mathbf{y}\} = [\min_i l_i,\ \max_i u_i].$$

Applied to (3.25), noting that $f(x,y)$ is monotone in both x and y, we find the upper bound

$$h_{jk} = \max_i u_i$$

with

$$\begin{aligned} u_1 &= \Delta b_{jk}((\underline{x}_j - z_j)\underline{x}_k - z_k\underline{x}_j), \quad u_2 = \Delta b_{jk}((\overline{x}_j - z_j)\underline{x}_k - z_k\overline{x}_j), \\ u_3 &= \Delta b_{jk}((\underline{x}_j - z_j)\overline{x}_k - z_k\underline{x}_j), \quad u_4 = \Delta b_{jk}((\overline{x}_j - z_j)\overline{x}_k - z_k\overline{x}_j). \end{aligned}$$

3.5.3 Approximation by separable quadratic terms

Alternatively, when a bound for x_j or x_k is large, but the coefficients a_j and a_k of the corresponding quadratic terms have negative sign, it is usually better to proceed as in Example 3.6 (i) and relax the bilinear entries by quadratic ones. Note that when the constraint in that example is rewritten in the form (3.21), the coefficients of the quadratic terms become negative. This is a necessary condition for the constraint to lead to a bounded feasible set.

To ensure good scaling behavior, we want to bound $b_{jk}x_jx_k$ by a multiple of $a_jx_j^2 + a_kx_k^2$:

Proposition 3.3. *Suppose that $a_k < 0$ and $a_j < 0$, and put*

$$v_{jk} := \operatorname{sign}(b_{jk})\sqrt{\frac{a_k}{a_j}}, \quad d_{jk} := \frac{b_{jk}}{2v_{jk}}.$$

Then

$$b_{jk}x_jx_k \le d_{jk}x_j^2 + \frac{b_{jk}v_{jk}}{2}x_k^2. \tag{3.27}$$

Proof. Since b_{jk} and v_{jk} have the same sign, $d_{jk} = \frac{b_{jk}}{2v_{jk}} \ge 0$ holds, and thus $d_{jk}(x_j - v_{jk}x_k)^2 \ge 0$ follows. In addition to this,

$$d_{jk}(x_j - v_{jk}x_k)^2 = d_{jk}x_j^2 + d_{jk}v_{jk}^2x_k^2 - 2d_{jk}v_{jk}x_jx_k = d_{jk}x_j^2 + \frac{b_{jk}v_{jk}}{2}x_k^2 - b_{jk}v_{jk}x_jx_k$$

and the inequality (3.27) follows from

$$0 \le d_{jk}(x_j - v_{jk}x_k)^2 = \frac{b_{jk}}{2v_{jk}}\frac{a_j}{a_j}x_j^2 + \frac{b_{jk}}{2v_{jk}}\frac{a_k}{a_j}x_k^2 - b_{jk}x_jx_k =$$
$$\frac{d_{jk}}{a_j}(a_jx_j^2 + a_kx_k^2) - b_{jk}x_jx_k = d_{jk}x_j^2 + \frac{b_{jk}v_{jk}}{2}x_k^2 - b_{jk}x_jx_k.$$

□

3.5.4 Combining the approximation methods

Since (3.22) tends to give better bounds on x_jx_k than (3.23) or (3.27) if the boxes $\mathbf{x}_j$ and $\mathbf{x}_k$ are not too wide, but infinite ones if the width of one of the boxes is infinite we combine the different methods and proceed as follows for each bilinear term with nonzero coefficients b_{jk}: First, we factor the quadratic, bilinear and linear terms which depend on the variables x_j and x_k from (3.21) and obtain

$$c(x) + a_kx_k^2 + a_jx_j^2 + b_{jk}x_jx_k + b_kx_k + b_jx_j \ge c, \tag{3.28}$$

The entries which do not depend on x_j or x_k are collected in

$$c(x) := \sum_{\substack{i \\ i \ne k, i \ne j}} (a_ix_i^2 + b_ix_i) + \sum_{\substack{m,i \\ m > i \\ (mi) \ne (jk)}} b_{mi}x_mx_i.$$

Then we handle the following cases:

1. If one of the bounds $\underline{x}_j$, $\overline{x}_j$, $\underline{x}_k$, $\overline{x}_k$ is large and both a_k and a_j are negative, we quadratically approximate the bilinear term $b_{jk}x_jx_k$ as described in Subsection 3.5.3. By the inequality (3.27) we obtain the relaxation

$$c(x) + (a_k + d_{jk})x_k^2 + \Big(a_j + \frac{b_{jk}v_{jk}}{2}\Big)x_j^2 + b_kx_k + b_jx_j \ge c, \tag{3.29}$$

showing that the bilinear term $b_{jk}x_jx_k$ has been eliminated from (3.28). We have separated the variables x_j and x_k in (3.28), ending up in

$$c(x) + a'_kx_k^2 + a'_jx_j^2 + b_kx_k + b_jx_j \ge c, \tag{3.30}$$

with the new quadratic coefficients

$$a'_k := a_k + d_{jk},\ a'_j := a_j + \frac{b_{jk}v_{jk}}{2}. \tag{3.31}$$

The linear and constant coefficients remain unchanged. If we have uncertainties in the coefficients; $a_k \in \mathbf{a}_k$ and $b_{jk} \in \mathbf{b}_{jk}$ then (3.31) is changes to

$$a'_k \in \mathbf{a}'_k, \quad a'_j \in \mathbf{a}'_j,$$

with

$$\mathbf{a}'_k = \mathbf{a}_k + \overline{d}_{jk}, \quad \mathbf{a}'_j = \mathbf{a}_j + \sup \frac{\mathbf{b}_{jk}\mathbf{v}_{jk}}{2}, \quad \mathbf{v}_{jk} = \operatorname{sign}(\mathbf{b}_{jk})\sqrt{\frac{\mathbf{a}_k}{\mathbf{a}_j}}, \quad \mathbf{d}_{jk} = \frac{\mathbf{b}_{jk}}{2\mathbf{v}_{jk}}.$$

2. If we have no large bounds on the variables x_j and x_k, the expression

$$p(x) := a_k x_k^2 + a_j x_j^2 + b_{jk} x_j x_k + b_k x_k + b_j x_j$$

can be bounded from above by a convex function only if it has a finite global maximum. This requires that the Hessian

$$H = \begin{pmatrix} 2a_k & b_{jk} \\ b_{jk} & 2a_j \end{pmatrix}$$

is negative semidefinite. If $b_{jk} \neq 0$ this implies that a_k and a_j are negative. Therefore, the constraint propagation on this constraint is useful only in this case.

3. If all bounds $\underline{x}_j$, $\overline{x}_j$, $\underline{x}_k$, $\overline{x}_k$ are not large we approximate the bilinear term $b_{jk}x_jx_k$ by applying the results of Subsection 3.5.1. In addition to this, if we assume that we have uncertainties in the coefficient $b_{jk} \in \mathbf{b}_{jk}$, we can add the supremum of $b_{jk}x_jx_k$ to the right hand side of the inequality (3.28) obtaining

$$c(x) + a_k x_k^2 + a_j x_j^2 + b_k x_k + b_j x_j \geq c', \tag{3.32}$$

where

$$c' := c + \begin{cases} \overline{b}_{jk} \max\{\underline{x}_j\underline{x}_k, \underline{x}_j\overline{x}_k, \overline{x}_j\underline{x}_k, \overline{x}_j\overline{x}_k\} & \text{if } \overline{b}_{jk} \geq -\underline{b}_{jk} \\ -\underline{b}_{jk} \max\{(-\underline{x}_j)\underline{x}_k, (-\underline{x}_j)\overline{x}_k, (-\overline{x}_j)\underline{x}_k, (-\overline{x}_j)\overline{x}_k\} & \text{if } \overline{b}_{jk} < -\underline{b}_{jk}. \end{cases}$$

Thus we have separated the variables x_j and x_k in (3.28). The quadratic and linear coefficients remain unchanged. Note that the signs in the above expression are intended to save rounding mode switches.

4. If the bounds $\underline{x}_j$, $\overline{x}_j$, $\underline{x}_k$, $\overline{x}_k$ are not large, for special applications (e.g. for computing linear relaxations as in Chapter 5), it can be suitable to approximate the bilinear terms by linear expressions. We apply the results of Subsection 3.5.2; we choose a $z \in \mathbf{x}$ (e.g., $z = (\overline{x} + \underline{x})/2$ is a good choice) and approximate each $b_{jk}x_jx_k$

with $x_j \in \mathbf{x}_j$ and $x_k \in \mathbf{x}_k$ by

$$b_{jk}z_j + b_{jk}z_k + d \geq b_{jk}x_jx_k, \quad d := \max_i u_i$$

where

$$\begin{aligned} u_1 &= \Delta b_{jk}((\underline{x}_j - z_j)\underline{x}_k - z_k\underline{x}_j), \quad u_2 = \Delta b_{jk}((\overline{x}_j - z_j)\underline{x}_k - z_k\overline{x}_j), \\ u_3 &= \Delta b_{jk}((\underline{x}_j - z_j)\overline{x}_k - z_k\underline{x}_j), \quad u_4 = \Delta b_{jk}((\overline{x}_j - z_j)\overline{x}_k - z_k\overline{x}_j). \end{aligned} \tag{3.33}$$

Then by (3.28) we have

$$\begin{aligned} &c(x) + a_kx_k^2 + a_jx_j^2 + (b_{jk}z_j + b_k)x_k + (b_{jk}z_k + b_j)x_j + d \\ &\geq c(x) + a_kx_k^2 + a_jx_j^2 + b_{jk}x_jx_k + b_kx_k + b_jx_j \geq c. \end{aligned}$$

Therefore we successfully separated the the variables x_j and x_k in (3.28) and obtain

$$c(x) + a_kx_k^2 + a_jx_j^2 + b'_kx_k + b'_jx_j \geq c' \tag{3.34}$$

with the new linear and constant coefficients

$$b'_k := b_{jk}z_j + b_k, \; b'_j := b_{jk}z_k + b_j, \; c' := c - \max_i u_i. \tag{3.35}$$

The quadratic coefficients remain unchanged. If we have the uncertainties $b_{jk} \in \mathbf{b}_{jk}$ and $b_k \in \mathbf{b}_k$ in the coefficients, (3.35) changes to

$$b'_k \in \mathbf{b}'_k, \quad b'_j \in \mathbf{b}'_j, \quad c' = c - \max_i u_i,$$

with

$$\begin{aligned} \mathbf{b}'_k &= \mathbf{b}_{jk}z_j + \mathbf{b}_k, & \mathbf{b}'_j &= \mathbf{b}_{jk}z_k + \mathbf{b}_j, \\ u_1 &= \sup(\mathbf{b}_{jk}((\underline{x}_j - z_j)\underline{x}_k - z_k\underline{x}_j)), & u_2 &= \sup(\mathbf{b}_{jk}((\overline{x}_j - z_j)\underline{x}_k - z_k\overline{x}_j)), \\ u_3 &= \sup(\mathbf{b}_{jk}((\underline{x}_j - z_j)\overline{x}_k - z_k\underline{x}_j)), & u_4 &= \sup(\mathbf{b}_{jk}((\overline{x}_j - z_j)\overline{x}_k - z_k\overline{x}_j)). \end{aligned}$$

Applying the above on (3.28) for all indexes $j \in \{1, \ldots, n\}$ and $k \in \{1, \ldots, n\}$ with $j < k$, we obtain the new separable system

$$\sum_k a'_kx_k^2 + b'_kx_k \geq c', \quad x \in \mathbf{x}. \tag{3.36}$$

All above bounds should be computed with upward rounding.

3.6 Constraint propagation in GloptLab

This section discusses how the new techniques presented above are implemented in the GloptLab environment to solve algebraic constraint satisfaction problems. The problems treated in GloptLab consist (after preliminary transformations) of simple bounds, linear constraint, and quadratic constraints. We represent simple bounds as box constraint $x \in \mathbf{x}$. The linear and quadratic constraints are represented in a sparse matrix notation. The linear, quadratic, and bilinear monomials occurring in at least one of the constraint (but not the constant term) are collected into an n_q-dimensional column vector $q(x)$. There we choose

$$q(x) = (x_1, \ldots, x_n,\ x_1^2, \ldots, x_1 x_n,\ \ \ldots\ \ x_n x_1, \ldots, x_n^2)^T$$

The coefficients of the ith constraint in the resulting monomial basis are collected in the ith row of a (generally sparse) matrix A, and any constant term (if present) is moved to the right hand side. Thus the linear and quadratic constraints take the form $A_{i:}q(x) \in \mathbf{F}_i$ $(i = 1 \ldots m)$, where $\mathbf{F}_i$ is a closed interval, and $A_{j:}$ denotes the jth row of A.

As in the case of simple bounds, this includes equality constraints and one-sided constraints by choosing for the corresponding $\mathbf{F}_i$ degenerate or unbounded intervals. In compact vector notation, the constraints take the form $Aq(x) \in \mathbf{F}$.

While traditionally the coefficients in a constraint are taken to be exactly known, we allow them to vary in (narrow) intervals, to be able to rigorously account for uncertainties due to measurements of limited accuracy, conversion errors from an original representation to our normal form, and rounding errors when creating new constraints by relaxation techniques. Thus the coefficient matrix A is allowed to vary arbitrarily within some interval matrix $\mathbf{A}$. The $m \times n_q$ interval matrix $\mathbf{A}$ with closed and bounded interval components $\mathbf{A}_{ik} = [\underline{A}_{ik}, \overline{A}_{ik}]$, is interpreted as the set of all $A \in \mathbb{R}^{m \times n}$ such that $\underline{A} \le A \le \overline{A}$, where $\underline{A}$ and $\overline{A}$ are the matrices containing the lower and upper bounds of the components of $\mathbf{A}$.

We therefore pose the general quadratic constraint satisfaction problem in the form

$$Aq(x) \in \mathbf{F}, \quad x \in \mathbf{x}, \quad A \in \mathbf{A}. \tag{3.37}$$

We now summarize the constraint propagation method for the quadratic constraint satisfaction problem (3.37).

Problem simplification. First we simplify the problem; we remove the constraints of the form $bx_j \in \mathbf{F}_i$ and modify the corresponding bounds on the variable x_j. We also

remove the variables which are fixed from the constraints, and the entries corresponding to the removed variables from the vector $q(x)$. The dimensions of the coefficient matrix A and the box $\mathbf{x}$ may change in this step.

Resolving the two-sided constraints. We resolve the two-sided constraints of (3.37) into inequalities. We define

$$A^n := \begin{pmatrix} -\underline{A}_{I,:} \\ \overline{A}_{J,:} \end{pmatrix}, \quad A^p := \begin{pmatrix} \overline{A}_{I,:} \\ -\underline{A}_{J,:} \end{pmatrix}, \quad c := \begin{pmatrix} \underline{F}_I \\ -\overline{F}_J \end{pmatrix},$$

where

$$I := \{i \in 1, \ldots, m \mid \underline{F}_i > -\infty\} \text{ and } J := \{i \in 1, \ldots, m \mid \overline{F}_i < \infty\}.$$

The system of quadratic inequalities

$$Aq(x) \geq c, \quad x \in \mathbf{x}, \quad A \in \mathbf{A} := [-A^n, A^p], \tag{3.38}$$

is another representation of the quadratic constraint satisfaction problem (3.37). The matrix A is $m \times n_q$ dimensional, where $m := n_I + n_J$ depends on the length n_I of the index set I and on the length n_J of the index set J.

Separating the constraints. We transform the quadratic constraint satisfaction problem into a separable one. The ith row of (3.38) matches the form of

$$\sum_k (a_k x_k^2 + b_k x_k) + \sum_{\substack{j,k \\ j>k}} b_{jk} x_j x_k \geq c, \quad x \in \mathbf{x},$$

of (3.21), with

$$a_k := \overline{A}_{i,kn+k}, \quad b_k \in \mathbf{A}_{i,k}, \quad b_{jk} \in \mathbf{A}_{i,jn+k} \text{ and } c := c_i. \tag{3.39}$$

Here we used the upper bounds for the quadratic terms since the sign of x_k^2 is known. Then we use the results of Section 3.5 to remove all bilinear entries from each constraint, obtaining the new coefficients

$$a'_k \in \mathbf{a}'_k, \quad b'_k \in \mathbf{b}_k, \quad b_{jk} = 0 \text{ and } c = c'_i. \tag{3.40}$$

Depending on the removal method we have applied either the quadratic coefficients or the bound c' or both of them have been changed, ending up in a new system

$$A'q(x) \geq c', \quad x \in \mathbf{x}, \quad A' \in \mathbf{A}' := [-A'^n, A'^p] \tag{3.41}$$

In (3.41) all bilinear coefficients are zero, therefore from this point on, the system is separable.

Forward and backward propagation. Since the ith row of (3.41) matches the form

$$\sum_{k=1}^{n}(a_k x_k^2 + b_k x_k) \geq c, \quad x \in \mathbf{x}, \quad a_k \in \mathbf{a}_k, \quad b_k \in \mathbf{b}_k$$

of (3.5), we can apply the forward and the backward propagation steps from Section 3.4.

We compute the enclosure $\mathbf{p}_k$ of each univariate quadratic term $p_k(x_k) := a_k x_k^2 + b_k x_k$ by using the theory developed in Section 3.2, where the uncertainties $\mathbf{a}_k$ and $\mathbf{b}_k$ of the constraint coefficients are also taken into account.

Then we use the $\mathbf{p}_k$ to verify that the constraint is feasible, to get a new bound on each $p_k(x_k)$ and to find a new lower bound for the constraint.

If the constraint has not yet been detected as infeasible, we can apply the backward propagation step (by using the theory from Section 3.3), which may yield tighter bounds on the variables.

3.7 Tests and Comparison

In this section we compare our quadratic constraint propagation method (QCP) with elementary constraint propagation (ECP).

The forward propagation step of the elementary constraint propagation finds the range of each expression in the constraints individually, then for all expressions in a constraint uses the ranges of all other expressions to get new bounds on them. The backward propagation step uses the inverse of the expressions to get new bounds on the variables.

Example 3.7. *To demonstrate the elementary constraint propagation and simultaneously compare it to the method presented in this chapter we again solve the problem*

$$-x_1^2 + 2x_1 - x_2 \geq -8 \quad \textit{with } x_1 \in [-7, 5],\ x_2 \in [0, \infty].$$

from Example 3.4 and give the step by step comparison of the two different approaches:

ECP:

- *For* $x_1 \in [-7,5]$, $x_2 \in [0,\infty]$, *we have* $-x_1^2 \in [-49,0]$, $2x_1 \in [-14,10]$ *and* $-x_2 \in [-\infty,0]$.
- *Therefore,* $-x_1^2 + 2x_1 - x_2 \leq 10$.
- *From* $0 \geq -x_2$, $0 \geq -x_1^2$ *and* $10 \geq 2x_1$, *the inequalities* $-x_1^2 \geq -8-0-10 = -18$ *and* $2x_1 \geq -8-0-0 = -8$ *follows.*
- *Since* $10 \geq -x_1^2 + 2x_1$, *the inequality* $-x_2 \geq -8-10 = -18$ *holds.*
- *From* $-x_1^2 \geq -18$, $x_1 \in [-\sqrt{18}, \sqrt{18}]$, *from* $2x_1 \geq -8$, $x_1 \in [-4,\infty]$, *and from* $-x_2 \geq -18$, $x_2 \in [-\infty, 18]$ *follows.*
- *Intersecting the bounds on the variables with the original bounds, we obtain* $-x_1^2 + 2x_1 - x_2 \in [-8,10]$ *with* $x_1 \in [-4, \sqrt{18}]$, $x_2 \in [0,18]$.

QCP:

- *For* $x_1 \in [-7,5]$, $x_2 \in [0,\infty]$, *we have* $-x_1^2 + 2x_1 \in [-63,1]$ *and* $-x_2 \in [-\infty,0]$.
- *Therefore,* $-x_1^2 + 2x_1 - x_2 \leq 1$.
- *From* $0 \geq -x_2$, *the inequality* $-x_1^2 + 2x_1 \geq -8-0 = -8$ *follows.*
- *Since* $1 \geq -x_1^2 + 2x_1$, *the inequality* $-x_2 \geq -8-1 = -9$ *holds.*
- *From* $-x_1^2 + 2x_1 \geq -8$, $x_1 \in [-2,4]$, *and from* $-x_2 \geq -9$, $x_2 \in [-\infty, 9]$ *follows.*
- *Intersecting the bounds on the variables with the original bounds, we obtain* $-x_1^2 + 2x_1 - x_2 \in [-8,1]$ *with* $x_1 \in [-2,4]$, $x_2 \in [0,9]$.

As the example shows for this problem the quadratic constraint propagation presented in this chapter gives significantly tighter bounds than the elementary constraint propagation used as the pruning step of several state-of-the-art constraint propagation methods (e.g. [19, 38, 102]) with hardly any extra work. Probing ([99], also called slicing or shaving) would yield the same results as QCP, but at a much higher cost.

Strategy 3.1 (Test strategies)**.**
In order to compare our method with the traditional approaches we reimplemented the elementary constraint propagation in MATLAB*, integrated in* GLOPTLAB *Chapter 2 and used branch and prune to solve random problems.* GLOPTLAB *executes configurable strategies, for the comparison we used the following one, with default tuning parameter settings (which are configurable in* GLOPTLAB*).*

```
01: Read Problem
02: Begin While
03:    Propagate*
04: End While
05: Begin Split
06:    Begin While
07:       Propagate*
08:    End While
09: End Split
11: Finish
```

Test strategies	
name	*Propagate* method selection*
ELEM	*elementary constraint propagation*
SCON	*quadratic CP with constant bilinear approximation*
SLIN	*quadratic CP with linear bilinear approximation*
SQUA	*quadratic CP with quadratic bilinear approximation*
SAUT	*quadratic CP with automatic bilinear approximation*

Each strategy from 3.1 first reads the problem then accomplishes a single propagation step (each of them using different method) until the gain is less than 20% *of the original box or until the number of iterations exceeds 20. Then the branching process follows; the box is split at the midpoint of a selected component then the same sequence of constraint propagation is applied to subboxes as before. Infeasible boxes are discarded, feasible boxes are split again if their maximum width is more than 0.001. Boxes of maximum width smaller than 0.001 are not split but saved for the final output.*

In the first test we use the strategies from 3.1 to test three test sets of 50 random, infeasible problems. The problems are 2 dimensional in the first, 5 dimensional in the second, and 10 dimensional in the third test set. Each problem in the test consists of a single conic inequality constraint with random coefficients and random bound constraints $x \in \mathbf{x}$, choosen such that $\mathbf{x}_i \subseteq [-1, 1]$ and the problem is infeasible (infeasibility was verified by using a more complicated strategy; Strategy 5.2 from Chapter 2, Section 5.2). The table below shows the median of the solution times (in seconds) and the median of the splits required to solve the problems contained in each test set.

Branch and prune test results.						
dimension	n = 2		n = 5		n = 10	
method	time	splits	time	splits	time	splits
ELEM	0.003	0	0.255	3.00	22.58	47.5
SCON	0.001	0	0.033	1.75	0.722	47.3
SLIN	0.003	0	0.039	1.50	0.984	55.0
SQUA	0.001	0	0.045	2.00	1.500	95.5
SAUT	0.002	0	0.035	1.75	0.813	47.3

As the results show, verifying in higher dimensions that the search space does not contain points of single conic inequality constraint consisting of bilinear terms using constraint propagation is a non-trivial task. The reason is that the approximation error of the bilinear terms (and in case of the elementary constraint propagation also the approximation of the separable quadratic expressions) makes the CP incapable to discard the regions of the search space which are close to region defined by the constraint. Only a division of the search space into several subboxes leads to a solution.

The elementary constraint propagation is slower than the quadratic constraint propagation, due to the need of more rounding mode switches in the interval arithmetic and the greater approximation error (see Example 3.7). Since the width of the bound constraint box is small, the constant approximation performs better than the linear or the quadratic ones. The automatic method is only slightly slower than the constant approximation, but has the advantage that it is also performs good when the bound constraints are large.

The following tests show how the constraint propagation method presented in this chapter scales favorably with the complexity of the constraints. The test problems are 9 dimensional, having linear equality constraints (depending on the variables x_i, x_{i+1} and x_{i+2}, $i = 1, \ldots, 7$) but the fifth constraint also has some quadratic and bilinear terms in 3 (`Test 1`), in 4 (`Test 2`), in 5 (`Test 3`), or in 6 variables (`Test 4`). In `Test 1-4` the fifth constraint is convex, while in `Test 1'-4'` non-convex. We have chosen 20 random bound constraints $x \in \mathbf{x}$ such that $\underline{x}_i \in [-b, 0]$ and $\overline{x}_i = \underline{x}_i + 2b$ for $i = 1, \ldots n$, and listed the different bs in the `bound` column of the tables. We added the random bounds to the test problems, and solved them using the `ELEM` and the `SAUT` strategy. The median of the solution times (in seconds) are shown in the following tables:

Median of the solution times for convex (`Test 1-4`) and non-convex (`Test 1'−4'`) problems. Problems in each test run: 20, strategy: `ELEM`.								
`bound`	`Test 1`	`Test 1'`	`Test 2`	`Test 2'`	`Test 3`	`Test 3'`	`Test 4`	`Test 4'`
1	0.082	0.055	0.094	0.081	0.257	0.171	0.170	0.348
10	1.391	1.031	1.823	1.605	2.772	2.589	4.229	4.597
100	1.416	1.008	1.969	1.894	2.624	2.862	4.688	4.414
1000	1.883	1.481	2.098	2.085	3.217	3.245	5.496	4.749

Median of the solution times for convex (`Test 1-4`) and non-convex (`Test 1'−4'`) problems. Problems in each test run: 20, strategy: `SAUT`.								
`bound`	`Test 1`	`Test 1'`	`Test 2`	`Test 2'`	`Test 3`	`Test 3'`	`Test 4`	`Test 4'`
1	0.017	0.028	0.027	0.021	0.025	0.021	0.023	0.040
10	0.244	0.282	0.293	0.256	0.325	0.319	0.339	0.404
100	0.256	0.289	0.286	0.298	0.294	0.335	0.338	0.360
1000	0.406	0.348	0.696	0.390	0.694	0.339	0.919	0.677

Note that the convex problems without additional bound constraints can be solved in less then 0.12 seconds by adding the ellipsoid hull enclosure method presented in Chapter 4 to the above strategy.

Remark 3.4. Testing and comparing the above methods with other constraint propagation methods on standard benchmarks only makes sense when the method is integrated in the same strategy (combined with branch and bound, shaving, relaxations etc.). Constraint propagation alone is not powerful enough to solve the most real life problems and the type and quality of the auxiliary methods does count a lot if we would like to compare our method with the constraint propagation methods implemented in other solvers like ICOS or Realpaver.
An implementation of the above methods in other programming languages (e.g., in C++, which is significantly faster than Matlab) or using other representations of the problem (e.g. using DAGs) should yield a significant reduction of the solution times. However, we expect that the relative quality of the different methods will approximately remain the same. A C++ implementation for the COCONUT Environment is in preparation.

Remark: The Sections 3.2 and 3.3 were originally written by Arnold Neumaier, the remaining Sections are my work.

Chapter 4

Rigorous enclosures of ellipsoids and directed Cholesky factorizations

Abstract.

This chapter discusses the rigorous enclosure of an ellipsoid by a rectangular box, its interval hull, providing a convenient preprocessing step for constrained optimization problems.

A quadratic inequality constraint with a strictly convex Hessian matrix defines an ellipsoid. The Cholesky factorization can be used to transform a strictly convex quadratic constraint into a norm inequality, for which the interval hull is easy to compute analytically. In exact arithmetic, the Cholesky factorization of a nonsingular symmetric matrix exists iff the matrix is positive definite. However, to cope efficiently with rounding errors in inexact arithmetic is nontrivial. Numerical tests show that even nearly singular problems can be handled successfully by our techniques.

To rigorously account for the rounding errors involved in the computation of the interval hull and to handle quadratic inequality constraints having uncertain coefficients, we define the concept of a directed Cholesky factorization, and give two algorithms for computing one. We also discuss how a directed Cholesky factorization can be used for testing positive definiteness. Some numerical test are given in order to exploit the features and boundaries of the directed Cholesky factorization methods.

Keywords. quadratic constraints, interval analysis, constraint satisfaction problems, bounding ellipsoids, interval hull, directed Cholesky factorization, verification of positive definiteness, rounding error control, preprocessing, verified computing

Several state of the art global optimization solvers, such as BARON (by SAHINIDIS & TAWARMALANI [86]) or COCOS (by SCHICHL et al. [88]), combine a number of methods and strategies to find one or more global solutions of a constrained optimization problem. Most of the techniques (e.g branch and bound, heuristics) require explicit bounds for each variable from below and from above. If a problem lacks these explicit prior bounds, the usual remedy is to set default upper and lower bounds on the variables, thereby changing the problem. If the global minimum lies outside the default bounds, the solver cannot find the solution.

A rigorous enclosure technique for strictly convex quadratic constraints presented in this paper give the possibility to obtain rigorous bounds on variables that are consequences of the constraints, without the need of giving explicit bounds on them. This makes the method a convenient preprocessing step for constrained optimization problems. On the other hand since the enclosures obtained by the method are rigorous, the method is also applicable in verified global optimization (e.g., KEARFOTT [54], LEBBAH [59], Chapter 2) and in computer assisted proofs (see, e.g., NEUMAIER [74]). Since it reduces the search space, it may also be important for stochastic, sampling-based optimization methods.

The chapter is logically divided into two parts. The first part (Sections 4.2 - 4.4) is about computing rigorous enclosures for strictly convex quadratic constraints. In the second part (Sections 4.5 - 4.7) the theory of the directed Cholesky factorization is developed as an essential tool for making the results of the first part rigorous.

In the first section we find an optimal box enclosure of an ellipsoid defined by a simple Euclidean norm inequality constraint. In Section 4.2 we extend these results and generate optimal enclosures for strictly convex quadratic constraints. We also consider the case of inexact arithmetic, where the error of the factorization of the coefficient matrix has to be controlled. The need of scaling when confronted with ill-conditioned coefficient matrices is discussed in Section 4.3. In Section 4.4 we develop the method into a useful tool for preprocessing constrained optimization problems to get finite bounds on the variables or to improve the existing ones. Since the method is rigorous, our preprocessing step finds finite bounds for all variables if each unbounded variable occurs in some strictly convex quadratic constraint, without losing any feasible point. These bounds on all n variables are obtainable with $O(n^3)$ operations. The method is implemented in the GLOPTLAB optimization environment (see Chapter 2).

To rigorously account for the rounding errors involved in the computation of the interval hull and to handle quadratic inequality constraint having uncertain coefficients, we define the concept of a *directed* Cholesky factorization.

In the second part of the chapter, we give algorithms which compute, if possible, for a real, symmetric matrix A a nonsingular triangular matrix R (a *directed* Cholesky factor) such that the error matrix $A - R^T R$ of the factorization is *small* compared to the entries of $|A|$, and guaranteed to be positive semidefinite. Clearly, this implies that A is positive definite; conversely (in the absence of overflow), any 'sufficiently' positive definite symmetric matrix has such a factorization with R representable in floating point arithmetic. The challenge is to find such a representation which makes the error as small as possible and works even for nearly singular matrices.

Two different versions of the directed Cholesky factorization for real symmetric matrices are discussed in Section 4.5. Both of them check positive definiteness and, when successful, compute a directed Cholesky factor with positive semidefinite error matrix containing small entries. The first approach uses an a priori error estimate, an approximate Cholesky factorization, and the so-called *Gerschgorin test* (explained later). The second one uses directed rounding and diagonal pivoting to obtain a directed Cholesky factor. Section 4.6 contains some tests and comparison of the two directed Cholesky factorization methods.

In some applications, it is necessary to safeguard the computations in order to ensure the mathematical correctness of the assertions in spite of rounding errors. This applies to computer-assisted proofs in which positive definiteness must be verified rigorously (a potential application to Lie group representations is described in Adams [1]). This also applies to box reduction methods for global optimization (see, e.g., [73, 85]) to guarantee that no feasible point is lost.

The last section is concerned with applications of the directed Cholesky factorization for verifying positive definiteness rigorously. Previous work includes Adjiman et al. [2, 4, 5], Neumaier [73], Rump [79–81, 83]. We show that a directed Cholesky factorization can be employed for the same task, and that the positive definiteness of a complex Hermitian matrix can be checked in real arithmetic by factorizing a related real matrix of twice the size.

The well-known theorem of Gerschgorin, (see, e.g. Stoer & Bulirsch [94]) implies that every symmetric H-matrix with non-negative diagonal entries is positive definite; we call this the *Gerschgorin test for positive definiteness*. Other sufficient conditions for positive definiteness based on scaled Gerschgorin theorems and semidefinite programming, form the basis of the αBB method Adjiman et al. [3] and Androulakis [8] and are given in Adjiman et al. [2, 4]. For further tests see the discussion in Section 4.7.

4.1 Bounding strictly convex norm constraints

In this section we construct an optimal box enclosure of an ellipsoid defined by the Euclidean norm constraint

$$\|Rx\|_2^2 + 2a^T x \le \alpha, \tag{4.1}$$

for a given $R \in \mathbb{R}^{n\times n}$, $a \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$. The following result — for ellipsoids centered on the origin ($a = 0$ in (4.1)) — is the basic tool for finding an optimal enclosure of (4.1).

Proposition 4.1. *Suppose that $R, C \in \mathbb{R}^{n\times n}$, $0 < \beta \in \mathbb{R}$ and $0 < d \in \mathbb{R}^n$ satisfy*

$$d_i \ge \sqrt{(CC^T)_{ii}} \text{ for all } i = 1, \dots, n, \tag{4.2}$$

as well as

$$\beta d \le \langle CR\rangle d. \tag{4.3}$$

Then R is invertible, and for $x \in \mathbb{R}^n$

$$\|Rx\|_2^2 \le \delta^2 \quad \Rightarrow \quad |x| \le \frac{\delta}{\beta} d. \tag{4.4}$$

Proof. We first note that (4.2) implies $(CC^T)_{ii} \le d_i^2$. Since CC^T is positive semidefinite, $|(CC^T)_{ik}|^2 \le (CC^T)_{ii}(CC^T)_{kk} \le d_i^2 d_k^2$, so that

$$|CC^T| \le dd^T.$$

Since (4.3) implies that CR is an H-matrix (Neumaier [70, Section 3.7]), the matrix CR is invertible (hence also R), and $|(CR)^{-1}| \le \langle CR\rangle^{-1}$. Moreover, multiplying (4.3) by $\langle CR\rangle^{-1}\beta^{-1} \ge 0$, we find $\langle CR\rangle^{-1} d \le \beta^{-1} d$. Now let $z := R^{-T} e^i$ with the ith unit vector $e^i = I_{:i}$. Then $e^i = R^T z$, hence

$$\begin{aligned} x_i^2 &= (e^{iT}x)^2 = (z^T Rx)^2 \le \|z\|_2^2 \|Rx\|_2^2 \le \delta^2 \|z\|_2^2 = \delta^2 z^T z \\ &= \delta^2 e^{iT} R^{-1} R^{-T} e^i = \delta^2 e^{iT} (CR)^{-1} CC^T (CR)^{-T} e^i \\ &\le \delta^2 e^{iT} \langle CR\rangle^{-1} dd^T \langle CR\rangle^{-T} e^i = (\delta e^{iT} \langle CR\rangle^{-1} d)^2 \\ &\le (\delta\beta^{-1} e^{iT} d)^2 = (\delta\beta^{-1} d_i)^2, \end{aligned} \tag{4.5}$$

proving (4.4). □

Example 4.1. *For the ellipsoid*

$$\|Rx\|_2^2 \le \delta^2 \text{ with } R = \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix} \text{ and } \delta^2 = 10,$$

choosing $C = R^{-1}$, $d_i = \sqrt{(CC^T)_{ii}}$, $\beta = 1$ *and applying the results of Proposition 4.1 we find the enclosure*

$$|x| \leq \frac{\delta}{\beta} d = \begin{pmatrix} \sqrt{5} \\ \sqrt{10} \end{pmatrix} \approx \begin{pmatrix} 2.24 \\ 3.17 \end{pmatrix} \quad \textit{(see Figure 4.2).}$$

MATLAB CODE FOR TESTING EXAMPLE 4.1.

```
R=[2 -1;0 1], delta=sqrt(10), C=inv(R),
d=sqrt(diag(C*C')), beta=1, bound=delta/beta*d
```

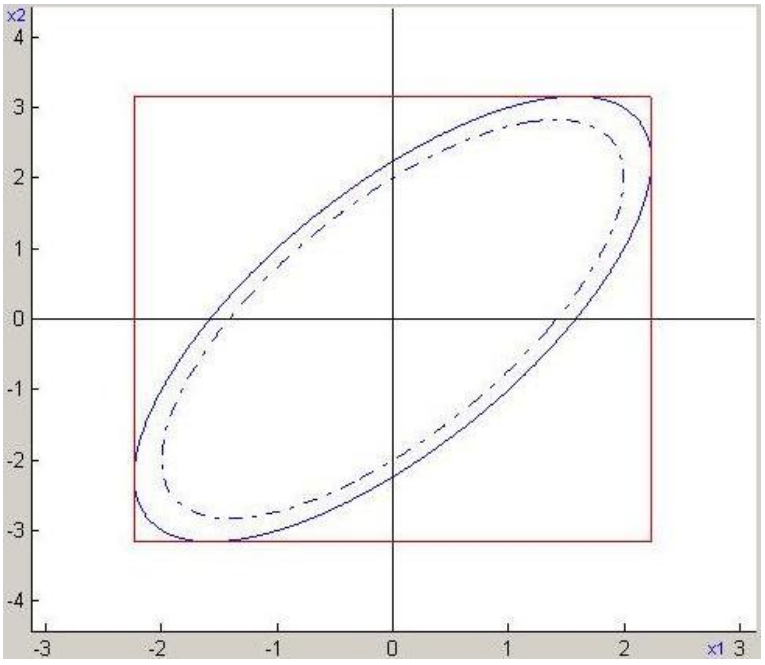

FIGURE 4.1: Box enclosure found for the ellipsoid from Example 4.1

We now show that the choice for the parameters C, d and β we made in the above example was the optimal one.

Proposition 4.2. *Under the assumptions of Proposition 4.1, the bound on x is optimal if $C = R^{-1}$, $d_i = \sqrt{(CC^T)_{ii}}$ and $\beta = 1$.*

Proof. From $\beta d \leq \langle CR \rangle d$ with $C = R^{-1}$ follows that $\beta \leq 1$. Therefore β is maximal if $\beta = 1$ and d_i is minimal if $d_i = \sqrt{(CC^T)_{ii}}$. The assertion that the bound is optimal follows if we show that for all $i = 1, \ldots, n$ the points

$$\hat{x}^i := \pm \frac{\delta}{d_i} R^{-1} R^{-T} e^i$$

satisfy $\|R\hat{x}^i\|_2 = \delta$ and the ith component of $\hat{x}^i$ matches the boundary of the box $[-\delta d_i, \delta d_i]$ enclosing the ellipsoid. Since

$$d_i = \sqrt{(CC^T)_{ii}} = \sqrt{e^{iT}R^{-1}R^{-T}e^i} = \|R^{-T}e^i\|_2 > 0$$

holds, the first claim follows from

$$\|R\hat{x}^i\|_2 = \| \pm \frac{\delta}{d_i} RR^{-1}R^{-T}e^i\|_2 = \frac{\delta}{d_i}\| \pm R^{-T}e^i\|_2 = \delta,$$

and the second claim follows from

$$d_i\hat{x}^i_i = d_i(e^{iT}\hat{x}^i) = \pm\delta e^{iT}R^{-1}R^{-T}e^i = \pm\delta\|R^{-T}e^i\|_2^2 = \pm\delta d_i^2, \tag{4.6}$$

after division by d_i. □

If we shift the center of the ellipsoid by replacing x in Propositions 4.1 and 4.2 by $x - \tilde{x}$, we find:

Corollary 4.3. *Suppose that $R \in \mathbb{R}^{n\times n}$ is invertible, $\tilde{x} \in \mathbb{R}^n$, $d_i \geq \sqrt{(R^{-1}R^{-T})_{ii}}$, and $\beta d \leq \langle R^{-1}R\rangle d$ then*

$$x \in \mathbb{R}^n, \quad \|R(x-\tilde{x})\|_2 \leq \delta \quad \Rightarrow \quad |x-\tilde{x}| \leq \frac{\delta}{\beta}d. \tag{4.7}$$

The bound on $x - \tilde{x}$ is optimal if $d_i = \sqrt{(R^{-1}R^{-T})_{ii}}$ and $\beta = 1$. □

We use Propositions 4.1 and 4.2 to achieve the main result of this section given by the following theorem; we derive cheap and in inexact arithmetic only slightly non optimal bounds on x for the general norm inequality (4.1). The theorem, which is valid for arbitrary $\tilde{z}$, $\tilde{x} \in \mathbb{R}^n$, will be used with

$$\tilde{z} = R^{-T}a, \quad \tilde{x} = -R^{-1}\tilde{z} = -R^{-1}R^{-T}a, \tag{4.8}$$

to make γ small. We know that if the choice is exact then $\gamma = 0$ and the bounds would be optimal.

Theorem 4.4 (Ellipsoid Hull). *For given $R \in \mathbb{R}^{n\times n}$, $a \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$, under the assumptions of Proposition 4.1, for arbitrary $\tilde{z}, \tilde{x} \in \mathbb{R}^n$ and $\gamma, \Delta \in \mathbb{R}$ satisfying*

$$\gamma \geq \|\tilde{z} + R\tilde{x}\|_2 + \beta^{-1}d^T|a - R^T\tilde{z}|, \tag{4.9}$$

and

$$\Delta \geq \gamma^2 + \alpha - 2a^T\tilde{x} - \|R\tilde{x}\|_2^2, \tag{4.10}$$

the following statements hold:

(i) If $\Delta < 0$ *then*

$$\|Rx\|_2^2 + 2a^T x \le \alpha \tag{4.11}$$

has no solution $x \in \mathbb{R}$.

(ii) If $\Delta \ge 0$ *then* (4.11) *implies that*

$$\|R(x-\tilde{x})\|_2 \le \delta := \gamma + \sqrt{\Delta}, \quad |x-\tilde{x}| \le \frac{\delta}{\beta} d. \tag{4.12}$$

Proof. For any $x \in \mathbb{R}^n$, Proposition 4.1 implies

$$|x-\tilde{x}| \le \frac{\varepsilon}{\beta} d, \quad \text{where } \varepsilon = \|R(x-\tilde{x})\|_2. \tag{4.13}$$

If (4.11) holds then

$$\begin{aligned} \|Rx\|_2^2 &\le \alpha - 2a^T x \le \Delta - \gamma^2 + 2a^T\tilde{x} + \|R\tilde{x}\|_2^2 - 2a^T x \\ &= \Delta - \gamma^2 + \|R\tilde{x}\|_2^2 - 2a^T(x-\tilde{x}). \end{aligned}$$

Therefore

$$\begin{aligned} \varepsilon^2 &= \|R(x-\tilde{x})\|_2^2 = (x-\tilde{x})^T R^T R(x-\tilde{x}) \\ &= x^T R^T Rx - 2\tilde{x}^T R^T Rx + \tilde{x}^T R^T R\tilde{x} \\ &= \|Rx\|_2^2 - 2\tilde{x}^T R^T Rx + \|R\tilde{x}\|_2^2 \\ &\le \Delta - \gamma^2 - 2a^T(x-\tilde{x}) - 2\tilde{x}^T R^T Rx + 2\|R\tilde{x}\|_2^2 \\ &= \Delta - \gamma^2 - 2(a + R^T R\tilde{x})^T(x-\tilde{x}). \end{aligned}$$

By (4.9), the inequality

$$\begin{aligned} \left|(a + R^T R\tilde{x})^T(x-\tilde{x})\right| &= \left|(\tilde{z} + R\tilde{x})^T R(x-\tilde{x}) + (a - R^T\tilde{z})^T(x-\tilde{x})\right| \\ &\le \|\tilde{z} + R\tilde{x}\|_2 \, \|R(x-\tilde{x})\|_2 + |a - R^T\tilde{z}|^T|x-\tilde{x}| \\ &\le \|\tilde{z} + R\tilde{x}\|_2\varepsilon + |a - R^T\tilde{z}|^T\tfrac{\varepsilon}{\beta} d \le \varepsilon\gamma, \end{aligned}$$

holds. We therefore conclude

$$(\varepsilon - \gamma)^2 \le \Delta - 2\varepsilon\gamma + 2\left|(a + R^T R\tilde{x})^T(x-\tilde{x})\right| \le \Delta.$$

If $\Delta < 0$, we get a contradiction, proving (i). And if $\Delta \ge 0$, we find $\varepsilon \le \gamma + \sqrt{\Delta} = \delta$, and (ii) follows from (4.13). □

For the choice (4.8) and assuming that the computations are exact, we get the optimal bounds

$$\tilde{x} - \frac{\delta}{\beta} d \leq x \leq \tilde{x} + \frac{\delta}{\beta} d,$$

by using Theorem 4.4. This is shown by the following corollary.

Corollary 4.5. *If we chose*

$$\begin{aligned} C &= R^{-1}, \\ \beta &= 1, \\ d_i &= \sqrt{(R^T R^{-T})_{ii}}, \\ \gamma &= \|\tilde{z} + R\tilde{x}\|_2 + \beta^{-1} d^T |a - R^T \tilde{z}|, \\ \Delta &= \gamma^2 + \alpha - 2a^T \tilde{x} - \|R\tilde{x}\|_2^2, \\ \tilde{z} &= R^{-T} a, \\ \tilde{x} &= -R^{-1} R^{-T} a, \end{aligned}$$

then the bound on $x - \tilde{x}$ in Theorem 4.4 are optimal.

Proof. By the choice of C, β, d, γ and Δ we have

$$\gamma = \|\tilde{z} + R\tilde{x}\|_2 + \beta^{-1} d^T |a - R^T \tilde{z}| = 0, \tag{4.14}$$

and

$$\begin{aligned} \Delta &= \alpha - 2a^T \tilde{x} - \|R\tilde{x}\|_2^2 \\ &= \alpha + 2a^T R^{-1} R^{-T} a - \| - RR^{-1} R^{-T} a\|_2^2 = \alpha + \|R^{-T} a\|_2^2. \end{aligned} \tag{4.15}$$

By the choice of $\tilde{z}$ and $\tilde{x}$ we have

$$\begin{aligned} \|R(x - \tilde{x})\|_2^2 &= \|Rx\|_2^2 + \|R\tilde{x}\|_2^2 + x^T R^T R\tilde{x} + \tilde{x}^T R^T Rx \\ &= \|Rx\|_2^2 + \|R^{-T} a\|_2^2 + 2a^T x, \end{aligned}$$

Therefore, $\|R(x - \tilde{x})\|_2^2 \leq \delta^2 = \Delta$ implies $\|Rx\|_2^2 + \|R^{-T} a\|_2^2 + 2a^T x \leq \alpha + \|R^{-T} a\|_2^2$ by (4.15), hence $\|Rx\|_2^2 + 2a^T x \leq \alpha$. This gives the forward direction of

$$\|R(x - \tilde{x})\|_2 \leq \delta \quad \Leftrightarrow \quad \|Rx\|_2^2 + 2a^T x \leq \alpha$$

and the reverse direction follows from (4.12). By the choice of d_i and β, we can apply the second part of Corollary 4.3, proving that the bound on $x - \tilde{x}$ is optimal. □

The reason for the strange formulation of Proposition 4.14 (using minimal and maximal instead of giving the best choices) is that in practice, one cannot make the required

choices exact, since rounding errors affect the results of the defining formulas. However, using approximations for $\tilde{x}$ and $\tilde{z}$ computed by ordinary floating point arithmetic, tight bounds which take account of the rounding errors are easy to get with directed, upward rounding. In this way we get nearly optimal enclosure. In 2 dimensions, the results are visually indistinguishable from the optimal enclosures. In exact arithmetics however, by Theorem 4.4 and Corollary 4.5 we can summarize:

Theorem 4.6. *Let* $\|Rx\|_2^2 + 2a^T x \leq \alpha$ *be an ellipsoid. Suppose that* R *is invertible, then*

$$\mathbf{x} = [\tilde{x} - \delta d, \tilde{x} + \delta d]$$

with $d_i = \sqrt{(R^{-1}R^{-T})_{ii}}$, $\tilde{x} = -R^{-1}R^{-T}a$ *and* $\delta = \sqrt{\alpha + \|R^{-T}a\|_2^2}$ *defines an interval hull for the given ellipsoid.* □

Example 4.2. *For the ellipsoid*

$$\|Rx\|_2^2 + 2a^T x \leq \alpha \text{ with } R = \begin{pmatrix} 2 & -1 \\ 0 & 1 \end{pmatrix},\ 2a^T = (2 \quad 3),\ \alpha = 10,$$

which is shown Figure 4.2, we apply Theorem 4.6 and obtain the outward rounded (to three significant digits) interval hull

$$\mathbf{x} = \begin{pmatrix} [-3.92, 1.42] \\ [-5.78, 1.78] \end{pmatrix}.$$

MATLAB CODE FOR TESTING EXAMPLE 4.2.

```
R=[2 -1;0 1], a=[2;3]./2, alpha=10, C=inv(R),
d=sqrt(diag(C*C')), delta=sqrt(alpha+norm(C'*a)^2,
xhat=-C*C'*a, xl=xhat-delta*d, xu=xhat+delta*d
```

4.2 Enclosing strictly convex quadratic constraints

In this section we apply results of the previous section to enclose strictly convex quadratic constraints in inexact arithmetic. To efficiently cope with the rounding errors we use a method called the directed Cholesky factorization to transform a strictly convex quadratic constraint into an Euclidean norm constraint (4.1). The directed Cholesky factorization takes the rounding errors involved in the transformation into account and is discussed in detail in Sections 4.5.

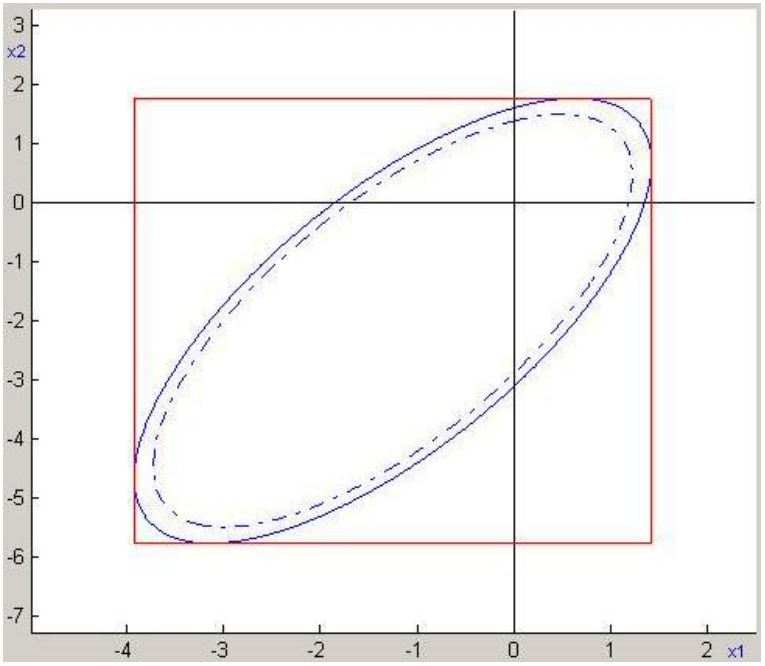

FIGURE 4.2: Optimal box enclosure of the ellipsoid defined in Example 4.2

Let A be a symmetric, positive definite matrix. The strictly convex quadratic constraint

$$x^T Ax + 2a^T x \leq \alpha \tag{4.16}$$

describes an ellipsoid. We derive a nearly optimal enclosure $\mathbf{x}$ for this ellipsoid such that each x satisfying (4.16) is contained in the box $\mathbf{x}$ (hence the method is rigorous).

We compute a permutation matrix P and an upper triangular matrix R such that the residual matrix $\hat{E} := PAP^T - \hat{R}^T\hat{R}$ is positive semidefinite and is very small with respect to PAP^T (details in Section 4.5). If the factorization fails, the positive definiteness of A cannot be verified and the enclosure cannot be computed. (This case only happens when A is an indefinite or a nearly indefinite matrix.) If the factorization is successful the constraint is strictly convex and we have

$$A = P^T\hat{R}^T\hat{R}P + P^T\hat{E}P, \tag{4.17}$$

where the residual matrix $\hat{E}$ (and also $P^T\hat{E}P$) is positive semidefinite and very small with respect to A. Substituting in (4.16) we have

$$x^T(P^T\hat{R}^T\hat{R}P + P^T\hat{E}P)x + 2a^T x = \|\hat{R}Px\|_2^2 + x^T P^T\hat{E}Px + 2a^T x \leq \alpha,$$

and if we define $R := \hat{R}P$, we end up in

$$\|Rx\|_2^2 + 2a^T x \leq \alpha - x^T P^T \hat{E} P x \leq \alpha. \tag{4.18}$$

This proves that the ellipsoid defined by (4.16) is fully contained in the ellipsoid given by the norm constraint

$$\|Rx\|_2^2 + 2a^T x \leq \alpha. \tag{4.19}$$

Note that (4.18) is only then a valid inequality if the residual matrix E is positive semidefinite. Since $\hat{E}$ is very small with respect to PAP^T, the relative approximation error

$$\delta(x) := \frac{x^T \hat{E} x}{\|Rx\|_2^2},$$

is also small, for all $x \in \mathbf{x}$.

We apply the main result of Section 4.1, Theorem 4.4 and Corollary 4.5, to (4.19), choosing $\tilde{z} \approx R^{-T}a$ and $\tilde{x} \approx -R^{-1}\tilde{z}$ by ordinary floating point calculations, and the remaining variables optimally, by computing the corresponding expressions with directed rounding or interval arithmetic. The details are given in the following algorithm.

Algorithm 4.1 (Ellipsoid Hull)**.**
Compute a box enclosure of strictly convex quadratic constraint $x^T Ax + 2a^T x \leq \alpha$:

1. *Find a directed Cholesky factorization of the matrix A:*
   (a) *if the factorization fails, the positive definiteness of A cannot be verified and the enclosure cannot be computed,*
   (b) *otherwise a directed Cholesky factor R is obtained.*
2. *Compute the approximative inverse C of the matrix R.*
3. *Compute d with $d_i = \inf(\sqrt{(CC^T)_{ii}})$ by using directed rounding.*
4. *Use upward rounding to compute $h = \langle CR \rangle d$ and obtain $\beta = \max\{h_i/d_i | i = 1 \ldots n\}$ which must be approximately one.*
5. *Set $\tilde{z} = C^T a$ and $\tilde{x} = -C\tilde{z}$ and compute an enclosure $[\underline{\gamma}, \overline{\gamma}]$ of the expression*

$$\|\tilde{z} + R\tilde{x}\|_2 + \beta^{-1} d^T |a - R^T \tilde{z}|$$

and an enclosure $[\underline{\Delta}, \overline{\Delta}]$ of the expression

$$\gamma^2 + \alpha - 2a^T \tilde{x} - \|R\tilde{x}\|_2^2,$$

by using interval arithmetic.

6. *Finally, use outward rounding to compute the interval*

$$[\underline{\delta}, \overline{\delta}] := [\underline{\gamma} + \sqrt{\underline{\Delta}}, \overline{\gamma} + \sqrt{\overline{\Delta}}].$$

7. *The result is an approximate but rigorous enclosing ellipsoid for* (4.16)*, given by the norm constraint* $\|R(x - \tilde{x})\|_2 \leq \overline{\delta}$*, as well as the rigorous box enclosure*

$$x \in \left[(\delta/\overline{\beta})d - \tilde{x}, (\delta/\underline{\beta})d + \tilde{x}\right].$$

The algorithm applies with trivial modifications for the case if A and a is uncertain (their components vary in intervals). This form is implemented in GloptLab (see Chapter 2).

4.3 Scaling

The ellipsoid hull approximation which was presented in the previous section may have difficulties when used on badly scaled systems. Scaling the constraints before applying the Cholesky factorization increases the range of matrices which can be successfully factorized[1].

To demonstrate this behavior we discuss a four dimensional problem, first presented in Domes & Neumaier [29], consisting of the single constraint

$$\begin{aligned} &4x_1^2 + 4Nx_1x_2 + 12x_1x_3 - 28x_1x_4 + (1+N^2)x_2^2 + (6N-2)x_2x_3 \\ &\quad -(10N+14)x_2x_4 + 11x_3^2 - 32x_3x_4 + 75x_4^2 + 2x_2 + 2Nx_3 + 26 \quad \leq \quad 0. \end{aligned} \tag{4.20}$$

Writing (4.20) in the form $x^T Ax + 2ax \leq -26$ with $x = (x_1, x_2, x_3, x_4)^T$, $2a = (0,\ 2,\ 2N, 0)$ and

$$A = B^T B, \text{ where } B := \begin{pmatrix} R & S \\ 0 & I \end{pmatrix}, \quad R = \begin{pmatrix} 2 & N \\ 0 & 1 \end{pmatrix} \text{ and } S = \begin{pmatrix} 3 & -7 \\ -1 & -5 \end{pmatrix},$$

we see that the symmetric matrix A is manifestly positive definite. Thus (4.20) describes an ellipsoid. If N is chosen large enough, A is very ill-conditioned.

For example if we choose $N = 5 \cdot 10^6$ then the 2-norm condition number of A is approximately $5 \cdot 10^{21}$, therefore A is nearly singular; the lowest eigenvalue is approximately $5 \cdot 10^{-15}$. It is no surprise that for this matrix both the directed Cholesky factorization

[1] It is interesting that Rump [83] also mentions the importance of scaling of the matrix A in his algorithm for checking definiteness. He uses a minimum degree ordering and a scaling technique based on the results on van der Sluis method given in Higham [44, Chapter 7])

using the Gerschgorin test and the directed Cholesky factorization with pivoting fails (the reasons for this and both methods are explained in Section 4.5).

If we use the scaling algorithm ScaleLP from Domes & Neumaier [29] on the problem, we obtain

$$D = \mathrm{Diag}(10^6 \quad 1 \quad 10^6 \quad 10^5)$$

as scaling matrix for the variables. Here, scaling makes an essential difference since the scaled problem

$$x^T DADx + bDx \leq -26$$

has a 10^6 times lower condition number (approximately $6 \cdot 10^{15}$) which — however it is still high — it is small enough for the directed Cholesky factorization with pivoting to be successful. Therefore we obtain the factorization

$$PAP^T = R^T R + E$$

with

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}, R = \begin{pmatrix} 5.0 \cdot 10^6 & 3.0 \cdot 10^6 & -7.0 \cdot 10^5 & 2.0 \cdot 10^6 \\ 0 & 1.4 \cdot 10^6 & 3.5 \cdot 10^5 & 2.8 \cdot 10^{-1} \\ 0 & 0 & 3.7 \cdot 10^5 & 2.7 \cdot 10^{-1} \\ 0 & 0 & 0 & 1.1 \cdot 10^{-2} \end{pmatrix},$$

and with the positive semidefinite residual matrix

$$E := \begin{pmatrix} 3.5210^{-2} & 0 & 0 & 0 \\ 0 & 2.58 \cdot 10^4 & 2.44 \cdot 10^{-4} & 9.77 \cdot 10^{-4} \\ 0 & 2.44 \cdot 10^{-4} & 2.42 \cdot 10^3 & 2.44 \cdot 10^{-4} \\ 0 & 9.77 \cdot 10^{-4} & 2.44 \cdot 10^{-4} & 146 \cdot 10^{-3} \end{pmatrix}.$$

Finally, computing the interval hull by the method from Section 4.2 we find the bounds

$$x \in ([-146, 1442], \ [-5 \cdot 10^8, 5 \cdot 10^7], \ [-3 \cdot 10^{-5}, 10^{-12}], \ [-10^{-2}, 10^{-3}])^T$$

for the constraint (4.20).

As one can notice the second and the third diagonal entry of E is very large. This seem to be a contradiction since E is supposed to be very small. However the component wise relative error

$$\delta := \max_{i,j} \left| \frac{E_{ij}}{A_{ij}} \right| = 1.0317 \cdot 10^{-9},$$

indicates that E is a very small perturbation of A. Thus E is indeed *very small with respect to* A in the sense as defined in (1) in notation part of the first section.

4.4 Preprocessing constrained optimization problems

Optimization is a constantly developing, complex and important field of the numerical mathematics. The goal of solving an optimization problem is to find a local or a global minimum of the objective function $f(x)$, subject to the general constraints $G(x) \in \mathbf{w}$ (including equality and inequality constraints) and to the bound constraints $x \in \mathbf{x}$:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{s.t.} & G(x) \in \mathbf{w} \\ & x \in \mathbf{x}. \end{array} \tag{4.21}$$

If we search for a global minimum of the problem, it is called a *global optimization problem.*

If an $\hat{x} \in \mathbf{x}$ satisfies $G(\hat{x}) \in \mathbf{w}$, $\hat{x}$ is called a feasible point. If there is no objective function given or it is constant, the goal is to find a good enclosure of the set of all feasible points. In this case the problem is called a *constraint satisfaction problem.* Also in the case of a global search the first step is often to solve the constraint satisfaction problem in order to bound and reduce the search space as much as possible. In this chapter we show how the ellipsoid hull enclosure technique presented in the previous sections can be used for this purpose.

Several state of the art global optimization solvers (e.g., [88] or [86]) combine a number of methods and strategies to find one or more global solution of (4.21). Most of the techniques (e.g branch and bound, heuristics) require that the bound constraints $x \in \mathbf{x}$ are finite. If a problem lacks the desired bounds, the usual remedy is to set default upper and lower bounds on the variables, thus changing the problem. If the global minimum lies outside the default bounds, the solver cannot find the solution.

The main advantage of the enclosure techniques from Section 4.1 and Section 4.2 that it gives us the possibility to obtain rigorous bounds on some variables which are consequences of the constraints without the need of having explicit bounds on the variables.

Since the enclosures are rigorous, the method is also applicable in verified global optimization (e.g, Chapter 2, [54], [59]) and in computer assisted proofs (see e.g., [74]).

If we already have found a good and feasible point $\hat{x} \in \mathbf{x}$ we can apply Algorithm 4.1 to the constraint satisfaction problem

$$\begin{array}{ll} \text{minimize} & 1 \\ \text{s.t.} & f(x) \leq f(\hat{x}) \\ & G(x) \in \mathbf{w} \\ & x \in \mathbf{x}, \end{array} \tag{4.22}$$

and may obtain new bounds on the variables as well as on the objective function $f(x)$. This makes our method a valuable tool not only for preprocessing and solving constraint satisfaction problems but also for global optimization.

To enhance the results of Section 4.1 and Section 4.2 we discuss the application to optimization problems given in the form of (4.21). The m general constraints are interpreted as component-wise enclosures $G_i(x) \in \mathbf{w}_i$ $(i = 1 \dots m)$. This includes equality constraints if $\mathbf{w}_i = [\underline{w}_i, \overline{w}_i]$ is a degenerate interval with $\underline{w}_i = \overline{w}_i$, inequality constraints if one bound of $\mathbf{w}_i$ is infinite, and two sided inequalities $\underline{w}_i \leq G_i(x) \leq \overline{w}_i$ if both bounds are finite. If we have bounds on the objective function, it should be threated like an ordinary general constraint. Similarly, the n bound constraints are interpreted as enclosures $x_j \in \mathbf{x}_j$ with $j = 1 \dots n$. Again, fixed variables and one-sided bounds on the variables are included as special cases.

We may apply Algorithm 4.1 for each quadratic constraint $G_i(x) \in \mathbf{w}_i$ separately. Thereby only the finite bounds of $\mathbf{w}_j$ are taken into account resulting in one or two inequality constraints in the form of

$$x^T A x + 2a^T x \leq \alpha \tag{4.23}$$

with coefficients obtained by the Taylor expansion of $G_j(x)$ around $x = 0$,

$$\begin{aligned} A_{jk} &= \frac{1}{2}\frac{\partial^2 G_i}{\partial x_j \partial x_k}(0), \quad 2a_j = \frac{\partial G_i}{\partial x_j}(0), \quad \alpha = \overline{w}_j - G_j(0) \quad \text{if} \quad \overline{w}_j \text{ is finite} \\ A_{jk} &= -\frac{1}{2}\frac{\partial^2 G_i}{\partial x_j \partial x_k}(0), \quad 2a_j = -\frac{\partial G_i}{\partial x_j}(0), \quad \alpha = -\underline{w}_j - G_j(0) \quad \text{if} \quad \underline{w}_j \text{ is finite} \end{aligned}$$

The size of the problem can of course be restricted to those variables on which $G_j(x)$ actually depends.

If the constraint (4.23) is strictly convex we obtain new bounding box $\mathbf{u}$ on the variables. If we cut the original bound constraints $x \in \mathbf{x}$ with $\mathbf{u}$ we obtain the new bound constrains

$$x \in \hat{\mathbf{x}} \text{ with } \hat{\mathbf{x}}_i := \mathbf{u}_i \cap \mathbf{x}_i = [\max(\underline{x}_i, \underline{u}_i), \min(\overline{x}_i, \overline{u}_i)]. \tag{4.24}$$

Because $\mathbf{u}$ is bounded, the box $\hat{\mathbf{x}}$ is also bounded. If we process all quadratic $G_j(x) \in \mathbf{w}_j$, we obtain an interval enclosure of the intersection of all strictly convex quadratic constraint.

The method can be greatly enhanced by removing the linear variables. This is crucial in the presence of slack variables which are only linear in the constraints. If the linear variables are not removed the matrix A has a zero row and column, hence it is singular, therefore the directed Cholesky factorization will fail and we cannot compute new bounds. To remove the linear terms from the constraint

$$x^T A x + 2a^T x \leq \beta$$

we write

$$x_I^T \hat{A} x_I + 2a^T x_I + c^T x_J \leq \beta \tag{4.25}$$

with J being the index set of the variables which are only linear and I being the index set of the variables which have nonlinear terms in the constraint. The dimension of $\hat{A}$ and a is reduced to $n' := |I|$, and the dimension of c is $n'' := |J|$. We modify (4.25) by bounding and removing the linear variables and obtain

$$x_I^T \hat{A} x_I + 2a^T x_I \leq \beta + \sum_{j \in J} (-c_j \underline{x}_j). \tag{4.26}$$

Here bracketing the right hand side of the above expression yields a correct bound when evaluating it using floating-point arithmetic with upward rounding. We can now write the new inequality (4.26) in the form of (4.23), with

$$x := x_I, \quad A := \hat{A}, \quad \alpha := \beta + \sum_{j \in J} (-c_j \underline{x}_j)$$

and factorizing A using a directed Cholesky factorization method. From this point on, all steps are the same as above, except from the fact that we compute new bounds only on the remaining n' variables. This should be accounted for when cutting the resulting $|x| \leq u$ with the original box. Thus we compute $\hat{\mathbf{x}}_i$ as in (4.24) only for $i \in I$ and set the remaining $\hat{\mathbf{x}}_j := \mathbf{x}_j$ for all $j \in J$. Proceeding in this way allows us to handle a bigger class of problems, by avoiding unnecessary singularity of the matrix A.

In practice, many problems have nonquadratic constraints. These relaxations can be handled as above if convex quadratic relaxations of such constraints can be computed (see Skutella [93] and Rendl [77] for possible techniques). This further extends the scope of our methods.

4.5 Directed Cholesky factorization

Let A be a symmetric matrix. A *directed Cholesky factorization* of the matrix A is an approximate factorization $A \approx R^T R$ with nonsingular upper triangular R such that the error matrix $A - R^T R$ of the factorization is positive semidefinite. The matrix R is called a directed Cholesky factor of A.

Proposition 4.7. *A directed Cholesky factorization exists iff A is positive definite.*

Proof. If R is a directed Cholesky factor of A then $E := A - R^T R$ is positive semidefinite, therefore $A = E + R^T R$ is positive definite. Conversely, if A is positive definite, we may take for R the Cholesky factor of A then $A - R^T R = 0$ is positive semidefinite; hence R is a directed Cholesky factor of A. □

In finite precision arithmetic, however, R is usually not representable exactly, and simply rounding it is often not sufficient to make $A - R^T R$ positive semidefinite. Thus finding a directed Cholesky factorization needs additional considerations. To represent the general setting we factor a symmetric interval matrix $\mathbf{A} := [\underline{A}, \overline{A}] \in \overline{\mathbb{IR}}^{n \times n}$. This form also represents the case if a matrix is not exactly known, as it is the result of inaccurate measurements or computations. We present the following methods to compute a directed Cholesky factorization such that the residual matrix $\mathbf{A} - R^T R$ is very small with respect to the matrix $\overline{A}$.

4.5.1 Directed Cholesky factorization using the Gerschgorin test

Our first method for computing a directed Cholesky factorization for an interval matrix $\mathbf{A}$ is based on the Gerschgorin test[2]. If $\underline{A}_{ii} \leq 0$ for any $i \in \{1, \ldots, n\}$ then not all symmetric $A \in \mathbf{A}$ are positive definite and a factorization with a nonsingular R is not possible. If the lower bounds of the diagonal entries of $\mathbf{A}$ are positive, we choose a matrix $\tilde{A} \in \mathbf{A}$ and slightly perturb its diagonal entries by using a suitable chosen a priori error estimation constant σ. Then we apply the approximate Cholesky factorization $R^T R \approx \tilde{A}$ to the perturbed matrix. If the error estimation constant σ was chosen correctly, even positive but nearly indefinite matrices (where the approximate Cholesky factorization would fail for the unperturbed matrix) can be factorized. If the Cholesky factorization succeeds the error matrix $\mathbf{E} := \mathbf{A} - R^T R$ is computed by using interval arithmetic. Finally we test $\mathbf{E}$ for positive definiteness with the Gerschgorin test. Again the right choice of σ is crucial, since if it was chosen unnecessary large the increased width of the

[2] the Gerschgorin test is described in the notation of the first section of this chapter.

interval error matrix has a negative effect on the outcome of the Gerschgorin test. If the Gerschgorin test is positive then R is a directed Cholesky factor of the matrix $\mathbf{A}$.

The following algorithm summarizes the above consideration:

Algorithm 4.2 (DirCholG)**.**
Compute a directed Cholesky factorization of a symmetric interval matrix, using the Gerschgorin test:

1. *Let $\mathbf{A} = [\underline{A}, \overline{A}]$ be a symmetric n–dimensional interval matrix.*
2. *If $\underline{A}_{ii} \leq 0$ for some $i \in \{1, \ldots, n\}$ the factorization is not possible. Stop.*
3. *We define the matrix*

$$\tilde{A}_{ij} = \begin{cases} \overline{A}_{ij} & \text{if} \quad \overline{A}_{ij} \geq -\underline{A}_{ij} \text{ and } i \neq j \\ \underline{A}_{ij} & \text{otherwise}\,. \end{cases} \tag{4.27}$$

4. *Perturb the diagonal entries of the matrix $\tilde{A}$:*
   (a) *Generate a diagonal perturbation matrix D ($D_{ij} = 0$ for $i \neq j$) which depends on the diagonal entries of $\tilde{A}$ and the width of the interval matrix $\mathbf{A}$:*

$$D_{ii} := \tilde{A}_{ii} - \frac{\sum_{j=1}^{n} (\overline{A} - \underline{A}) u_j}{u_i}, \text{ where } u_i = 1/\tilde{A}_{ii} \text{ for all } 1 \leq i \leq n.$$

   (b) *Choose an approximate a priori error estimation constant σ such that the Cholesky factorization of $A' := \tilde{A} - \sigma D$ is positive definite enough even in a nearly indefinite case (suitable selections for σ are discussed later).*
5. *Compute $A' \approx R^T R$ approximately, and $\mathbf{E} := \mathbf{A} - R^T R$ by using interval arithmetic.*
6. *If $\underline{E}_{ii} \geq 0$ for all i and $\mathbf{E}$ is an H-matrix then $\mathbf{E}$ is positive definite (Gerschgorin test), the factorization is successful and the directed Cholesky factor R is returned.*

Proposition 4.8. *If Algorithm 4.2 is successful we obtain a directed Cholesky factor R such that for all symmetric $A \in \mathbf{A}$, $A - R^T R$ is positive definite.*

Proof. Let be R the matrix returned by the algorithm. Since we use interval arithmetic in Step 5 of Algorithm 4.2 the bound $\mathbf{E}$ on $\mathbf{A} - R^T R$ is rigorous. Since by Step 6 of Algorithm 4.2, $\mathbf{E}$ is a H-matrix, the Gerschgorin test implies the assertion. □

Comments on Algorithm 4.2:

(ad 3.) The algorithm would be also correct if $\tilde{A}$ in (4.27) is replaced by an arbitrary $\tilde{A} \in \mathbf{A}$ with $\tilde{A}_{ii} = \underline{A}_{ii}$.

(ad 4.) The perturbation applied to the diagonal entries of $\tilde{A}$ is needed for nearly indefinite matrices. By using this approach we may obtain the approximate Cholesky factor of the perturbed matrix, even if we would fail for the unperturbed one.

(ad 6.) The test whether or not the matrix $\mathbf{E}$ is an H-matrix can be done by choosing a suitable $u > 0$ and test whether or not $\langle \mathbf{E} \rangle u > 0$ holds. Different choices of u are

- $u = (1, \ldots, 1)^T$ the simplest, proving diagonal dominance, but not scaling invariant,
- $u \approx 1/\operatorname{diag}(E)$, is a generally good and cheap choice,
- $u \approx \langle E \rangle^{-1} (1, \ldots, 1)^T$, is the best choice (see Neumaier [70]), but requires $O(n^3)$ operation for solving the linear system.

The selection of an approximate a priori error estimation constant σ is critical for nearly indefinite matrices. If we choose σ too small, the approximate Cholesky factorization will possibly fail; if we choose it to large, the error matrix $\mathbf{E}$ will be too large and it will not pass the H-matrix test.

The following theorem which can be found in Higham [44, pp. 203–224] gives information about the feasibility of a numerical Cholesky factorization when all arithmetic operations are executed with a relative error of at most ε (when no overflow or underflow occurs).

Theorem 4.9 (Demmel)**.** *Let $A \in \mathbb{R}^{n \times n}$ be a symmetric matrix with positive diagonal elements, and a diagonal matrix D with $D_{ii} = A_{ii}^{-1/2}$. If*

$$\lambda_{min}(DAD) > \sigma := \frac{n(n+1)\varepsilon}{1 - 2(n+1)\varepsilon} \tag{4.28}$$

then the Cholesky factorization applied to A succeeds and produces a nonsingular R. If $\lambda_{min}(A) \leq -\sigma$ then the computation is certain to fail.

Theorem 4.9 seems to give a good choice for σ, but in reality it is significantly larger than it would be needed to successfully factor nearly indefinite matrices by the approximate Cholesky factorization. This makes it harder to pass Gerschgorin test. By our heuristic experiments we found a more suitable choice for σ; we try $\sigma = \varepsilon(0.015\mathrm{nnz}(\mathbf{A}) + 0.5n)$

in the first run, then in the case of failure $\sigma = \varepsilon(0.015\text{nnz}(\mathbf{A}) + n)$ in the second one. The results of this strategy are satisfactory.

4.5.2 Directed Cholesky factorization with pivoting

For a symmetric, positive definite interval matrix $\mathbf{A}$, the *directed Cholesky factorization with diagonal pivoting* computes a permutation matrix P and an upper triangular matrix R such that for every $A \in \mathbf{A}$, the residual matrix $E := PAP^T - R^TR$ is positive semidefinite and is very small with respect to A. We first state the algorithm, then discuss the conditions under which the residual matrix is positive semidefinite and is very small.

The following algorithm either computes a directed Cholesky factor R and a permutation matrix P such that the residual matrix E is positive semidefinite and is very small with respect to A, or it terminates with an error message and returns an incomplete factorization:

Algorithm 4.3 (DirCholP)**.**
Directed Cholesky factorization of symmetric interval matrix, using directed rounding and diagonal pivoting.

1. *Let* $\mathbf{A} = [\underline{A}, \overline{A}]$ *be an* n*–dimensional symmetric interval matrix. Set* $\mathbf{A}_1 = \mathbf{A}$*,* $R = 0_n$*,* $P = I_n$*, and the rounding mode to upward rounding.*

2. *For* $k = 1, \ldots, n$ *do the following steps:*

   (a) *Find the pivot element* $\alpha = \max(\text{diag}(\underline{A}_k))$ *on the diagonal of the matrix* $\underline{A}_k \in \mathbb{R}^{n-k+1}$*. Let* j *denote the index of the pivot element; interchange row* j *with the first row and column* j *with the first column, in the interval matrix* $\mathbf{A}_k$*.* $\mathbf{A}_k$ *remains symmetric. Exchange the same rows and columns in the matrix* P*.*

   (b) *Partition the permuted interval matrix* $\mathbf{A}_k$ *as:*

$$\mathbf{A}_k = \begin{pmatrix} \boldsymbol{\alpha}_k & \mathbf{a}^T \\ \mathbf{a}_k & \mathbf{B}_k \end{pmatrix}.$$

   (c) *If* $\underline{\alpha}_k \leq 0$ *terminate Step 2. and return an error message.*

   (d) *Choose* $\gamma_k < 1$ *and* $\rho_k = \gamma_k\sqrt{\underline{\alpha}_k}$ *and* $r_k = (\overline{a}_k + \underline{a}_k)/(2\rho_k)$*.*

   (e) *Set* $R_{kk} = \rho_k$ *and* $R_{k,k:n} = r_k^T$*.*

   (f) *Compute* $\delta_k := \underline{\alpha}_k + \rho_k(-\rho_k)$ *and* $d_k := \max(\overline{a}_k + \rho_k(-r_k),\ \rho_k r_k - \underline{a}_k)$*.*

(g) If the residual pivot $\delta_k \leq 0$ terminate Step 2. and return an error message.

(h) Set $\mathbf{A}_{k+1} := [\underline{B}_k - r_k r_k^T - d_k d_k^T/\delta_k,\ \overline{B}_k + (-r_k) r_k^T + d_k d_k^T/\delta_k]$.

3. *If Step 2. is finished without an error message we obtain the upper triangular matrix R and the permutation matrix P, if an error message was produced the incomplete factorization is returned.*

Theorem 4.10. *Suppose that Algorithm 4.3 used for the symmetric interval matrix* $\mathbf{A}$ *terminates without an error message and returns the matrix R. Then $PAP^T - R^T R$ is positive semidefinite for all symmetric $A \in \mathbf{A}$.*

To prove the proposition we need some preparations:

Proposition 4.11. *Let*

$$A := \begin{pmatrix} \alpha & a^T \\ a & B \end{pmatrix} \in \mathbf{A} := \begin{pmatrix} \boldsymbol{\alpha} & \mathbf{a}^T \\ \mathbf{a} & \mathbf{B} \end{pmatrix} \in \mathbb{IR}^{n\times n}, \tag{4.29}$$

then:
(i) For arbitrary $\rho \in \mathbb{R}$, $|\rho| < \sqrt{\underline{\alpha}}$, $r \in \mathbb{R}^{n-1}$ and

$$\varepsilon := \alpha - \rho^2 > 0, \quad e := a - \rho r, \quad A_0 := B - rr^T - \frac{ee^T}{\varepsilon}, \tag{4.30}$$

we have

$$A = \begin{pmatrix} \rho \\ r \end{pmatrix} \begin{pmatrix} \rho \\ r \end{pmatrix}^T + \begin{pmatrix} \varepsilon & e^T \\ e & ee^T/\varepsilon \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ 0 & A_0 \end{pmatrix}. \tag{4.31}$$

(ii) The bounds $\varepsilon \geq \delta$ and $|e| \leq d$ and $A_0 \in \mathbf{A}_0$ are satisfied if

$$\begin{aligned} &0 < \delta \leq \underline{\alpha} + \rho(-\rho), \\ &d \geq \max(\overline{a} + \rho(-r), \rho r - \underline{a}), \\ &\mathbf{A}_0 \supseteq [\underline{B} - rr^T - dd^T/\delta,\ \overline{B} + (-r)r^T + dd^T/\delta]. \end{aligned} \tag{4.32}$$

Proof. (i) Since $|\rho| < \sqrt{\underline{\alpha}}$ we have $\varepsilon = \alpha - \rho^2 > \alpha - \underline{\alpha} \geq 0$ for all $\alpha \in \boldsymbol{\alpha}$, so that $\varepsilon > 0$. Thus (4.31) is well defined. Substituting (4.30) into (4.31) gives

$$A = \begin{pmatrix} \rho^2 + \varepsilon & \rho r^T + e^T \\ \rho r + e & rr^T + ee^T/\varepsilon + A_0 \end{pmatrix} = \begin{pmatrix} \alpha & a^T \\ a & B \end{pmatrix}.$$

(ii) By (4.30) we have

$$\varepsilon = \alpha - \rho^2 \geq \underline{\alpha} + \rho(-\rho) \geq \delta > 0, \tag{4.33}$$

$$\begin{aligned} &e = a - \rho r \le \overline{a} + \rho(-r), \quad -e = -a + \rho r \le \rho r - \underline{a}, \\ &|e| \le \max(\overline{a} + \rho(-r), \rho r - \underline{a}) \le d, \end{aligned} \tag{4.34}$$

and

$$A_0 = B - rr^T + ee^T/\varepsilon.$$

Since $|ee^T/\varepsilon| \le dd^T/\delta$ by (4.33) and (4.34) we find that

$$A_0 \ge \underline{B} - rr^T - dd^T/\delta \ge \underline{A}_0, \quad A_0 \le \overline{B} + (-r)r^T + dd^T/\delta \le \overline{A}_0,$$

resulting in $A_0 \in \mathbf{A}_0$. □

Evaluating the right hand side of (4.32) in finite precision arithmetic, with directed rounding and priorities given by the parentheses, results in the correct bounds $\underline{A}$, $\overline{A}$, δ and d which satisfy (4.32).

We now use the Proposition 4.11 to prove the following proposition, which is then used in the induction proof of Theorem 4.10.

Proposition 4.12. *Suppose that for some real constants δ, ε and ρ, for some $(n-1)$-dimensional vectors d, r and e, for*

$$\mathbf{A} := \begin{pmatrix} \boldsymbol{\alpha} & \mathbf{a}^T \\ \mathbf{a} & \mathbf{B} \end{pmatrix} \in \mathbb{IR}^{n\times n}, \tag{4.35}$$

and for some symmetric interval matrix $\mathbf{A}_0 \in \mathbb{IR}^{(n-1)\times(n-1)}$ the inequalities

$$\begin{aligned} &|\rho| < \sqrt{\underline{\alpha}}, \\ &0 < \delta \le \underline{\alpha} + \rho(-\rho), \\ &d \ge \max(\overline{a} + \rho(-r), \rho r - \underline{a}), \\ &\mathbf{A}_0 \supseteq [\underline{B} - rr^T - dd^T/\delta,\ \overline{B} + (-r)r^T + dd^T/\delta], \end{aligned} \tag{4.36}$$

are satisfied. If for all symmetric matrices $A_0 \in \mathbf{A}_0$ an $R_0 \in \mathbb{R}^{(n-1)\times(n-1)}$ exists such that $A_0 - R_0^T R_0$ is positive semidefinite, then for every symmetric matrix $A \in \mathbf{A}$ an $R \in \mathbb{R}^{n\times n}$ exists such that $A - R^T R$ is positive semidefinite.

Proof. By assumption, the Cholesky factorization

$$LL^T = A_0 - R_0^T R_0 \tag{4.37}$$

exists, with a lower triangular matrix $L \in \mathbb{R}^{n\times n}$.

Since by (4.35) every symmetric $A \in \mathbf{A}$ can be written as (4.29), (4.30) holds by Proposition 4.11 for arbitrary $\rho \in \mathbb{R}$, $|\rho| < \sqrt{\underline{\alpha}}$, $r \in \mathbb{R}^{n-1}$ and

$$\varepsilon := \alpha - \rho^2, \quad e := a - \rho r, \quad A_0 := B - rr^T - \frac{ee^T}{\varepsilon}.$$

By (4.36) the bounds $\varepsilon \geq \delta > 0$ and $A_0 \in \mathbf{A}_0$ are satisfied.

If we substitute (4.37) into (4.30), we get

$$\begin{aligned} A &= \begin{pmatrix} \rho^2 + \varepsilon & \rho r^T + e^T \\ \rho r + e & rr^T + ee^T/\varepsilon + R_0^T R_0 + LL^T \end{pmatrix} = \\ &= \begin{pmatrix} \rho^2 & \rho r^T \\ \rho r & rr^T + R_0^T R_0 \end{pmatrix} + \begin{pmatrix} \varepsilon & e^T \\ e & ee^T/\varepsilon + LL^T \end{pmatrix} = \\ &= \begin{pmatrix} \rho & 0 \\ r & R_0 \end{pmatrix} \begin{pmatrix} \rho & r^T \\ 0 & R_0 \end{pmatrix} + \begin{pmatrix} \varepsilon & 0 \\ e & L \end{pmatrix} \begin{pmatrix} 1/\varepsilon & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} \varepsilon & e^T \\ 0 & L \end{pmatrix} = \\ &= R^T R + S^T D S, \end{aligned}$$

with

$$R = \begin{pmatrix} \rho & r^T \\ 0 & R_0 \end{pmatrix}, \quad S = \begin{pmatrix} \varepsilon & e^T \\ 0 & L^T \end{pmatrix}, \quad D = \begin{pmatrix} 1/\varepsilon & 0 \\ 0 & I \end{pmatrix}.$$

Since $\varepsilon \geq \delta > 0$ the matrix D is positive semidefinite and

$$x^T(A - R^T R)x = x^T(S^T D S)x = (Sx)^T D S x = (Sx)^T D S x \geq 0.$$

holds, proving the assertion. □

We are now prepared to prove that for all symmetric $A \in \mathbf{A}$ the residual matrix of the directed Cholesky factorization computed by Algorithm 4.3 is positive semidefinite:

Proof of Theorem 4.10: First we show by induction that the interval matrices $\mathbf{A}_k$, $k = 1, \ldots, n$ constructed by Algorithm 4.3 are symmetric. Without loss of generality we may assume that A_k is already permuted, such that no further pivoting is required ($P = I_n$).

$\mathbf{A}_1 = \mathbf{A}$ is symmetric by definition. Assuming that the interval matrix $\mathbf{A}_k$ is symmetric, $\mathbf{B}_k$ is also symmetric as an $(n-k) \times (n-k)$ submatrix of $\mathbf{A}_k$. For arbitrary vectors r and d, the matrices rr^T and dd^T are symmetric by construction. Therefore by (h) of Algorithm 4.3, the interval matrix $\mathbf{A}_{k+1} \in \mathbb{IR}^{(n-k)\times(n-k)}$ is symmetric. From this follows that each $\mathbf{A}_k$ can be factorized as in (4.35).

First we assume that every computation is exact and prove by induction on $m := n-k+1$ that $A_k - R_k^T R_k$ is positive semidefinite for each symmetric $A_k \in \mathbf{A}_k$.

For $m = 1$, $k = n$, $R_n^T R_n = \rho_n^2 = \gamma_n^2 \alpha_n$ and $A_n = \alpha_n \in \boldsymbol{\alpha}_n$ and since $\alpha_n \geq \underline{\alpha}_n > 0$ and $0 < \gamma_n < 1$,

$$A_n - R_n^T R_n = \alpha_n - \gamma_n^2 \alpha_n = \alpha_n(1 - \gamma_n^2) > 0$$

is positive semidefinite for all $A_n \in \mathbf{A}_n$.

We now assume for all symmetric matrices $A_{k+1} \in \mathbf{A}_{k+1}$ an $R_{k+1} \in \mathbb{R}^{n-k \times n-k}$ exists such that $A_{k+1} - R_{k+1}^T R_{k+1}$ is positive semidefinite. Since (d), (f) and (h) of Algorithm 4.3 imply (4.36) with $\delta = \delta_k$, $\varepsilon = \varepsilon_k$, $\rho = \rho_k$, $r = r_k$, $e = e_k$, $\mathbf{A} = \mathbf{A}_k$ and $\mathbf{A}_0 = \mathbf{A}_{k+1}$, we can find for every symmetric matrix $A_k \in \mathbf{A}_k$ an $R_k \in \mathbb{R}^{n-k+1 \times n-k+1}$ such that $A_k - R_k^T R_k$ is positive semidefinite.

By induction, this holds for $m = n$, $k = 1$ when $A = A_1$ and $R = R_1$ proving that $PAP^T - R^T R$ is positive semidefinite for $A \in \mathbf{A}$.

In finite precision arithmetic, the results satisfy the required inequalities in Proposition 4.12 for d_k δ_k and $\mathbf{A}_{k+1}$ if the right hand sides of the inequalities in (4.36) are computed with directed rounding.

By successfully factoring a symmetric positive semidefinite interval matrix $\mathbf{A}$ by Algorithm 4.3 we obtain a matrix R such that for all symmetric $A \in \mathbf{A}$ the residual matrix $S := PAP^T - R^T R$ is positive semidefinite. In addition to this we also expect (and our numerical experiments show that it is typically true) that S is very small with respect to A (for a suitable tolerance, e.g. $\kappa = 10^{-6}$). The choices of ρ_k, r_k and γ_k in Algorithm 4.3 were made to satisfy this criteria:

- To make S positive semidefinite, we had to ensure that $\varepsilon > 0$. Therefore we needed $\delta_k > 0$ which is the case when, $|\rho_k| < \sqrt{\underline{\alpha}_k}$. If we additionally want δ_k to be very small and assume that $\underline{\alpha}_k > 0$ (which is true if $\underline{A}$ is positive definite), we can set $\rho_k = \gamma_k \sqrt{\underline{\alpha}_k}$ with $\gamma_k < 1$. If in addition to this we choose $\gamma_k \approx 1$, the condition that $\delta_k \approx 0$ is also satisfied.

- The entries of $d_k = a_k - \rho_k r_k$ can be made to vanish by setting $r_k := a_k/\rho_k$. Even when r_k and ρ_k are computed inaccurately we can set $r_k = (\overline{a}_k + \underline{a}_k)/(2\rho_k)$ using the midpoint of the interval $\mathbf{a}_k$ to get a very small d_k.

- To make $d_k^T d_k/\varepsilon_k$ very small, we also have to guarantee that $d_k^T d_k \ll \delta_k$. In case of rounding errors $d_k^T d_k = (a_k - (\rho_k a_k)/\rho_k)^T (a_k - (\rho_k a_k)/\rho_k) = O(\epsilon)$ with ϵ representing the machine precision, so $1 \gg \delta_k = \alpha_k - \gamma_k^2 \alpha_k \gg \epsilon$ should be satisfied.

Since $a_k \in \mathbf{a}_k$ the width of $\mathbf{a}_k$ should be accounted for, too, heuristic choice for the regularization term γ_k is

$$\begin{aligned}\gamma_k &= 1 - \min((\sqrt{n}+1)\zeta_k, 0.01),\\ \zeta_k &= \epsilon + \max(\mathrm{mid}(\mathbf{a}_k)/w),\\ w_i &= \sqrt{\underline{\alpha}_k(\underline{A}_k)_{ii}},\end{aligned}$$

where n is the dimension of A. This choice will usually produce good results.

Using these choices in Algorithm 4.3 makes the residual matrix not only positive semidefinite but also very small with respect to A for all $A \in \mathbf{A}$.

While the Cholesky factorization is numerically stable without pivoting, it is of advantage to use a permuted version. To enhance the robustness of the factorization we use *diagonal pivoting*. Thus in each step we permute two rows and the corresponding two columns of the matrix A_k in order to have the maximum of all diagonal entries as the pivot element α, while retaining the symmetric structure of the matrix. In our implementation, the diagonal pivoting can be turned off in order to reduce the time needed for the factorization in case of very big matrices. Testing has shown that when we turn off the pivoting, the factorization will fail more often, and the reduction of the computation time is not really significant unless the dimension is huge.

4.6 Testing the directed Cholesky factorization

We tested the new methods on random real interval matrices of different dimension (column `dim` in the tables below), width (column `width` in the tables below) and density (column `density` in the tables below). These matrices are constructed to be positive definite but nearly singular, with a very small inverse condition number (column `icond` in the tables below). For the inverse condition number we take the median of the quotients of the absolute value of the smallest eigenvalues and the absolute value of the largest ones of all k test matrices, where the eigenvalues are approximately computed by MATLAB, formally:

$$\texttt{icond} := \operatorname*{med}_i \left(\frac{|\lambda_{min}(\underline{A}_i)|}{|\lambda_{max}(\underline{A}_i)|} \right),\ i \in \{1, \ldots, k\}.$$

The following algorithm shows how the test matrices are created:

Algorithm 4.4 (Nearly singular positive definite interval matrix generator)**.**
Given is the dimension n, a tiny singularity factor η (e.g. $\eta = 10^{-12}$) and the required relative width $\delta \geq 0$ of the interval matrix $\mathbf{A}$ to be created.

1. *Generate a random matrix* $B \in \mathbb{R}^{n-1\times n}$ *with* $B_{ij} \in [-1,1]$ *for all* $i = 1,\ldots,n-1$ *and* $j = 1,\ldots,n$.
2. *Generate an random vector* $u \in \mathbb{R}^n$ *with* $u_j \in [-1,1]$ *for all* $j = 1,\ldots,n$.
3. *Divide* u *by* $\max(u)$ *such that* $\|u\|_2 = 1$ *holds.*
4. *Compute* $C = B^T B \in \mathbb{R}^{n\times n}$ *and* $d = max(C_{ii})$.
5. *If* $d = 0$ *start again with step 1.*
6. *Else set* $\underline{A} = C/d + \eta uu^T$ *and* $\overline{A} = \underline{A} + \delta|\underline{A}|$ *and return the interval matrix* $\mathbf{A} := [\underline{A}, \overline{A}] \in \mathbb{IR}^{n\times n}$.

The tests show that both methods can be used to verify the positive definiteness and to decompose ill-conditioned matrices into their directed Cholesky factors. We first

show that the approximative method factors all the matrices and the directed methods factor nearly all of them. Comparison of the approximate Cholesky factorization of MATLAB (row CHOL in the tables below), the directed Cholesky factorization based on the Gerschgorin test (computed by Algorithm 4.2, row DIRCHOLG in the tables below) and the directed Cholesky factorization based with diagonal pivoting (computed by Algorithm 4.3, row DIRCHOLP in the tables below) on 500, 20–dimensional real matrices with a small inverse condition number:

method	dim	density	width	sfact	iters	icond	solved
CHOL	20	100%	0	1e-012	500	1.3e-016	100%
DIRCHOLH	20	100%	0	1e-012	500	1.4e-016	94%
DIRCHOLP	20	100%	0	1e-012	500	1.5e-016	88%

The next few tests of the directed Cholesky factorization based on the Gerschgorin test show increasing the dimension or the width makes the factorization more difficult, while more sparsity makes it easier. Test of the directed Cholesky factorization based using the Gerschgorin test on 500 real matrices of different dimensions (50,100,200) with a small inverse condition number:

method	dim	density	width	sfact	iters	icond	solved
DIRCHOLH	10	100%	0	1e-012	500	1.7e-016	95%
DIRCHOLH	40	100%	0	1e-012	500	1.3e-016	90%
DIRCHOLH	100	100%	0	1e-012	500	1e-016	74%

Test of the directed Cholesky factorization using the Gerschgorin test on 500 real interval matrices of different dimensions (50,100,200), different average density and with a small inverse condition number:

method	dim	density	width	sfact	iters	icond	solved
DirCholH	10	46%	0	1e-012	500	1.3e-016	100%
DirCholH	40	39%	0	1e-012	500	5.6e-017	100%
DirCholH	100	38%	0	1e-012	500	3.7e-017	70%

Test of the directed Cholesky factorization using the Gerschgorin test on 500 real interval matrices of width $1e-014$ of different dimensions (50,100,200) with a small inverse condition number:

method	dim	density	width	sfact	iters	icond	solved
DirCholH	10	100%	1e-014	1e-012	500	1.7e-016	82%
DirCholH	40	100%	1e-014	1e-012	500	1.3e-016	67%
DirCholH	100	100%	1e-014	1e-012	500	9.1e-017	55%

The last five tests Cholesky factorization with diagonal pivoting show similar results with respect to the increasing dimension and more sparsity. We can also see that the results of this factorization method are not as good as the results of the directed Cholesky factorization using the Gerschgorin test. Since the most applications are not as ill-conditioned as the problems in our test set and this method also returns an incomplete factorization it is still interesting. The fourth test is done in order to show the correlation between the dimension and the singularity factor, while the last one shows the effect if the pivoting is turned off. Test of the directed Cholesky factorization with diagonal pivoting on 500 real matrices of different dimensions (50,100,200) with a small inverse condition number:

method	dim	density	width	sfact	iters	icond	solved
DirCholP	10	100%	0	1e-012	500	1.8e-016	93%
DirCholP	40	100%	0	1e-012	500	1.3e-016	78%
DirCholP	100	100%	0	1e-012	500	1.1e-016	65%

Test of the directed Cholesky factorization with diagonal pivoting on 500 real interval matrices of different dimensions (50,100,200), different average density and with a small inverse condition number:

method	dim	density	width	sfact	iters	icond	solved
DirCholP	10	46%	0	1e-012	500	1.3e-016	100%
DirCholP	40	40%	0	1e-012	500	5.5e-017	93%
DirCholP	100	36%	0	1e-012	500	3.9e-017	11%

Test of the directed Cholesky factorization with diagonal pivoting on 500 real interval matrices of with $1e-014$ of different dimensions (50,100,200) with a small inverse condition number:

method	dim	density	width	sfact	iters	icond	solved
DirCholP	10	100%	1e-014	1e-012	500	1.7e-016	78%
DirCholP	40	100%	1e-014	1e-012	500	1e-016	65%
DirCholP	100	100%	1e-014	1e-012	500	1.1e-016	54%

Test of the correlation between the inverse condition number and the dimension for the directed Cholesky factorization with diagonal pivoting on 500 real matrices:

method	dim	density	width	sfact	iters	icond	solved
DirCholP	10	100%	0	1e-013	500	1.9e-017	79%
DirCholP	40	100%	0	1e-012	500	1.1e-016	76%
DirCholP	100	100%	0	3e-011	500	2.9e-015	92%

Test of the directed Cholesky factorization with diagonal pivoting on 500 real interval matrices of with $1e-014$ of different dimensions (50,100,200) with a small inverse condition number (pivoting turned off):

method	dim	density	width	sfact	iters	icond	solved
DirCholP(0)	10	100%	0	1e-012	500	1.7e-016	92%
DirCholP(0)	40	100%	0	1e-012	500	1.1e-016	77%
DirCholP(0)	100	100%	0	1e-012	500	1.1e-016	65%

4.7 Verification of positive definiteness

Proposition 4.7 shows that the existence of a directed Cholesky factorization of a symmetric matrix A implies that A is positive definite, and that of a symmetric interval matrix $\mathbf{A}$ implies that all symmetric matrices $A \in \mathbf{A}$ are positive definite. On the other

hand, if the directed factorization fails, **A** either contains a singular or indefinite, or a very ill-conditioned matrix. Many of the latter cases can still be verified when we apply an appropriate scaling before verifying positive definiteness, see Section 4.3.

Such definiteness test are useful independent of the goal of this chapter, for several applications ranging from the solution of linear interval equations (see below) over semidefinite programing problems VANDENBERGH & BOYD [100] to the representation theory of Lie groups (ADAMS [1]).

Any test for the positive definiteness of real symmetric matrices can easily be extended to a test for complex Hermitian matrices, using the following result; no complex arithmetic is required.

Theorem 4.13. *A matrix $H = A + iB$ with $A, B \in \mathbb{R}^{n\times n}$ is Hermitian and positive definite iff the real matrix*

$$C := \begin{pmatrix} A & -B \\ B & A \end{pmatrix} \tag{4.38}$$

is symmetric and positive definite.

Proof. The matrix C is symmetric iff $A^T = A$ and $B^T = -B$, and this holds iff H is Hermitian. H is positive definite iff

$$(x + iy)^*(A + iB)(x + iy) > 0 \text{ whenever } \begin{pmatrix} x \\ y \end{pmatrix} \neq 0. \tag{4.39}$$

Now

$$\begin{aligned}(x + iy)^*(A + iB)(x + iy) &= (x - iy)^T(A + iB)(x + iy) \\ &= x^TAx + y^TAy + y^TBx - x^TBy + i(x^TAy - y^TAx + x^TBx + y^TBy) \\ &= x^TAx + y^TAy - 2x^TBy = \begin{pmatrix} x \\ y \end{pmatrix}^T C \begin{pmatrix} x \\ y \end{pmatrix}\end{aligned}$$

since

$$\begin{aligned} x^TAy &= (x^TAy)^T = y^TA^Tx = y^TAx, \\ y^TBx &= (y^TBx)^T = x^TB^Ty = -x^TBy, \\ x^TBx &= (x^TBx)^T = x^TB^Tx = -x^TBx \quad \Rightarrow \quad x^TBx = 0, \\ y^TBy &= (y^TBy)^T = y^TB^Ty = -y^TBy \quad \Rightarrow \quad y^TBy = 0. \end{aligned}$$

Thus (4.39) holds iff C is positive definite. □

We also note that all the results from Section 4.5 could be developed for the complex case.

Rump [79–81, 83] gave criteria for the definiteness of interval matrices in the context of solving linear interval equations. Here we discuss only his most recent work [83]. His method is based on a single floating-point Cholesky factorization; all possible computational and rounding errors, including underflow, are taken into account via a floating-point error analysis. To find an error estimation of the Cholesky factorization, Rump presents three different selection methods. These error estimations are worst case bounds; so when they are used to perturb the diagonal entries of the matrix A and the approximative Cholesky factorization is successful, the positive definiteness of A is guaranteed. Uncertainties in the matrix are accounted for only coarsely by bounding them in the Frobenius norm. Rump & Ogita [84] reduce the computational overhead in Rump's method, but only for exactly given floating-point matrices A.

In contrast, in our directed Cholesky factorization using the Gerschgorin test, the perturbation terms are based on heuristics that account for the typical case rather than a worst case floating-point analysis. To justify the heuristic choice, the actual verification is done by the additional Gerschgorin test. The directed Cholesky factorization with diagonal pivoting is based on different principles and is not directly comparable with Rump's approach. It is likely that the ideas of Rump can be combined with directed Cholesky factorizations to get improved enclosures for linear systems with positive definite interval coefficient matrices.

The following alternative test for positive definiteness of symmetric interval matrices is given in Neumaier [73, p. 32].

Theorem 4.14. *Let* $\mathbf{A}$ *be a symmetric interval matrix.*

(i) If some symmetric matrix $A \in \mathbf{A}$ *is positive definite and all symmetric matrices in* $\mathbf{A}$ *are nonsingular then they are all positive definite.*

(ii) In particular, this holds if the midpoint matrix

$$\hat{A} = (\overline{A} + \underline{A})/2$$

is positive definite with inverse C*, and the preconditioned radius matrix*

$$\Delta = |C| \operatorname{rad}(\mathbf{A});$$

satisfies (in an arbitrary norm) the condition

$$\|\Delta\| < 1.$$

Since verifying definiteness is not the focus of this chapter we refrain from giving numerical comparison of the various definiteness tests.

Remark: The Sections 4.1 and 4.2 were originally written by Arnold Neumaier then modified my myself, the idea of the directed Cholesky factorizations also comes from Arnold Neumaier, the proofs are made by myself. The remaining Sections are my work.

Chapter 5

Rigorous filtering using linear relaxations

Abstract. This chapter presents rigorous filtering methods for continuous constraint satisfaction problems based on linear relaxations. Filtering or pruning stands for reducing the search space of constraint satisfaction problems. We discuss partially improved and existing methods as well as new approaches for rigorously enclosing the solution set of linear systems of inequalities. We also discuss different methods for computing linear relaxations. This allows one to customize combinations of relaxation and filtering. Care is taken to ensure that all methods correctly account for rounding errors in the computations.

Although most of the results apply more generally, strong emphasis is given to relaxing and filtering quadratic constraints, as implemented in the GloptLab environment, which internally exploits a quadratic structure. Demonstrative examples and tests comparing the different linear relaxation methods are also presented.

Keywords. linear relaxations, filtering, pruning, continuous constraints, quadratic constraint satisfaction problems, rounding error control, verified computation, quadratic programming, branch and bound, global optimization.

5.1 Introduction

Context. This chapter consider rigorous filtering methods based on computing linear relaxations for continuous constraint satisfaction problems. A constraint satisfaction problem is the task of finding one or all points satisfying a given family of equations and/or

inequalities, called constraints. Many real word problems are continuous constraint satisfaction problems, often high dimensional ones. Applications include robotics (Grandon et al. [37], Merlet [66]), localization and map building (Jaulin [49], Jaulin et al. [50], biomedicine Cruz & Barahona [23]), or the protein folding problem (Krippahl & Barahona [58]). In practice, constraint satisfaction problems are solved by a combination of a variety of techniques, often involving constraint propagation with either some form of stochastic search or a branch and bound scheme for a complete search. These techniques are mainly complemented by filtering or pruning techniques based on techniques borrowed from optimization, such as linear or convex relaxations (see, e.g. Neumaier [73]).

Filtering or pruning stands for reducing the search space of constraint satisfaction problems. There are many filtering techniques which are usually combined with branch and bound methods and provide more or less reduction of the search space. If applied to quadratic constraints the classical filtering algorithms which are based upon local consistencies like 2B-consistency or Box-consistency (see, e.g., Benhamou et al. [14]) do not take advantage of the special properties of quadratic forms therefore often yield poor results. 3B-consistency is more effective, but the practice shows that for quadratic problems they usually tend to be slow due to the exhaustive branching needed to achieve the required precision. Hull consistency techniques like the classical HC4 (Benhamou et al. [12]) or the newly developed OCTUM (Chabert & Jaulin [20]) show promising results, but still do not use the special structure of quadratic problems.

Another approach — originally developed for global optimization — is to compute linear relaxations for a problem, and then use these to reduce the search space of the original problem. As the name suggests, we may also lose some structural information of the original problem, by computing a relaxation. However this loss is often complementary to that of the consistency techniques described before. Here only linear relaxations are considered, higher degree relaxations and convex relaxations are discussed in the literature; for example, affine and convex functions for non-convex multivariate polynomials in Garloff et al. [36]. Constructing relaxations by using the right method can effectively approximate the structure of the original constraint; in fact, the criteria for a good relaxation is having a minimal distance (in some sense) to the original constraints. The resulting linear system usually contains more variables/constraints then the original problem but is much easier to solver than the original problem. A classical method, called RLT (reformulation-linearization technique) by Sherali & Adams [92] is used by Lebbah et al. [60] in the QUAD algorithm; another interesting approach was given by Kolev [57]. Since the last two fit the main scope of this chapter they will be discussed in detail. The above methods often require rigorous solutions of linear programs, as for example given for programs with uncertain data Jansson & Rump [47].

Software. A number of software packages for solving constraint satisfaction problems make extensive use of linear relaxations. The ICOS solver by Lebbah [59] is a free software package for solving nonlinear and continuous constraints, based on constraint programming, relaxation and interval analysis techniques. The prize winning, commercial solver Baron by Sahinidis & Tawarmalani [86] — a highly developed, approximate global optimization solver — uses a special linear relaxation technique called the sandwich method, while the COCONUT Environment [88, 89] applies both linear relaxations using slopes and reformulation-linearization on DAGs (Directed Acyclic Graphs).

Note that solving constrained global optimization problems by branch and bound is in practice reduced to solving a sequence of constraint satisfaction problems, each obtained by adding a constraint $f(x) \leq f_{\text{best}}$ to the original constraints, where f is the objective function and f_{best} the function value of the best feasible point found so far. Thus all techniques for solving constraint satisfaction problems have immediate impact on global optimization. This widens the scope of the possible applications of the methods presented.

Outline. The chapter is organized as follows. In Sections 5.2–5.5 rigorous techniques for enclosing the solution set of linear systems of inequalities are discussed while Sections 5.6–5.7 are about creating linear relaxations for quadratic constraint satisfaction problems. In detail, Section 5.2 considers finding the interval hull of a bounded polyhedron by means of solving a single linear optimization problem. In Section 5.3 the concept of a Gauss-Jordan preconditioner is introduced. In Section 5.4 it is shown how the preconditioner can be used to find cheap bounds for a linear system of inequalities. In Section 5.5 another much costlier approach is given. Here for the most promising directions two linear programs are solved approximately and the approximate solutions are verified. Section 5.6 gives step-by-step instructions how linear relaxations for multivariate quadratic expressions can be generated, and how the method of Lebbah et al. [60] and Kolev [57] can be unified, even considering new relaxation techniques for bilinear terms. In Section 5.7 linear relaxations and filtering for constraint satisfaction problems are considered, by discussing how the linear methods can be combined in order to effectively reduce the search space of a quadratic constraint satisfaction problem. The integration of the methods in GloptLab (see Chapter 2) environment is explained in detail. In Section 5.8 some demonstrative example is given while Section 5.9 presents some test results and comparison of different relaxation techniques.

5.2 Bounding a polyhedron

Geometrically the linear system (1.8) defines a polyhedron. If the polyhedron is bounded and nonempty, the method presented in this section finds a finite enclosure of the polyhedron, by solving a single linear program. By using this method only *large bounds* are reduced; for a fixed, large constant μ (we set $\mu = 10^6$ in our implementation) and a given interval $\mathbf{a}$ the lower bound $\underline{a}$ is large iff $\underline{a} \leq -\mu$ and the upper bound $\overline{a}$ is large iff $\overline{a} \geq \mu$. Choosing μ too small may result in improvement of small bounds but it is not recommended since the gain is not significant enough compared to other methods presented in this chapter. In oder to avoid numerical problems, in this section we also assume that we have found a suitable scaling vector $\omega \in \mathbb{R}^m$ for the constraints and a suitable scaling vector $\rho \in \mathbb{R}^n$ for the variables (for finding scaling vectors we refer to Domes & Neumaier [29]).

5.2.1 Completing one-sided bound constraints

Let

$$Ex \in \mathbf{b},\ x \in \mathbf{x},$$

be a linear system as given in (1.8). We partition the components of the box $\mathbf{b}$ in only lower bounded (B_-), only upper bounded (B_+) and bounded (B_f) ones. We also partition the components of the box $\mathbf{x}$ in unbounded (X_∞), only lower bounded (X_-), only upper bounded (X_+) and bounded (X_f) ones. According to this for $i \in \{1,\dots,m\}$ and $j \in \{1,\dots,n\}$, we define the index sets

$$\begin{aligned}
B_- &:= \{i \mid \underline{b}_i > -\omega_i\mu,\ \overline{b}_i \geq \omega_i\mu\}, & X_+ &:= \{j \mid \underline{x}_j \leq -\rho_j\mu,\ \overline{x}_j < \rho_j\mu\},\\
B_+ &:= \{i \mid \underline{b}_i \leq -\omega_i\mu,\ \overline{b}_i < \omega_i\mu\}, & X_f &:= \{j \mid \underline{x}_j > -\rho_j\mu,\ \overline{x}_j < \rho_j\mu\},\\
B_f &:= \{i \mid \underline{b}_i > -\omega_i\mu,\ \overline{b}_i < \omega_i\mu\}, & X_\pm &:= X_+ \cup X_-,\\
X_\infty &:= \{j \mid \underline{x}_j \leq -\rho_j\mu,\ \overline{x}_j \geq \rho_j\mu\}, & X_b &:= X_\pm \cup X_f,\\
X_- &:= \{j \mid \underline{x}_j > -\rho_j\mu,\ \overline{x}_j \geq \rho_j\mu\}, & X_u &:= X_\pm \cup X_\infty.
\end{aligned} \tag{5.1}$$

Multiplying (1.8) by a vector $y \in \mathbb{R}^m$ (choosen later) leads to the enclosure

$$y^T Ex \in y^T \mathbf{b}.$$

Bringing the terms containing the variables with index in X_f and X_∞ to the right hand side, substituting their bounds and evaluating the results by using interval arithmetic, leads to

$$(y^T E_{:X_\pm}) x_{X_\pm} \in \mathbf{d} := y^T \mathbf{b} - (y^T E_{:X_f}) \mathbf{x}_{X_f} - (y^T E_{:X_\infty}) \mathbf{x}_{X_\infty}. \tag{5.2}$$

Therefore if

$$(E^T y)_{X_\infty} = 0. \tag{5.3}$$

then by (5.2) follows that

$$(y^T E_{:X_\pm}) x_{X_\pm} \geq \underline{d} = \inf(y_{B_-}^T \mathbf{b}_{B_-}) + \inf(y_{B_+}^T \mathbf{b}_{B_+}) + \inf(y_{B_f}^T \mathbf{b}_{B_f}) - \sup(y^T E_{:X_f} \mathbf{x}_{X_f}). \tag{5.4}$$

The following proposition shows which conditions must hold such that (5.4) yields finite bounds on the half–bounded variables:

Proposition 5.1. *If we can find an y such that the conditions*

$$\begin{aligned} y_i &> 0 && \text{if } i \in B_-, \\ y_i &< 0 && \text{if } i \in B_+, \\ (E^T y)_j &= 0 && \text{if } j \in X_\infty, \\ (E^T y)_j &< 0 && \text{if } j \in X_-, \\ (E^T y)_j &> 0 && \text{if } j \in X_+, \end{aligned} \tag{5.5}$$

hold, then for each $k \in X_-$

$$x_k \leq c_k := (\underline{d} - y^T E_{:X_-^k} \underline{x}_{X_-^k} - y^T E_{:X_+} \overline{x}_{X_+}) / (y^T E_{:,k}) \tag{5.6}$$

is satisfied, and for each $k \in X_+$

$$x_k \geq c_k := (\underline{d} - y^T E_{:X_-} \underline{x}_{X_-} - y^T E_{:X_+^k} \overline{x}_{X_+^k}) / (y^T E_{:,k}) \tag{5.7}$$

holds. The bounds c_k are finite.

Proof. Since $y_i > 0$ for all $i \in B_-$ and $y_i < 0$ for all $i \in B_+$ by definition of B_- and B_+ the terms $y_i^T \mathbf{b}_i$ have finite lower bounds for all $i \in B_\pm$. Since $(E^T y)_{X_\infty} = 0$ and $\mathbf{x}_{X_f}$ is bounded by definition the inequality (5.4) holds and $\underline{d}$ is finite. By definition the bounds $\underline{x}_{X_-}$ are finite and by (5.5) the terms $(E^T y)_j = y^T E_{:j} < 0$ for all $j \in X_-$, therefore we have finite approximation

$$y^T E_{:X_-} x_{X_-} \leq y^T E_{:X_-} \underline{x}_{X_-}. \tag{5.8}$$

Similarly, the bounds $\overline{x}_{X_+}$ are finite and $(E^T y)_j = y^T E_{:,j} > 0$ for all $j \in X_+$, therefore we have finite approximation

$$y^T E_{:X_+} x_{X_+} \leq y^T E_{:X_+} \overline{x}_{X_+}. \tag{5.9}$$

For any $k \in X_-$ and $X_-^k := X_- \setminus \{k\}$ since by (5.5) the inequality $y^T E_{:,k} x_k < 0$ holds, considering (5.4) and (5.9) we have

$$\begin{aligned} & y^T E_{:,k} x_k + y^T E_{:X_-^k} x_{X_-^k} + y^T E_{:X_+} x_{X_+} \geq \underline{d} \\ \Rightarrow \quad & y^T E_{:,k} x_k + y^T E_{:X_-^k} \underline{x}_{X_-^k} + y^T E_{:X_+} \overline{x}_{X_+} \geq \underline{d} \\ \Rightarrow \quad & y^T E_{:,k} x_k \geq \underline{d} - y^T E_{:X_-^k} \underline{x}_{X_-^k} - y^T E_{:X_+} \overline{x}_{X_+}, \end{aligned}$$

which by (5.5) implies (5.6) with a finite bound c_k. Therefore $\mathbf{x}'_k = [\underline{x}_k, c_k]$ is finite. By similar considerations for any $k \in X_+$ and $X_+^k := X_+ \setminus \{k\}$ since $y^T E_{:,k} x_k > 0$ we have (5.7) and $\mathbf{x}'_k = [c_k, \overline{x}_k]$ is finite. □

Now we have the necessary conditions on y which allow to find bounds on $x_{X_\pm}$. We note that (5.6) and (5.7) are automatically obtained by standard constraint propagation on (5.2) resulting in finite bounds for the half–bounded variables and improving the bounds which are already finite. If the polyhedron has a finite hull, the constraint propagation succeeds. The constraint propagation method introduced by Chapter 3 can be used for this task whereby this method is even capable of handling the more general case of quadratic constraints.

To find tight bounds on $x_{X_\pm}$ the entries of y should not be larger than necessary. This is achieved by solving the linear program with the objective

$$\text{minimize} \quad \sum_{i \in B_-} \omega_i y_i - \sum_{i \in B_+} \omega_i y_i, \tag{5.10}$$

where ω is the constraint scaling vector, subject to the constraints given by (5.5). Solving the linear program we either obtain a solution $y \in \mathbb{R}^m$, or the linear program is infeasible. In the latter case and the polyhedron is empty or unbounded:

Proposition 5.2. *Suppose that $\mu = \infty$ in (5.1). If the constraints (5.5) are inconsistent then the polyhedron defined by (1.8) is empty or unbounded.*

Proof. Let x^0 be a point satisfying (1.8). If no such x^0 can be found the polyhedron is empty. If this is not the case then that x^0 satisfies (1.8) is equivalent to

$$\begin{aligned} (Ex^0)_{B_-} &\geq \underline{b}_{B_-}, \quad & x^0_{X_-} &\geq \underline{x}_{X_-}, \\ (Ex^0)_{B_+} &\leq \overline{b}_{B_+}, \quad & x^0_{X_+} &\leq \overline{x}_{X_+}, \\ (Ex^0)_{B_f} &\in \mathbf{b}_{B_f}, \quad & x^0_{X_f} &\in \mathbf{x}_{X_f}. \end{aligned}$$

Therefore if a $z \in \mathbb{R}^n$ with $z \neq 0$ satisfies

$$\begin{aligned} (Ez)_{B_-} &\geq 0, \quad z_{X_-} \geq 0, \\ (Ez)_{B_+} &\leq 0, \quad z_{X_+} \leq 0, \\ (Ez)_{B_f} &= 0, \quad z_{X_f} = 0, \end{aligned} \tag{5.11}$$

then

$$\begin{aligned} E(x^0 + \lambda z)_{B_-} &= (Ex^0)_{B_-} + \lambda(Ez)_{B_-} \geq \underline{b}_{B_-}, & (x^0 + \lambda z)_{X_-} &= x^0_{X_-} + \lambda z_{X_-} \geq \underline{x}_{X_-}, \\ E(x^0 + \lambda z)_{B_+} &= (Ex^0)_{B_+} + \lambda(Ez)_{B_+} \leq \overline{b}_{B_+}, & (x^0 + \lambda z)_{X_+} &= x^0_{X_+} + \lambda z_{X_+} \leq \overline{x}_{X_+}, \\ E(x^0 + \lambda z)_{B_f} &= (Ex^0)_{B_f} \in \mathbf{b}_{B_f}, & (x^0 + \lambda z)_{X_f} &= x^0_{X_f} \in \mathbf{x}_{X_+}, \end{aligned}$$

and thus all $x \in L := \{x^0 + \lambda z \mid \lambda \geq 0\}$ satisfy (1.8). Since the set L describes a line segment of infinite length and L is contained in the polyhedron defined by (1.8) the polyhedron must be unbounded.

For any $y \in \mathbb{R}^m$ satisfying (5.5) and $z \in \mathbb{R}^n$, $z \neq 0$ satisfying (5.11)

$$\begin{aligned} 0 &\leq \sum_{i \in B+} y_i(Ez)_i + \sum_{i \in B-} y_i(Ez)_i + \sum_{i \in B_f} y_i(Ez)_i \\ &= \sum_{j \in X\infty} (E^T y)_j z_j + \sum_{j \in X+} (E^T y)_j z_j + \sum_{j \in X-} (E^T y)_j z_j + \sum_{j \in Xf} (E^T y)_j z_j < 0. \end{aligned} \tag{5.12}$$

Therefore (5.5) and (5.11) cannot be solved simultaneously. The Motzkin's transposition theorem (see Motzkin [69]) implies that exactly one of (5.5) and (5.11) is satisfied. Therefore if the constraints (5.5) are inconsistent then (5.11) holds and the polyhedron defined by (1.8) is unbounded. □

In reality we only have an approximate solution $\tilde{y}$ of (5.10) which usually does not need to satisfy (5.3). Therefore using a matrix C (chosen in the next subsection) and a vector z (chosen below) we construct the corrected solution

$$y = \tilde{y} - C^T z, \tag{5.13}$$

such that (5.3) it satisfied. If we substitute (5.13) into (5.3) we see that y satisfies (5.3) if $\tilde{y}E_{:X_\infty} - z^T C E_{:X_\infty} = 0$. Thus we choose z such that

$$(CE_{:X_\infty})^T z = E^T_{:X_\infty} \tilde{y},$$

holds and therefore

$$y = \tilde{y} - C^T((CE_{:X_\infty})^{-T}(E_{:X_\infty}\tilde{y})) \tag{5.14}$$

satisfies (5.3). In floating point arithmetic we have to take the rounding errors into account therefore we evaluate (5.14) using interval arithmetic and obtain a box $\mathbf{y}$ such that for an $y \in \mathbf{y}$ equality (5.3) is satisfied.

5.2.2 Bounding free variables

From this point on we assume that all one sided unbounded constraints are bounded by the method presented in the previous subsection and thus the set $X_\pm$ is empty. Therefore we use (5.2) with $\mathbf{y}$ instead of y and choose C such that we find finite bounds on the free variables x_{X_∞}:

Proposition 5.3. *Let C be a preconditioner for $E_{:X_\infty}$ such that $CE_{:X_\infty} \approx I$. Suppose y satisfies*

$$\begin{aligned} y_i &> 0 && \text{if } i \in B_-, \\ y_i &< 0 && \text{if } i \in B_+, \\ (E^T y)_j &= 0 && \text{if } j \in X_\infty. \end{aligned} \tag{5.15}$$

Let $u^+, u^- \in \mathbb{R}^m$ be vectors with

$$u_j^+ \le \min\{-C_{ij}/y_j \mid i \in X_\infty\}, \quad u_j^- \ge \max\{-C_{ij}/y_j \mid i \in X_\infty\}, \tag{5.16}$$

and

$$C^+ := C + u^+ y^T, \quad C^- := C + u^- y^T, \tag{5.17}$$

then

$$CE_{X_\infty:} x_{X_\infty} \in \mathbf{z} \tag{5.18}$$

for a bounded box

$$\mathbf{z} := [\inf(C^-\mathbf{b} - (C^- E_{:X_b})\mathbf{x}_{X_b}), \ \sup(C^+\mathbf{b} - (C^+ E_{:X_b})\mathbf{x}_{X_b})]. \tag{5.19}$$

Proof. By (5.17) and (5.2), the equation

$$CE_{:X_\infty} x_{X_\infty} = (C^\pm - u^\pm y^T) E_{:X_\infty} x_{X_\infty} = C^\pm E_{:X_\infty} x_{X_\infty} - u^\pm y^T E_{:X_\infty} x_{X_\infty} = C^\pm E_{:X_\infty} x_{X_\infty}$$

holds. On the other hand, (1.8) implies

$$E_{:X_\infty} x_{X_\infty} + E_{:X_b} x_{X_b} \in \mathbf{b},$$

so that

$$\begin{aligned} CE_{:X_\infty} x_{X_\infty} &= C^- E_{:X_\infty} x_{X_\infty} \ge \inf(C^-(\mathbf{b} - E_{:X_b}\mathbf{x}_{X_b})), \\ CE_{:X_\infty} x_{X_\infty} &= C^+ E_{:X_\infty} x_{X_\infty} \le \sup(C^+(\mathbf{b} - E_{:X_b}\mathbf{x}_{X_b})), \end{aligned}$$

proving (5.18).

Since $\mathbf{x}_{X_b}$ is bounded, $E_{:X_b}\mathbf{x}_{X_b}$ is also bounded, we have to show that $\sup(C^+\mathbf{b})_j < \infty$ and $\inf(C^-\mathbf{b})_j > -\infty$ for all $j \in \{1,\ldots,n\}$. By (5.16) we have

$$u_j^+ \leq (-C_{ij}/y_j) \text{ for all } i \in X_\infty. \tag{5.20}$$

If $y_i < 0$ then $\overline{b}_i$ is finite after the construction of y. Using this together with (5.17) implies that $u_j^+ y_i \geq -C_{ij}$ and with (5.20) results in

$$C_{ij}^+ = C_{ij} + u_j^+ y_i \geq 0.$$

Therefore

$$\sup(C_{ij}^+ \mathbf{b}_i) = C_{ij}^+ \overline{b}_i < \mu. \tag{5.21}$$

On the other hand, if $y_i > 0$ then $\underline{b}_i > -\mu$ implies that $C_{ij}^+ = C_{ij} + u_j^+ y_i \leq 0$, again ending up in

$$\sup(C_{ij}^+ \mathbf{b}_i) = C_{ij}^+ \underline{b}_i < \mu. \tag{5.22}$$

Since

$$\sup(C^+\mathbf{b})_j = \sum_{i=1}^{m} \sup(C_{ij}^+ \mathbf{b}_i),$$

for all $j \in \{1,\ldots,n\}$, by inequalities (5.21) and (5.22) we obtain

$$\sup(C^+\mathbf{b})_j < \mu.$$

The proof for the lower bounds is similar, with (5.16) and (5.17) implying that

$$\inf(C_{ij}^- \mathbf{b}_k) = C_{ij}^+ \underline{b}_i > -\mu \tag{5.23}$$

if $y_i > 0$ or

$$\inf(C_{ij}^- \mathbf{b}_k) = C_{ij}^+ \overline{b}_i > -\mu \tag{5.24}$$

if $y_i < 0$, proving that $\inf(C^-\mathbf{b})_j$ is finite for all j. □

By the above proposition

$$x_{X_\infty} \in (CE_{:X_\infty})^{-1}\mathbf{z} \tag{5.25}$$

gives finite bounds on the free variables x_{X_∞}.

5.2.3 Bounding a polyhedron

Summarizing the results of both subsections we are ready to give the following algorithm for bounding a polyhedron:

Algorithm 5.1 (Bounding a polyhedron)**.**
Purpose: Obtain rigorous finite bounds $\mathbf{x}$ *on the variables and improve the bounds* $\mathbf{b}$ *of the linear program* (1.8)*.*

1. *In* (5.10) *we replace the sharp inequalities by non-sharp ones:* $y_i > 0$ *and* $y_i < 0$ *is replaced by* $y_i \geq \omega^{-1}$ *and* $y_i \leq -\omega^{-1}$ *respectively, where* $\omega \in \mathbb{R}^m$ *is the scaling vector for the constraints. Similarly* $(E^T y)_j > 0$ *and* $(E^T y)_j < 0$ *is replaced by* $(E^T y)_j \geq \rho^{-1}$ *and* $(E^T y)_j \leq -\rho^{-1}$ *respectively, where* $\rho \in \mathbb{R}^n$ *the scaling vector for the variables.*

2. *We solve the linear program* (5.10) *by using an approximate linear solver:*

   (a) *If the linear program is feasible, we obtain the approximate solution* $\tilde{y}$ *and compute* $\mathbf{y}$ *according to* (5.14) *using interval arithmetic.*

   (b) *If the linear program is infeasible, the polyhedron is empty or unbounded. The algorithm ends.*

3. *Using constraint propagation on* (5.2) *we obtain finite bounds* $\mathbf{x}'_{X_\pm}$ *on the half-bounded variables* $x_{X_\pm}$*.*

4. *We compute* u^+*,* u^-*,* C^+*,* C^- *and* $\mathbf{z}$ *as defined in Proposition 5.3. Since* x_{X_b} *is already bounded the proposition holds and evaluating* (5.25) *yields finite bounds* $\mathbf{x}'_{X_\infty}$ *on* x_{X_∞}*.*

5. *Substituting the new components* $\mathbf{x}'_{X_\pm}$ *and* $\mathbf{x}'_{X_\infty}$ *for the corresponding components of the bound constraints in* (1.8) *and applying constraint propagation to* (1.8) *results in the new bounds* $\mathbf{x}$ *on the variables.*

6. *We compute the new bounds* $\mathbf{b}' := \mathbf{b} \cap E\mathbf{x}$ *for the constraints.*

5.3 Gauss-Jordan preconditioning

In this section we discuss an extension of Gauss-Jordan elimination, used for preconditioning interval linear systems of equations. We first discuss the original method then modify it to suit our applications.

Discussion of the original method. The Gauss-Jordan inversion is similar to Gaussian elimination but computes the inverse of a matrix. The iterative algorithm starts with an $m \times n$ matrix B (with $n \geq m$) and transforms the leading $m \times m$ sub-matrix of B to an identity matrix. The transformation is done by permuting rows and columns, multiplying whole rows with constants, and subtracting multiplies of a row from other rows. Formally, the Gauss-Jordan elimination algorithm finds an $m \times (n-m)$ matrix L, an $m \times m$ transformation matrix G and an $n \times n$ permutation matrix P such that

$$GBP = [I_m, L]. \tag{5.26}$$

In practice only the matrices P and L are computed explicitly.

Algorithm 5.2 (Gauss-Jordan elimination for $m \times n$ matrices with pivot search)**.**

1. *Given is the $m \times n$ matrix B. The permutation matrix P is initially set to the $n \times n$ identity matrix matrix I_n.*
2. *For $k = 1 \dots m$ do:*
   (a) *Find the (pivot) element p_k; the entry in $B_{k:m,k:n}$ having the maximum absolute value.*
   (b) *If $|p_k| \ll 1$, B is numerically singular, terminate the algorithm and return an error message.*
   (c) *Shift the pivot to B_{kk} by exchanging the rows and columns of B; the k-th row $B_{k:}$ is exchanged with the row of the pivot and the k-th column $B_{:k}$ is exchanged with the column of the pivot.*
   (d) *Exchange the same columns in the permutation matrix P as in B.*
   (e) *Divide all nonzero entries in the k-th column of B by the pivot element p_k;*

$$\lambda_i := \begin{cases} B_{ik}/p_k & \text{if } B_{ik} \neq 0, \\ 0 & \text{otherwise,} \end{cases} \qquad p_k = B_{kk}. \tag{5.27}$$

   (f) *Overwrite the rows $B_{i:}$ of B with*

$$B'_{i:} := \begin{cases} \lambda_k B_{k:} & \text{if } i = k, \\ B_{i:} - \lambda_i B_{k:} & \text{otherwise,} \end{cases}$$

   making the kth column of B' to the kth column of the identity matrix I_m.
3. *Since the matrix B now has the form $B = [I_m, L]$, return the matrix L and the column permutation matrix P.*

Example 5.1. *We apply the above algorithm for*

$$B = \begin{pmatrix} 3 & 3 & 1 & 0 \\ 2 & 6 & 0 & 1 \end{pmatrix} \text{ and } P = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

- *($k = 1$) The maximal element of B is $B_{22} = 6$, which is chosen as the first pivot p_1. In B we exchange the first row with the second one, and in B and P we exchange the first column with the second one, which results in*

$$B = \begin{pmatrix} 6 & 2 & 0 & 1 \\ 3 & 3 & 1 & 0 \end{pmatrix} \text{ and } P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

  Then the multiplier $\lambda_2 = 3/6 = 1/2$ is computed. We divide the first row with the pivot, subtract $\lambda_2 B_{1:} = (3 \;\; 1 \;\; 0 \;\; 1/2)$ from the second one, and get

$$B = \begin{pmatrix} 1 & \frac{1}{3} & 0 & \frac{1}{6} \\ 0 & 2 & 1 & -\frac{1}{2} \end{pmatrix}.$$

- *($k = 2$) The pivot p_2 is $B_{22} = 2$ which is the maximum element of the submatrix $B_{2,2:4}$. In this case the pivot is at the correct position, therefore no exchange of the rows or columns of B or P is needed. Since $\lambda_1 = (1/3)/2 = 1/6$ from the first row we subtract $\lambda_1 B_{2:} = (0 \;\; 1/3 \;\; 1/6 \;\; -1/12)$, then divide the second one with the pivot, and get*

$$B = \begin{bmatrix} I_2 & L \end{bmatrix}, \quad L := \begin{pmatrix} -\frac{1}{6} & \frac{1}{4} \\ \frac{1}{2} & -\frac{1}{4} \end{pmatrix} \text{ and } P = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{5.28}$$

The algorithm is finished, the matrices L and P from (5.28) are returned.

The aim of the original Gauss-Jordan inversion is to find the inverse of the $m \times m$ matrix A. The first version of the Gauss-Jordan inversion was numerically unstable since it did not use pivoting. If we set $B := [A, I_m]$ in the Gauss-Jordan elimination Algorithm 5.2 we get the Gauss-Jordan inversion with pivoting, by (5.26) we have

$$G[A, I_m]P = [I_m, L], \tag{5.29}$$

and after m iterations the Algorithm 5.2 results in the matrix $[I_m, L]$ and the column permutation P. Note that for this selection of B even if A is singular B is regular by construction and thus Algorithm 5.2 never returns an error message.

Proposition 5.4. *If the column permutation matrix P returned by Algorithm 5.2 consists only of permutations of the first m columns, then the matrix $C := \hat{P}L$ with $\hat{P} := P_{1:m,1:m}$ is the inverse of A.*

Proof. Since P consist only of permutations of the first m columns it must have the form

$$P = \begin{pmatrix} \hat{P} & 0 \\ 0 & I_m \end{pmatrix}. \tag{5.30}$$

Then by (5.29)

$$G[A, I_m]P = [I_m, L] \Rightarrow GA\hat{P} = I_m,\ GI_m = L \Rightarrow LA\hat{P} = I_m \Rightarrow (\hat{P}L)A = I_m. \tag{5.31}$$

Proving that A is regular and $\hat{P}L$ is the inverse of A. □

Example 5.2. *In Example 5.1, we had*

$$B = [A, I_2] \text{ with } A = \begin{pmatrix} 3 & 3 \\ 2 & 6 \end{pmatrix}.$$

By (5.28), P has the form of (5.30) with $\hat{P} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, thus the matrix

$$C = \hat{P}L = B = \begin{pmatrix} \frac{1}{2} & -\frac{1}{4} \\ -\frac{1}{6} & \frac{1}{4} \end{pmatrix},$$

is the inverse of A.

Scaling: Note that if the matrix A is regular, but not scaled correctly, entries from the wrong part of B may be chosen as pivot elements. In this case even though A is regular, (5.30) does not hold, and the inverse of A cannot be found by the algorithm. This would happen if in the above example we would divide the entries of A by 10. Since this would only scale the matrix A, it would be still invertible. In this case in the first step of the algorithm $B_{31} = 1$ would be chosen as the pivot therefore the prerequisites of Proposition 5.4 would not be met.

To solve this problem, Algorithm 5.2 can be improved by applying *suitable scaling* to A and I_m. We choose a diagonal row scaling matrix $U \in \mathbb{R}^{m\times m}$, a diagonal column scaling

matrix $V \in \mathbb{R}^{m\times m}$ and an additional scaling constant δ and set

$$B := [UAV, \delta I_m].$$

in Algorithm 5.2 and obtain

$$G[UAV, \delta I_m]P = [I_m, L]. \tag{5.32}$$

Theorem 5.5. *If the column permutation matrix P returned by Algorithm 5.2 consists only of permutations of the first m columns, then the matrix $C := \delta^{-1}V\hat{P}LU$ is the inverse of A.*

Proof. Since P consist only permutations of the first n columns we have

$$P = \begin{pmatrix} \hat{P} & 0 \\ 0 & I_m \end{pmatrix}, \tag{5.33}$$

then by (5.32)

$$GUAV\hat{P} = I_m,\ G\delta I_m = L \Rightarrow \delta^{-1}LUAV\hat{P} = I_m \Rightarrow (\delta^{-1}V\hat{P}LU)A = I_m, \tag{5.34}$$

holds, proving the assumption. □

We suggest the following choice of the scaling matrices U and V and the scaling constant δ: By the scaling method presented in Domes & Neumaier [29] we find matrices U and V such that the entries of UAV are between zero and one but not too close to zero. The second part of $B = [UAV, \delta I_m]$ consists of an $m \times m$ identity matrix, scaled with the constant δ. Setting δ as a very small positive number (e.g., $\delta := \sqrt{\varepsilon}$) prevents that – even for well conditioned matrices – some elements of the second part of B are chosen as pivots.

Extension of the Gauss-Jordan inversion. If the matrix A is not square or regular, there is no (two-sided) inverse, but preconditioner for A can be found, such that for a suitable chosen index set J the equality $CA_{:J} = I$ holds.

Theorem 5.6. *Let $A, L \in \mathbb{R}^{m\times n}$, $U, G \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$, $P \in \mathbb{R}^{(n+m)\times(n+m)}$ and $\delta > 0$.*

(1) Suppose that

$$G[UAV, \delta I_m] = [I_m, L]P, \tag{5.35}$$

then

$$G = \delta^{-1}(P_{MM} + LP_{NM}), \tag{5.36}$$

where $M = \{1, \ldots, m\}$ *and* $N = \{m+1, \ldots, m+n\}$.

(2) If V *is an invertible diagonal matrix,* P *is a permutation matrix and* $R \subseteq \{1, \ldots, n\}$ *is an index set of size* $r \leq m$ *such that*

$$P_{RM} P_{MR} = I_r \tag{5.37}$$

holds, then

$$C := V_{RR} P_{RM} G U \tag{5.38}$$

satisfies

$$C A_{:R} = I_r. \tag{5.39}$$

Proof. We write P in block form as

$$P = \begin{pmatrix} P_{MN} & P_{MM} \\ P_{NN} & P_{NM} \end{pmatrix} \tag{5.40}$$

with $P_{MN} \in \mathbb{R}^{m \times n}$, $P_{NN} \in \mathbb{R}^{n \times n}$, $P_{MM} \in \mathbb{R}^{m \times m}$ and $P_{NM} \in \mathbb{R}^{n \times m}$. Then we have

$$[I_m, L] P = [I_m P_{MN} + L P_{NN}, I_m P_{MM} + L P_{NM}]. \tag{5.41}$$

From (5.41) and (5.35) the identities

$$GUAV = P_{MN} + L P_{NN} \text{ and } G = \delta^{-1}(P_{MM} + L P_{NM})$$

follow. The second equality proves assumption (1). To prove (2) we multiply the second equality with the matrix $W := V_{R:} P_{NM}$ from the left and with a vector x from the right side and obtain

$$WGUAVx = W P_{MN} x + W L P_{NN} x.$$

Without loss of generality, we assume that $R = \{1, \ldots, r\}$. Choosing

$$x_i := \begin{cases} z_i / V_{ii} & \text{for } i \in R, \\ 0 & \text{otherwise,} \end{cases}$$

we find $x_R = V_{RR}^{-1} z_R$ and $AVx = A_{:R} z_R$. Since P is a permutation matrix, from (5.37) follows that all columns of P_{MR} contain a one; therefore the columns P_{NR} have to be zero columns. Since V^{-1} is diagonal, the matrix V_{RR}^{-1} only contains nonzero elements in the jth rows when $j \in R$. We summarize and obtain

$$\begin{aligned} (P_{NR})_{ij} &= 0 \quad \text{if} \quad j \in R \\ (V_{RR}^{-1})_{jk} &= 0 \quad \text{if} \quad j \notin R. \end{aligned} \tag{5.42}$$

Then by (5.42) follows that

$$(P_{NR}V_{RR}^{-1})_{ik} = \sum_{j=1}^{n}(P_{NR})_{ij}(V_{RR}^{-1})_{jk} = 0, \quad \text{for all } i, j.$$

Therefore $P_{NR}V_{RR}^{-1}z_R = 0$ and

$$V_{RR}P_{RM}GUAz_R = V_{RR}P_{RM}P_{MR}V_{RR}^{-1}z_R.$$

By (5.37) we get

$$V_{RR}P_{RM}GUAz_R = z_R.$$

implying (5.38) and (5.39). □

Using the results of the above proposition we generalize Algorithm 5.2:

Algorithm 5.3 (Gauss-Jordan preconditioner)**.**

1. *Given is the matrix $A \in \mathbb{R}^{m\times n}$, the diagonal row scaling matrix $U \in \mathbb{R}^{m\times m}$ and the diagonal column scaling matrix $V \in \mathbb{R}^{n\times n}$.*
2. *We set $u = n + m$, $K = (1, \ldots, u)$ and $B = [UAV, \delta I_m] \in \mathbb{R}^{m\times u}$.*
3. *For $k = 1 \ldots m$ do:*
   (a) *Find the pivot $p_k := B_{ij}$ having the maximum absolute value from the submatrix $B_{k:m,k:u}$.*
   (b) *Exchange the kth row $B_{k,:}$ of B with the row $B_{i,:}$ of the pivot and the kth column $B_{:,k}$ of B with the column $B_{:,j}$ of the pivot.*
   (c) *Exchange K_k with K_j in the index list K.*
   (d) *Compute each λ_i as given in (5.27).*
   (e) *For each $i \neq k$ overwrite the row $B_{i:}$ with $B_{i:} - \lambda_i B_{k:}$.*
   (f) *Overwrite $B_{k:}$ with $B_{k:}/p_k$.*
4. *The row permutation matrix can be set by using the found index set K:*

$$P = \begin{pmatrix} P_{MN} & P_{MM} \\ P_{NN} & P_{NM} \end{pmatrix} = (I_{n+m})_{:K}.$$

5. *The matrix $G = \delta^{-1}(P_{MM} + LP_{NM})$ is computed.*
6. *Generate the index set R:*

$$R = \{j \in R_{1:m} \mid K_j \leq n\}$$

7. *Since for* $\hat{P}_{RM} := P_{MR}^T$ *condition* (5.37) *in Theorem* 5.6 *holds, the preconditioner,*

$$C = V_{RR} P_{RN}^T G U$$

with $CA_{:R} = I_r$ *is obtained.*

8. *We have found an index set* R *and a matrix* $C \in \mathbb{R}^{r \times m}$ *such that* (5.35) *holds. Return the matrix* C *and the index set* R*.*

The preconditioner found by Algorithm 5.3 can be used for solving under- or overdetermined linear equation systems. If the matrix A is square and has full rank, Algorithm 5.3 returns the inverse of A.

Example 5.3. *Let*

$$A = \begin{pmatrix} \frac{1}{2} & \frac{1}{6} & \frac{1}{6} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{pmatrix}, \; U = \begin{pmatrix} 6 & 0 \\ 0 & 8 \end{pmatrix}, \; V = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 3 \end{pmatrix}, \quad \text{and } \delta = 1.$$

Then the matrix

$$B = [UAV, \delta I_m] = \begin{pmatrix} 3 & 1 & 3 & 1 & 0 \\ 2 & 4 & 6 & 0 & 1 \end{pmatrix},$$

is similar to the matrix in Example 5.2*; the same pivots will be chosen. After two iterations, we get*

$$B = \begin{pmatrix} 1 & 0 & \frac{1}{2} & -\frac{1}{6} & \frac{1}{4} \\ 0 & 1 & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{4} \end{pmatrix}, \quad K = (3, 1). \tag{5.43}$$

Then we have $R = \{1, 3\}$*,*

$$G = \delta^{-1}(P_{MM} + L P_{NM}) = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} + \begin{pmatrix} \frac{1}{2} & -\frac{1}{6} & \frac{1}{4} \\ -\frac{1}{2} & \frac{1}{2} & -\frac{1}{4} \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} -\frac{1}{6} & \frac{1}{4} \\ \frac{1}{2} & -\frac{1}{4} \end{pmatrix},$$

$$C = V_{RR} P_{RM} G U = \begin{pmatrix} 1 & 0 \\ 0 & 3 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} -\frac{1}{6} & \frac{1}{4} \\ \frac{1}{2} & -\frac{1}{4} \end{pmatrix} \begin{pmatrix} 6 & 0 \\ 0 & 8 \end{pmatrix} = \begin{pmatrix} 3 & -2 \\ -3 & 6 \end{pmatrix},$$

and find that $CA_{:R} = I_r$ *holds.*

The above considerations about a suitable scaling (in the sense of making the algorithm choose all linear independent columns as pivot columns by correctly choosing U, V and δ) applies again. If P permutes some of the last m columns, then either A has non-maximal numerical rank or the scaling was not chosen suitably.

Lemma 5.7. *If the matrix R returned by Algorithm 5.3 was computed by using exact arithmetic and suitable scaling then $r := |R|$ is the rank of A .*

Proof. Without the loss of generality we assume that in the kth iteration step the first part $B_{k,1:n}$ of the pivot row and the first part $B_{i,1:n}$ of another row ($k < i$) are linearly dependent. Therefore $B_{i,1:n} = cB_{k,1:n}$ holds for some constant c. Since $\lambda_i = B_{ik}/B_{kk} = c$ the entries $B_{i,1:n}$ will be overwritten with $B_{i,1:n} - cB_{k,1:n} = 0$. From this point on the first n entries of this row only contain zeros, and sooner or later a pivot has to be selected from the lower right $m \times m$ part of B. Since this happens for each linear dependent row the number of pivots selected from the first part of B gives us the rank of the matrix A. □

Since in our implementation inexact arithmetic is used, due to the rounding errors, we only get the numerical rank, which (if the scaling is suitable) is correct for the most non-degenerate matrices.

5.4 Linear contraction

Based on the Gauss-Jordan method discussed in Section 5.3, we present a simple technique for reducing the bounds of x of the linear system (1.8).

First a Gauss-Jordan preconditioner for the matrix E is computed; we choose suitable scaling matrices U and V and a scaling factor δ then apply Algorithm 5.3 for the matrices E, U, V and the scaling factor δ.

A possible choice for the scaling was given in Section 5.3. For this application a better alternative is to specify the scaling matrices U and V such that the rows of E matching the constraints having tighter bounds are preferred as pivot rows. Similarly, by this scaling the columns of E matching the variables with tighter bounds, are preferred as pivot columns. According to this we set

$$d := \max\{\underline{x}_i,\ \overline{x}_j \mid i = 1 \dots n,\ j = 1 \dots n\}, \quad \mathbf{z} = \mathbf{x} \cap [-d, d]^n$$

and for the scaling matrices

$$U = \operatorname{diag}(u), \quad V = \operatorname{diag}(v) \tag{5.44}$$

with

$$u = ((\overline{b} - \underline{b}) + \delta|\mathbf{b} - E\mathbf{x}|) \quad \text{and} \quad v = (\overline{z} - \underline{z})/(\max\{\overline{z}_i - \underline{z}_i \mid i = 1 \dots n\}).$$

For the scaling constant (as in Section 5.3) we choose $\delta = \sqrt{\varepsilon}$.

The algorithm returns an index list R with $|R| = r$ and a matrix $C \in \mathbb{R}^{r \times m}$ such that $CE_{:R} = I_r$. We set $K := \neg R$, multiply (1.8) with the matrix C and obtain

$$CE_{:R}x_R + CE_{:K}x_K \in C\mathbf{b},\ x_R \in \mathbf{x}_R,\ x_K \in \mathbf{x}_K. \tag{5.45}$$

Since $CE_{:R} = I_r$, if we substitute the bounds for x_K we get

$$x_R \in \hat{\mathbf{b}},\ \hat{\mathbf{b}} := (C\mathbf{b} - CE_{:K}\mathbf{x}_K),\ x_R \in \mathbf{x}_R.$$

and if we cut $x_R \in \mathbf{b}'$ with the original bounds $\mathbf{x}_R$ on the variables x_R we end up in

$$x_R \in \hat{\mathbf{x}},\ \hat{\mathbf{x}} := \hat{\mathbf{b}} \cap \mathbf{x}_R. \tag{5.46}$$

If the matrix E is square and has full rank ($n = m = r$) then we get

$$x \in \hat{\mathbf{x}},\ \hat{\mathbf{x}} := C\mathbf{b} \cap \mathbf{x}.$$

In inexact arithmetic, the computation of the preconditioner C is not rounding error free, and thus only $CE_{:R} \approx I_r$ holds. This modifies (5.46) to

$$Mx_R \in \hat{\mathbf{b}},\ M := CE_{:R},\ x_R \in \mathbf{x}_R.$$

Since the off diagonal entries of M are tiny we have

$$x_R \in \hat{\mathbf{x}} \text{ with } \hat{\mathbf{x}}_i := (M_{ii}^{-1}(\hat{\mathbf{b}}_i - \sum_{j=1,\ j\neq i}^{r} M_{ij}\mathbf{x}_i)) \cap \mathbf{x}_i,\ M := CE_{:R},\ \hat{\mathbf{b}} := (C\mathbf{b} - CE_{:K}\mathbf{x}_K). \tag{5.47}$$

Again, if the matrix E is square and of full rank, then $CE \approx I_m$ and we get the bounds

$$x \in \hat{\mathbf{x}} \text{ with } \hat{\mathbf{x}}_i := \frac{(C\mathbf{b})_i - \sum_{j=1,\ j\neq i}^{r}(CE)_{ij}\mathbf{x}_i}{(CE)_{ii}} \cap \mathbf{x}_i.$$

By using this method we obtain new bounds for the variables x_R.

An alternative to the above method is to use constraint propagation on (5.45). Constraint propagation for quadratic (and linear) systems is discussed in Chapter 3. This alternative costs more computational time but it also yields new bounds on the variables x_K not only on x_R. Both approaches are useful; the decision which alternative is preferable is based on the dimension of the problem.

5.5 Linear bounding

Another simple, efficient, but costly method for improving the bounds on the variables in linear systems is presented in this section.

Let (1.8) be a linear system of n variables and m two-sided inequalities. We choose $k \leq n$ variables, where we expect to achieve the most reduction. Alternatively we could select all n variables. For each variable x_i, $i \in I$ we solve two linear programs (one for each sign in e^i) given by

$$\begin{aligned} \min \quad & f(x) := e^i x \\ \text{s.t.} \quad & Ex \in \mathbf{b},\ x \in \mathbf{x}, \end{aligned} \tag{5.48}$$

with

$$e^i_j = \begin{cases} \pm 1 & \text{if } i = j \\ 0 & \text{otherwise,} \end{cases}.$$

Let $\hat{x}^{i+}$ and $\hat{x}^{i-}$ be approximative solutions of (5.48) for $e^i_i = 1$ and $e^i_i = -1$ respectively. Let y^{i+} and y^{i-} the approximate multipliers of the solutions without the multipliers corresponding to the bound constraints. If we solve all the $2k$ linear programs the multipliers are collected in a $2k \times m$ matrix

$$Y \in \mathbb{R}^{2k \times m},\quad Y_{:,2i-1} = y^{i+},\quad Y_{:,2i} = y^{i-} \quad \text{for all } i = 1, \ldots, k$$

The matrix Y can be used to precondition the linear system (1.8) resulting in a new system of $2k$ inequalities

$$\hat{\mathbf{E}}x \in \hat{\mathbf{b}},\ x \in \mathbf{x},\ \hat{\mathbf{E}} := Y[E, E],\ \hat{\mathbf{b}} := Y\mathbf{b}. \tag{5.49}$$

To ensure mathematical rigor, the interval coefficient matrix $\hat{\mathbf{E}}$ and the box $\hat{\mathbf{b}}$ must be computed by using interval arithmetics. Since each rows $\hat{\mathbf{E}}_{k:}$ of $\hat{\mathbf{E}}$ should only contain one dominant entry $\hat{\mathbf{E}}_{kj}$, all other ones should be approximately zero, (5.49) can be solved easily. This can be done by substituting the bounding intervals $\mathbf{x}_i$ for the variables with nearly zero coefficients into (5.49) and bringing the corresponding terms to the right hand side. This results in a new bound

$$x_j \in \hat{\mathbf{x}}_j,\ \hat{\mathbf{x}}_j := \mathbf{x}_j \cap (\hat{\mathbf{b}} - \sum_{i=1, i \neq j}^{n} \hat{\mathbf{E}}_{ki}\mathbf{x}_i)$$

on the variable x_j arising from all $2k$ inequalities. Alternately, constraint propagation for quadratic (and linear) systems is discussed in Chapter 3 can be used to solve (5.49).

5.6 Linear relaxations for multivariate quadratic expressions

The following sections elaborate techniques for creating linear relaxations of quadratic constraint satisfaction problems.

Let $p(x) : \mathbb{R}^n \to \mathbb{R}$ be a mapping. The function $u(x)$ is called an *underestimator* of $p(x)$ in the box $\mathbf{x}$ if $u(x) \leq p(x)$ holds for all $x \in \mathbf{x}$. Similarly, the function $v(x)$ is called an *overestimator* of $p(x)$ in the box $\mathbf{x}$, if $p(x) \leq v(x)$ holds for all $x \in \mathbf{x}$. If both an underestimator $u(x)$ and an overestimator $v(x)$ is given then

$$p(x) \in [u(x), v(x)] \text{ for all } x \in \mathbf{x}$$

is an *enclosure* of $p(x)$ in the box x.

Theorem 5.8. *Let $p(x), h(x) : \mathbb{R}^n \to \mathbb{R}$ be mappings, let $\mathbf{c}$, $\mathbf{d}$ be intervals and let $x \in \mathbf{x}$. If*

$$p(x) \in \mathbf{c} \quad \Rightarrow \quad h(x) \in \mathbf{d}. \tag{5.50}$$

for all $x \in \mathbf{x}$ then

$$h(x) - p(x) \in [\underline{d} - \underline{c},\ \overline{d} - \overline{c}] \tag{5.51}$$

is satisfied for all $x \in \mathbf{x}$. In this case, the two-sided inequality $h(x) \in \mathbf{d}$ is called a relaxation *of $p(x) \in \mathbf{c}$ in the box $\mathbf{x}$.*

Proof. ($\Rightarrow$) By (5.50), for a real number r (choosen later) the inequality $h(x) \geq p(x) + r$ must hold for all $x \in \mathbf{x}$. Since $p(x) \in \mathbf{c}$, $h(x) \geq p(x) + r \geq \underline{c} + r$. Since $h(x) \in \mathbf{d}$, $h(x) \geq \underline{d}$ must also hold. Choose r as minimal and get $\underline{c} + r = \underline{d}$ ending up in $h(x) \geq p(x) + r = p(x) + \underline{d} - \underline{c}$. This given the lower inequality $\underline{d} - \underline{c} \leq h(x) - p(x)$ of (5.51). The upper inequality $h(x) - p(x) \leq \overline{d} - \overline{c}$ can be obtained in the same way.
($\Leftarrow$) By (5.51) the inequality $\underline{d} - \underline{c} \leq h(x) - p(x)$ holds for all $x \in \mathbf{x}$. Bringing $p(x)$ to the left hand side results in $p(x) + \underline{d} - \underline{c} \leq h(x)$. By (5.61) $\underline{c} \leq p(x)$ and thus $\underline{d} = \underline{c} + \underline{d} - \underline{c} \leq h(x)$. The inequality $h(x) \leq \overline{d}$ can be obtained similarly. Therefore (5.50) holds, proving the assumption. □

In the following subsections give a step-by-step explanation how linear relaxations for multivariate quadratic expressions are generated; in Subsection 5.6.1 linear relaxations for univariate, quadratic expressions are construct, then in Subsection 5.6.2 separable, multivariate, quadratic expressions are handled, finally in Subsection 5.6.3 the most general case of generating linear relaxations for multivariate, not necessary separable, quadratic expressions is discussed.

5.6.1 Linear relaxations for univariate quadratic expressions

Without loss of generality, an arbitrary univariate quadratic expressions, can be written in the form

$$q(x) \in \mathbf{c}, \quad q(x) := ax^2 + bx, \quad x \in \mathbf{x}, \tag{5.52}$$

where a and b are constant and $\mathbf{c}$ and $\mathbf{x}$ are intervals. We assume without the loss of generality that $a > 0$ since for $a = 0$ we already have a linear expression, with no need of relaxing, and for $a < 0$ all the observations below hold with trivial modifications.

For univariate functions Kolev [57] proposes linear relaxations of the form

$$ex \in \mathbf{d} \text{ for } x \in \mathbf{x} \text{ with } q(x) \in \mathbf{c}, \tag{5.53}$$

where e is a constant and $\mathbf{d}$ is an interval (see, Figure 5.1). Kolev states that this relaxation is optimal if $\mathbf{w}$ has minimal width and uses a generalized representation of intervals to compute e and $\mathbf{d}$.

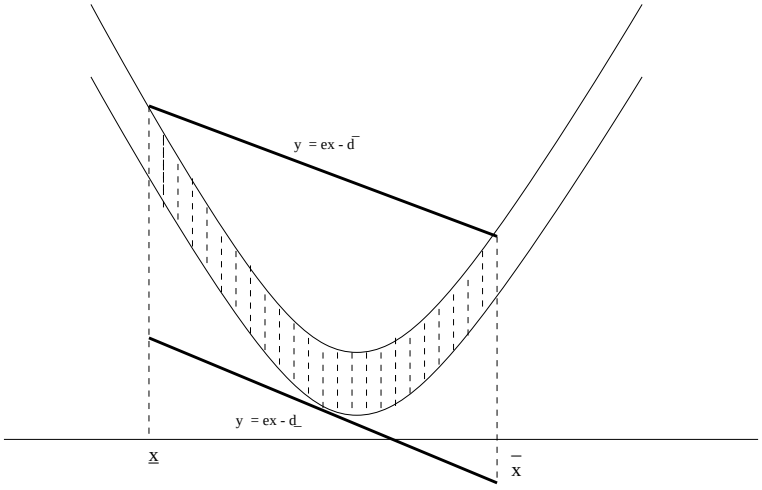

Figure 5.1: Linear relaxations by Kolev

Another approach is the Quad algorithm of Lebbah et al. [60], where linear under- and overestimators are used to generate linear relaxations. Since $a > 0$, we obtain for any $z \in \mathbf{x}$ linear underestimators

$$L_z(x) := l'(z)(x - z) + l(z) \text{ where } l(x) := q(x) - \underline{c},$$

(in [60], the two tangents of $l(x)$ for $z = \underline{x}$ and $z = \overline{x}$ are chosen) and the linear overestimator

$$L(x) := \frac{q(\overline{x}) - q(\underline{x})}{\overline{x} - \underline{x}} x + \frac{u(\underline{x})\overline{x} - u(\overline{x})\underline{x}}{\overline{x} - \underline{x}} \text{ where } u(x) := q(x) - \overline{c},$$

(the secant of $u(x)$ between the points $(\underline{x}, u(\underline{x}))$ and $(\overline{x}, u(\overline{x}))$) (see, Figure 5.2).

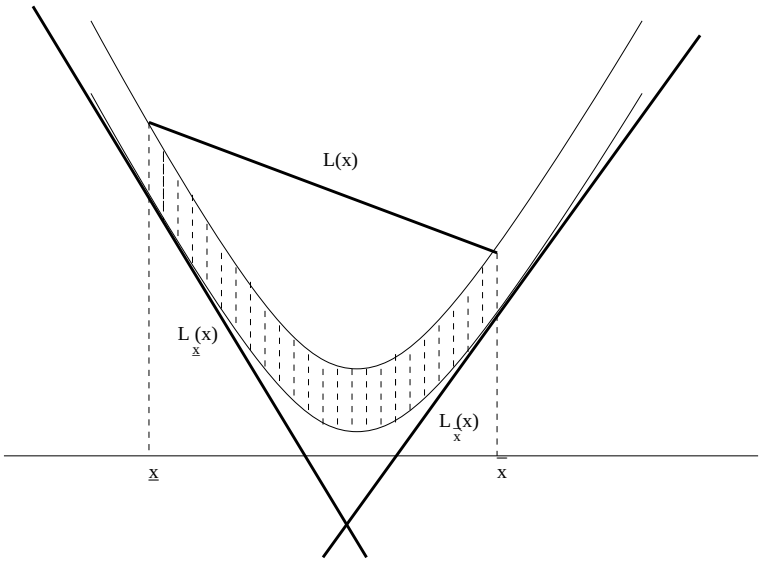

FIGURE 5.2: Linear relaxations by Lebbah & Rueher

Note that while L is the best choice for linear overestimator, the choice and the number of the underestimators L_z are arbitrary. The two underestimators suggested by [60] could be replaced by a single one (e.g., $L_z(\text{mid}(\mathbf{x}))$) or refined by adding more (e.g., $L_z(\text{mid}(\mathbf{x}))$ would be a good choice). The latter can be made adaptive to satisfy a given error bound, and is then called the sandwich method (see TAWARMALANI & SAHINIDIS [97]), it used in the solver BARON.

According to this if Z is a finite set with values in the box $\mathbf{x}$ and $n_z = |Z|$, then the system

$$L_z(x) \geq 0, \quad L(x) \leq 0, \quad z \in Z, \quad x \in \mathbf{x}, \tag{5.54}$$

of $n_z + 1$ linear inequalities is a linear relaxation of (5.52).

To give an uniform representation for the two methods we choose the form (5.53) where e is now a k–dimensional vector $\mathbf{d}$ is a k–dimensional box. The relaxation by Kolev is included in this form for $k = 1$ while the inequalities by Lebbah & Rueher can be embedded by setting

$$e_i = \begin{cases} q'(X_i) & \text{for} \quad i = 1, \ldots, n_z, \\ \frac{q(\overline{x}) - q(\underline{x})}{\overline{x} - \underline{x}} & \text{if} \quad i = n_z + 1, \end{cases} \quad \mathbf{d}_i = \begin{cases} [q'(X_i)X_i - q(X_i) + \underline{c}, \infty] & \text{for} \quad i = 1, \ldots, n_z, \\ [-\infty, \frac{q(\overline{x})\underline{x} - q(\underline{x})\overline{x}}{\overline{x} - \underline{x}} - \overline{c}] & \text{if} \quad i = n_z + 1. \end{cases}$$

5.6.2 Linear relaxations for separable multivariate quadratic expressions

We consider an arbitrary separable multivariate quadratic expression, which we write without loss of generality in the form

$$p(x) \in \mathbf{c}, \quad p(x) := a^T x^2 + b^T x, \quad x \in \mathbf{x}, \tag{5.55}$$

where x^2 is the component-wise square of x, a and b are n–dimensional vectors, $\mathbf{c}$ is an interval and $\mathbf{x}$ is an n–dimensional box. We assume that $a \neq 0$ since otherwise we already would have a linear expression, with no need of relaxing. The results of the univariate case can be directly applied to the multivariate case with slight modifications:

For a multivariate function of n variables, we consider linear relaxations of the form

$$e^T x \in \mathbf{d}, \quad x \in \mathbf{x}, \tag{5.56}$$

where e is an n–dimensional vector and $\mathbf{d}$ is an interval. Since (5.56) is a linear relaxation of (5.55) by (5.51)

$$e^T x - q(x) \in [\underline{d} - \underline{c}, \overline{d} - \overline{c}],$$

holds. For this special case of a separable quadratic expression this simplifies to

$$u^T x - a^T x^2 \in [\underline{d} - \underline{c}, \overline{d} - \overline{c}], \quad u := e + b.$$

If we choose a suitable slope vector e, the exact range $\mathbf{t}$ of the quadratic expression on the left hand side is easy to compute rigorously (see Chapter 3). This results in the equality $\mathbf{t} = [\underline{d} - \underline{c}, \overline{d} - \overline{c}]$ from which the interval $\mathbf{c}$ follows directly. Possible selections of the slope vector could be the derivate $2a \cdot x + b$ of $q(x)$ (where $\cdot$ denotes the componentwise product) in a suitable chosen point $z \in \mathbf{x}$ (midpoint, upper or lower bound, or midpoint of a promising region of $\mathbf{x}$). Another useful selection for the slope vector is the the secant slope $\frac{q(\overline{x}) - q(\underline{x})}{\overline{x} - \underline{x}}$, of between the points $(\underline{x}, q(\underline{x}))$ and $(\overline{x}, q(\overline{x})))$.

To integrate the method of Lebbah & Rueher [60] for the multivariate case, we generate the linear relaxations for each univariate quadratic expression separately. Let

$$Q := \{k \in \{1, \ldots, n\} \mid a_k \neq 0\}, \quad L := \{k \in \{1, \ldots, n\} \mid b_k \neq 0\}, \tag{5.57}$$

with $n_q = |Q|$ and $n_l = |L|$ be the index sets then (5.55) can be written as

$$\begin{aligned} &\textstyle\sum_{k \in Q} y_k + \sum_{k \in L} b_k x_k \in \mathbf{c}, \quad x \in \mathbf{x}, \\ &y_k = a_k x_k^2 \text{ for all } k \in Q. \end{aligned}$$

To generate the linear relaxations of the n_q univariate quadratic expressions $a_k x_k^2$ we apply the results of the previous section; we choose the set Z, compute the n_z+1 linear inequalities

$$e^k x_k \in \mathbf{d}^k, \text{ for } x_k \in \mathbf{x}_k,$$

for the n_q quadratic expressions separately, whereby e^k is now a (n_z+1)–dimensional vector and $\mathbf{d}^k$ is a (n_z+1)–dimensional box. Let Z be a finite set with values in $\mathbf{x}$ and $n_z = |Z|$ then the linear relaxation can be given as a system of $n_y(n_z+1)+1$ inequalities:

$$\sum_{k\in Q} y_k + \sum_{k\in L} b_k x_k \in \mathbf{c}, \quad x \in \mathbf{x}, \quad y_k - e_i^k x_k \in \mathbf{d}_i^k, \text{ for all } k \in Q.$$

$$e_i^k = \begin{cases} 2a_k X_i & \text{for} \quad i = 1,\ldots,n_z, \\ a_k(\overline{x}+\underline{x}) & \text{if} \quad i = n_z+1, \end{cases} \qquad \mathbf{d}_i^k = \begin{cases} [-a_k X_i^2, \infty] & \text{for} \quad i = 1,\ldots,n_z, \\ [-\infty, -a_k \overline{x}_k \underline{x}_k] & \text{if} \quad i = n_z+1. \end{cases}$$

In order to give an uniform representation for the two methods, we propose the general form

$$Ex \in \mathbf{d}, \quad x \in \mathbf{x}, \tag{5.58}$$

with $E \in \mathbb{R}^{h\times n}$ and $\mathbf{d}$ is an h–dimensional box. The relaxation by Kolev is included in this form for $h = 1$ while the inequalities by Lebbah & Rueher can be embedded by setting the $h = n_y(n_z+1)+1$ components and increasing the number of variables to $n + n_q$.

5.6.3 Linear relaxations for multivariate quadratic expressions

We consider an arbitrary multivariate quadratic expressions

$$p(x) \in \mathbf{c}, \quad p(x) := \sum_{k\in Q} a_k x_k^2 + \sum_{(j,k)\in B} b_{jk} x_j x_k + \sum_{k\in L} b_k x_k, \quad x \in \mathbf{x}, \tag{5.59}$$

where the $a_k x_k^2$ are the quadratic, the $b_{jk} x_j x_k$ are the bilinear, the $b_k x_k$ are the linear terms and $\mathbf{c}$ is an interval. The sets Q and L are are from (5.57), while

$$B := \{(j,k) \in \{1,\ldots,n\} \times \{1,\ldots,n\} \mid b_{jk} \neq 0\}, \quad n_b = |B|,$$

and we assume that B is non-empty.

We discuss two different methods to deal with the bilinear entries, the first one is based on the results of Chapter 3 and removes the bilinear terms by modifying the quadratic or linear coefficients of (5.59) while the second one from McCormick [65] adds four linear inequalities for each bilinear term.

In Section 5 of Chapter 3 two different methods are presented for separating the constrains; approximation of the bilinear terms by quadratic, by linear and by constant ones. The choice is made for each bilinear term $b_{jk}x_jx_k$ separately and the decision is based on the bounds of the variables x_k and x_j.

Case 1: If both $\mathbf{x}_k$ and $\mathbf{x}_j$ are bounded we approximate the bilinear term by linear terms, obtaining

$$b_{jk}x_jx_k - b_{jk}z_jx_k - b_{jk}z_kx_j \in [\min_i \nabla u_i, \max_i \Delta u_i]$$

where $z = \operatorname{mid} x$ and

$$\begin{aligned} u_1 &= b_{jk}((\underline{x}_j - z_j)\underline{x}_k - z_k\underline{x}_j), \quad u_2 = b_{jk}((\overline{x}_j - z_j)\underline{x}_k - z_k\overline{x}_j), \\ u_3 &= b_{jk}((\underline{x}_j - z_j)\overline{x}_k - z_k\underline{x}_j), \quad u_4 = b_{jk}((\overline{x}_j - z_j)\overline{x}_k - z_k\overline{x}_j). \end{aligned}$$

This modifies the linear and the constant constraint coefficients to

$$b'_k := b_{jk}z_j + b_k, \quad b'_j := b_{jk}z_k + b_j, \quad \mathbf{c}' := [\underline{c} - \max_i \Delta u_i,\ \underline{c} - \min_i \nabla u_i].$$

Case 2: If the interval $\mathbf{x}_k$ or the interval $\mathbf{x}_j$ is unbounded we eliminate the bilinear terms $b_{jk}x_jx_k$ by adding the quadratic term

$$d_{jk}(x_j - v_{jk}x_k)^2 \text{ with } v_{jk} := \operatorname{sign}(b_{jk})\sqrt{\frac{a_k}{a_j}}, \quad d_{jk} := \frac{b_{jk}}{2v_{jk}},$$

to the constraint. This results in the new quadratic coefficients

$$a'_k := a_k + d_{jk}, \quad a'_j := a_j + \frac{b_{jk}v_{jk}}{2}. \tag{5.60}$$

This results in a separable quadratic expression (5.55), which can be relaxed by using the methods discussed in the Section 5.6.2.

The convex envelope of a function $f(x)$ over the box $\mathbf{x}$ is the tightest convex underestimating function for $f(x)$ for $x \in \mathbf{x}$. Al-Khayyal [6] and McCormick [65] developed an efficient relaxation technique to obtain the convex envelope for the bilinear terms x_jx_k. This requires that $\mathbf{x}_j$ and $\mathbf{x}_k$ are bounded. In this case the convex envelope of $\mathbf{x}_j\mathbf{x}_k$ is convex polyhedral (see Rikun [78]), and its convex and concave parts can be given as

$$\begin{aligned} \operatorname{Conv}(x_jx_k) &:= \max\{\underline{x}_kx_j + \underline{x}_jx_k - \underline{x}_k\underline{x}_j,\ \overline{x}_kx_j + \overline{x}_jx_k - \overline{x}_k\overline{x}_j\}, \\ \operatorname{Conc}(x_jx_k) &:= \min\{\overline{x}_kx_j + \underline{x}_jx_k - \overline{x}_k\underline{x}_j,\ \underline{x}_kx_j + \overline{x}_jx_k - \underline{x}_k\overline{x}_j\}. \end{aligned}$$

Therefore, a linear relaxation of the bilinear terms can be given by substituting a new variables y_{jk} for every $x_j x_k$, and adding the following linear constraints:

$$y_{jk} \geq \underline{x}_k x_j + \underline{x}_j x_k - \underline{x}_k \underline{x}_j, \quad y_{jk} \geq \overline{x}_k x_j + \overline{x}_j x_k - \overline{x}_k \overline{x}_j,$$
$$y_{jk} \leq \overline{x}_k x_j + \underline{x}_j x_k - \overline{x}_k \underline{x}_j, \quad y_{jk} \leq \underline{x}_k x_j + \overline{x}_j x_k - \underline{x}_k \overline{x}_j.$$

Androulakis et al. [8] showed that the maximum difference between variable y_{jk} and the bilinear term $x_j x_k$ depends on the widths of $\mathbf{x}_j$ and $\mathbf{x}_k$ and can be given as $\frac{1}{4}(\overline{x}_j - \underline{x}_j)(\underline{x}_k - \underline{x}_k)$. Therefore, algorithms using convex envelopes to underestimate bilinear terms seek maximal domain reduction, making preprocessing methods helpful in uncovering implicit bounds.

In their Quad algorithm, Lebbah & Rueher [60] used McCormick's convex and concave envelopes to relax the bilinear terms. This results in $4n_b$ additional inequalities which can added to the representation (5.56), increasing the total number of inequalities to $h = n_y(n_z + 1) + 4n_b + 1$ and the number of variables to $n + n_q + n_b$. The method of Domes & Neumaier does not generate additional inequalities but McCormick's method may yield relaxations of higher quality.

5.7 Polynomial constraint satisfaction problems

We consider continuous constraint satisfaction problems of the form

$$G(x) \in \mathbf{F}, \quad x \in \mathbf{x}, \quad G(x) \in \mathbf{G}(x). \tag{5.61}$$

The m general constraint are interpreted as componentwise enclosures $G_i(x) \in \mathbf{F}_i$ $(i = 1, \ldots, m)$. This form includes equality constraints if $\mathbf{F}_i = [\underline{F}_i, \overline{F}_i]$ is a degenerate interval $(\underline{F}_i = \overline{F}_i)$, inequality constraints if one of the bounds is infinite and two–sided constraints $\underline{F}_i \leq G_i(x) \leq \overline{F}_i$ if both bounds are finite. For allowing uncertainties in the constraint coefficients, we allow $G(x)$ to vary in the given interval function $\mathbf{G}(x)$. Similarly, the n bound constraints are interpreted as enclosures $x_j \in \mathbf{x}_j$ with $j = 1, \ldots, n$. Again, fixed variables and one-sided bounds on the variables are included as special cases. Each $x \in \mathbf{x}$ for which the constraints of (5.61) are satisfied, is called a feasible point or a solution of the constraints satisfaction problem. The set of all feasible points is called the feasible domain. If the function $G(x)$ has only quadratic, bilinear and linear terms (5.61) is called a quadratic constraint satisfaction problems. If $G(x)$ is only linear in the variables we end up in the linear system given by (1.8). A linear system of the form of (1.8) can be obtained by relaxing (5.61):

Theorem 5.9. *Every feasible point of the constraint satisfaction problem* (5.61) *satisfies* (1.8) *iff for all* $x \in \mathbf{x}$ *and* $G(x) \in \mathbf{G}(x)$ *the inequalities*

$$Ex - G(x) \in [\underline{b} - \underline{F},\ \overline{b} - \overline{F}] \tag{5.62}$$

hold. In this case, the linear system (1.8) *is a linear relaxation of* (5.61).
If (5.62) *holds, then*

$$Ex \in \mathbf{b}' \text{ with } \mathbf{b}' = \mathbf{b} \cap E\mathbf{x} \tag{5.63}$$

and

$$G(x) \in \mathbf{F}' \text{ with } \mathbf{F}' = \mathbf{F} \cap G(\mathbf{x}) \cap [\inf(E\mathbf{x}) - \overline{b}' + \overline{F},\ \sup(E\mathbf{x}) - \underline{b}' + \underline{F}] \tag{5.64}$$

holds for all $x \in \mathbf{x}$ *and* $G(x) \in \mathbf{G}(x)$.

Proof. That the linear system (1.8) is a linear relaxation of (5.61) follows directly from Theorem 5.8, with $p(x) := G(x)$, $\mathbf{c} := \mathbf{F}$, $h(x) := Ex$, and $\mathbf{d} := \mathbf{b}$ and for all $G(x) \in \mathbf{G}(x)$. By (5.50), every feasible point of (5.61) satisfies (1.8).
In addition to this $Ex \in E\mathbf{x}$ holds for all $x \in \mathbf{x}$ and by (1.8) $Ex \in \mathbf{b}$ also holds for all $x \in \mathbf{x}$ proving (5.63).
Since (5.63) is a linear relaxation of (5.61) by Theorem 5.9 the two-sided inequality (5.62) holds, implying that

$$G(x) \in [Ex - \overline{b}' + \overline{F},\ Ex - \underline{b}' + \underline{F}]. \tag{5.65}$$

Since $Ex \in E\mathbf{x}$ for all $x \in \mathbf{x}$, with (5.65) implies that

$$G(x) \in [\inf(E\mathbf{x}) - \overline{c} + \overline{F},\ \sup(E\mathbf{x}) - \underline{c} + \underline{F}]. \tag{5.66}$$

for all $x \in \mathbf{x}$ and for all $G(x) \in \mathbf{G}(x)$. From this, (5.64) follows since both $G(x) \in G(\mathbf{x})$, and $G(x) \in \mathbf{F}$ holds for all $x \in \mathbf{x}$ and for all $G(x) \in \mathbf{G}(x)$. □

If $G(x)$ is quadratic in x, the linear relaxations of (5.61) can be computed according to the results of the previous section. If for each constraint the quadratic terms are relaxed by the method of Kolev and the bilinear terms are eliminated by our approach, the resulting linear system (1.8) has m inequalities and n variables. If the approach of Lebbah & Rueher for the quadratic terms is combined with our approach for eliminating the bilinear terms, the resulting linear system has at most $m(3n + 4)$ inequalities and $2n$ variables. The original method of Lebbah & Rueher results in a linear system of at most $m(7n + 4)$ inequalities and $3n$ variables. The methods discussed above can now be applied to the linear relaxation in order to obtain tighter bounds on the variables. The

following corollary shows how the relaxation and the new bounds on the variables can be used to tighten the bounds of the constraints of (5.61).

If we have tightened the bounds on the variables, the bounds of the relaxation can be tightened with (5.63). With (5.64) we may also tighten the bounds on the general constraints of the original constraint satisfaction problem.

The Gauss-Jordan preconditioner Algorithm 5.3 from Section 5.3 can be also applied to precondition a quadratic system. In GloptLab (see Chapter 2), the quadratic constraints are represented as

$$Aq(x) \in \mathbf{F}, \quad x \in \mathbf{x}, \quad A \in \mathbf{A}, \tag{5.67}$$

where $A \in \mathbb{R}^{m \times n^2+n}$ is a (generally sparse) matrix, $\mathbf{A}$ represents the bounds for the constraint coefficients, $\mathbf{x}$ is n-dimensional and $\mathbf{F}$ is m-dimensional. The linear, quadratic, and bilinear monomials occurring in at least one of the constraints (but not the constant term) are collected into the $n^2 + n$ dimensional column vector

$$q(x) := (x_1, \ldots, x_n,\ x_1^2, \ldots, x_1x_n, \ldots, x_nx_1, \ldots, x_n^2)^T.$$

For this system the Gauss-Jordan preconditioner algorithm 5.3 can be applied.

All our methods can be applied after suitable preprocessing to arbitrary algebraic constraints. We can always transform a polynomial constraint to a collection of quadratic constraints by introducing explicit intermediate variables, and the same holds for constraints involving roots, provided that we also add nonnegativity constraints to the intermediate variables representing the roots. Rewriting an algebraic constraint satisfaction problem as an equivalent problem with linear and quadratic constraints only increases the number of variables, but allows one to apply the methods discussed in this chapter. Of course, all techniques can be applied to the subset of quadratic (or algebraic) constraints in an arbitrary constraint satisfaction problem.

How the different techniques presented in this chapter can be applied and combined to solve quadratic constraint satisfaction problems is visualized in Figure (5.3).

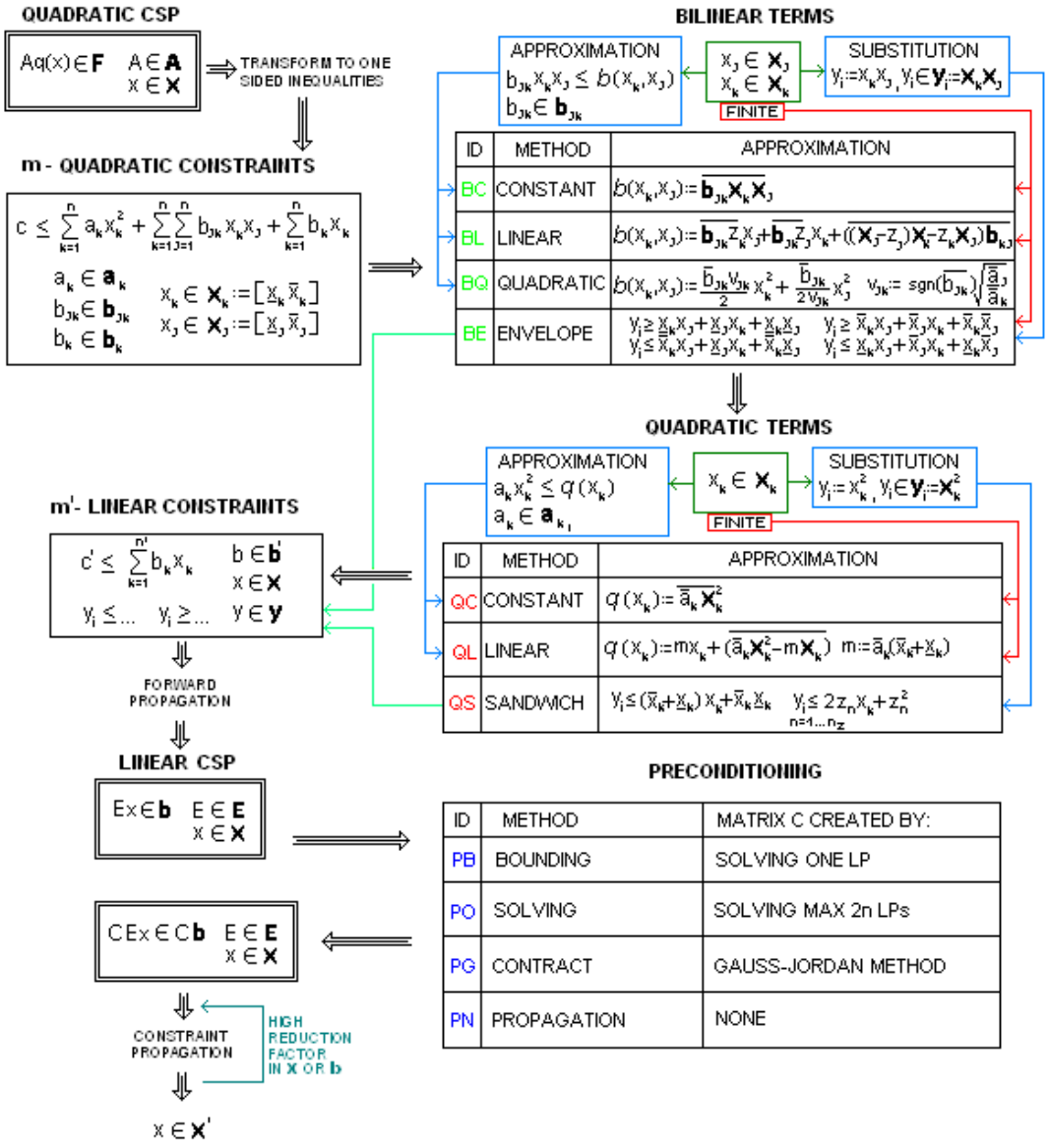

FIGURE 5.3: Rigorous filtering using linear relaxations.

5.8 Examples

We first give a simple example of a quadratic constraint satisfaction problem, demonstrate linearization by Lebbah and Kolev and solve the arising linear system by both linear contraction (see, Section 5.4) and linear bounding (see, Section 5.5).

Example 5.4. *Let*

$$x_1^2 + x_2^2 \leq 25, \quad x_1 \in \mathbf{x}_1 = [4,5], \quad x_2 \in \mathbf{x}_2 = [0,5]. \tag{5.68}$$

We linearize the quadratic expression (5.68). *Since both quadratic terms* x_1^2 *and* x_2^2 *have positive coefficients we compute the tangents*

$$t(x_i) := mx_i + d, \quad t(x_i) \leq x_i^2, \quad m = 2\tilde{x}_i, \quad d = \tilde{x}_i^2 - m\tilde{x}_i$$

for $i = 1, 2$ *at the midpoints* $\tilde{x}_1 = 4.5$ *and* $\tilde{x}_2 = 2.5$ *of the intervals* $\mathbf{x}_1$ *and* $\mathbf{x}_2$ *obtaining*

$$t(x_1) = 9x_1 - 20.25 \leq x_1^2, \; t(x_2) = 5x_2 - 6.25 \leq x_2^2. \tag{5.69}$$

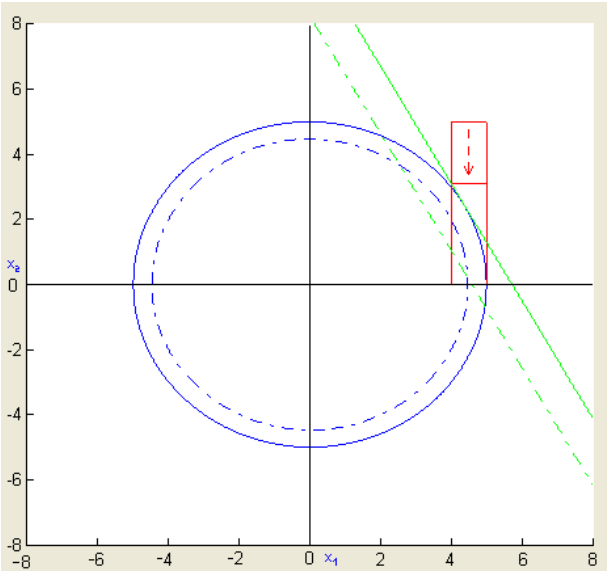

FIGURE 5.4: Example of solving a quadratic constraint satisfaction problem by linear relaxation. The arrow indicates the reduction of the bound $\overline{x}_2$.

By **Lebbah's method** *we substitute the new variables* $x_3 \in \mathbf{x}_1^2$ *and* $x_4 \in \mathbf{x}_2^2$ *for the terms* x_1^2 *and* x_2^2 *and get the linear relaxation*

$$\begin{aligned} &x_3 + x_4 \leq 25, \\ &9x_1 - x_3 \leq 20.25 \\ &5x_2 - x_4 \leq 6.25 \\ &x_1 \in [4, 5], \quad x_2 \in [0, 5], \quad x_3 \in [16, 25], \quad x_4 \in [0, 25], \end{aligned} \tag{5.70}$$

of the constraint satisfaction problem (5.68).

Then we can use linear contraction to obtain tighter bounds on x_2*: In the first step we have*

$$\begin{aligned} x_3 + x_4 \leq 25, \; 16 \leq x_3, \; 0 \leq x_4 \quad &\Rightarrow \quad x_3 \leq 25, \; x_4 \leq 9 \\ 9x_1 - x_3 \leq 20.25, \; 4 \leq x_1 \quad &\Rightarrow \quad 15.75 \leq x_3 \\ 5x_2 - x_4 \leq 6.25, \; 0 \leq x_2 \quad &\Rightarrow \quad -6.25 \leq x_4, \end{aligned}$$

getting improved bounds $x_4 \in [0, 9]$ *and in the second step*

$$5x_2 - x_4 \leq 6.25, \quad -9 \leq -x_4, \ 0 \leq x_2 \quad \Rightarrow \quad x_2 \in [0, 3.05].$$

If we use the linear bounding and minimize x_1, $-x_1$, x_2, *and* $-x_2$ *subject to the constraints* (5.70) *we obtain the approximate multiplier matrix* Y *which has all zero rows expect for the last row* $Y_{4:} = \begin{pmatrix} 0.2 & 0 & 0.2 \end{pmatrix}$ *corresponding to the objective* $-x_2$. *The matrix representation of* (5.70) *can be given as*

$$Ex \geq c, \quad E = \begin{pmatrix} 0 & 0 & -1 & -1 \\ -9 & 0 & 1 & 0 \\ 0 & -5 & 0 & 1 \end{pmatrix}, \quad c = \begin{pmatrix} -25 & -20.25 & -6.25 \end{pmatrix}^T.$$

Since the first three rows of Y *are zero, the first three of the inequalities* $YE \geq Yc$ *are trivial and the last one is*

$$5x_2 + 0.2x_3 + 2.7 \cdot 10^{-17} x_4 \leq 6.25$$

can be solved by substituting the lower bounds for x_3 *and* x_4, *obtaining* $x_2 \leq 3.005$.

We use **Kolev's method** *to linearize the quadratic expression* (5.68) *by substituting* (5.69) *into it, obtaining*

$$9x_1 - 20.25 + 5x_2 - 6.25 \leq x_1^2 + x_2^2 \leq 25,$$

ending up in

$$9x_1 + 5x_2 \leq 51.5 \quad x_1 \in [4, 5], \quad x_2 \in [0, 5].$$

From there a single step of linear contraction

$$9x_1 + 5x_2 \leq 51.5 \quad 4 \leq x_1 \quad \Rightarrow \quad x_2 \leq 3.1,$$

yields improved bounds on x_2.

If we use the linear bounding method we obtain the approximate multiplier matrix Y *which has all zero rows expect for the last row* $Y_{4:} = \begin{pmatrix} 0.2 & 1.8 \end{pmatrix}$ *resulting in*

$$x_2 \leq 10.3 - 1.8x_1 \leq 10.3 - 1.8\underline{x}_1 = 3.1.$$

The second example extends the first one by solving a simple system of separable constraint satisfaction problem.

Example 5.5. *Let*

$$x_1^2 + x_1x_2 + x_2^2 \leq 25, \quad x_1 \in \mathbf{x}_1,\ \mathbf{x}_1 = [4,5], \quad x_2 \in \mathbf{x}_2,\ \mathbf{x}_2 = [0,5]. \tag{5.71}$$

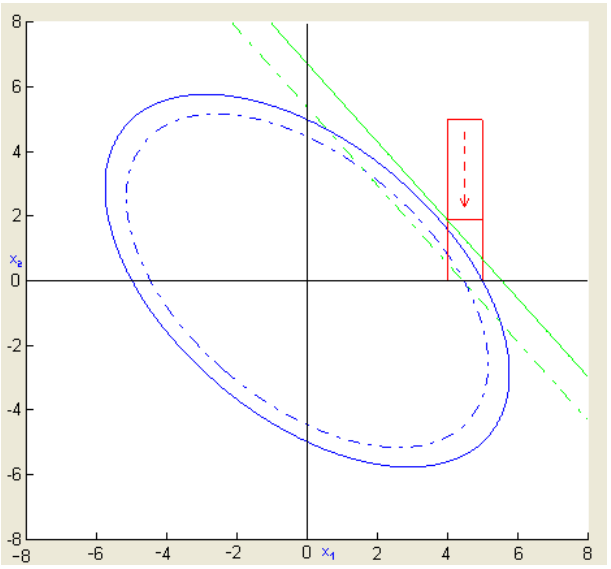

FIGURE 5.5: Example of solving a separable constraint satisfaction problem by linear relaxation. The arrow indicates the reduction of the bound $\overline{x}_2$.

By **Lebbah's method** *we compute the McCormick relaxations for the bilinear terms and approximate the quadratic terms as in the previous example, obtaining*

$$\begin{aligned}
&x3 + x_4 + x_5 \leq 25,\\
&4x_2 - x_3 \leq 0,\\
&5x_1 + 5x_2 - x_3 \leq 25,\\
&9x_1 - x_4 \leq 20.25,\\
&5x_2 - x_5 \leq 6.25,\\
&x_1 \in [4,5], \quad x_2 \in [0,5], \quad x_3 \in [0,25], \quad x_4 \in [16,25], \quad x_5 \in [0,25].
\end{aligned} \tag{5.72}$$

Solving (5.72) *with linear contraction results in* $x_2 \leq 2.25$ *while by solving with linear bounding we get* $x_2 \leq 1.6944$.

By **Kolev's method** *we first separate* (5.71) *by approximating the bilinear term* x_1x_2 *by linear terms obtaining*

$$x_1^2 + x_2^2 + 2.5x_1 + 4.5x_2 \leq 37.5,$$

then we approximate the quadratic terms as in the previous example, ending up in

$$11.5x_1 + 9.5x_2 \leq 64. \tag{5.73}$$

Solving (5.73) *with either linear contraction or with the linear bounding we get* $x_2 \leq 1.8947$.

5.9 Test Results

In this section we compare different linearization techniques. We use the following linear relaxation methods to reduce the bounds of the different test problems:

`identifier`	`bilinear`	`quadratic`
`LinCL`	constant	linear
`LinLL`	linear	linear
`LinQL`	quadratic	linear
`LinLN`	linear	new inequalities
`LinEL`	envelope	linear
`LinEN`	envelope	new inequalities

whereby the `bilinear` column describes the technique for approximating the bilinear terms and the `quadratic` column describes the technique for approximating the quadratic terms.

After the linearization the three most promising variables are chosen, and linear solving are used tho reduce the bound constraints (for details see Section 5.5). Each method is applied to all test problems of a test set, one by one. If a method fails to reduce the bound constraints for some of the test problems, these will be solved again with the same method but with tighter bound constraints. With each retry the width of the bound constraints are reduced by 33% but the retries are counted and the solution times are summed up. The table below shows the average solution times (in seconds), the minimum, the average and the maximum number of retires as well as the average of the gain:

$$g := \frac{1}{n} \sum_{i=1}^{n} \frac{\mathrm{wid}(\mathbf{x}_i')}{\mathrm{wid}(\mathbf{x}_i)}$$

where $\mathbf{x}$ are the original bounds on the n variables, and $\mathbf{x}'$ the reduced bounds.

The first test consist of 200, two dimensional, quadratic, random generated problems. Each problems has two equality constraints which intersect at least at the origin. We set bound constraints for each variable ranging from -10 to 10.

Linearization Test Results.			
`dimension`	`n = 2`		
`method`	`time`	`retry`	`gain`
`LinCL`	0.136	[0 1.345 3]	0.127
`LinLL`	0.159	[0 1.345 3]	0.127
`LinQL`	0.110	[0 0.515 3]	0.187
`LinLN`	0.128	[0 0.305 3]	0.186
`LinEL`	0.132	[0 0.6 3]	0.201
`LinEN`	0.102	[0 0 0]	0.388

The second test consist quadratic, random generated problems consisting equality constraints which intersect at least at the origin. The test parameters are shown in the table below, followed by the test results.

Test case parameters.						
name	probs	variables		constraints		
		#	bounds	#	relations	types
Test 1	20	2	$[-20, 20]$	2	equalities	ellipsoids
Test 2	20	5	$[-1, 1]$	5	equalities	ellipsoids
Test 3	20	10	$[-0.1, 0.1]$	10	equalities	ellipsoids

Linearization Test Results.									
`dimen`	`n = 2`			`n = 5`			`n = 10`		
`method`	`time`	`retry`	`gain`	`time`	`retry`	`gain`	`time`	`retry`	`gain`
`LinCL`	0.134	[0 2.3 3]	0.107	0.260	[0 1 2]	0.019	0.185	[0 0 0]	0.054
`LinLL`	0.166	[0 2.3 3]	0.107	0.734	[0 1 2]	0.019	2.276	[0 0 0]	0.054
`LinQL`	0.137	[0 1.25 3]	0.174	0.287	[0 0.3 2]	0.040	0.511	[0 0 0]	0.070
`LinLN`	0.248	[0 1.65 3]	0.159	0.511	[0 0.3 1]	0.020	2.360	[0 0 0]	0.068
`LinEL`	0.140	[0 0.75 3]	0.225	0.193	[0 0 0]	0.095	0.568	[0 0 0]	0.153
`LinEN`	0.107	[0 0 0]	0.422	0.232	[0 0 0]	0.192	0.630	[0 0 0]	0.183

The third test consist of 200, two dimensional, quadratic, random generated problems. Each problems has a single inequality constraint describing the boundary and the interior of an ellipsoid through the origin. The bound constraints are set to $[0, 3]$, no retries are allowed and the number of problems where the methods did not improve the bound constraints is listed in the column `no gain`.

Linearization Test Results.			
dimension	n = 2		
method	time	gain	no gain
LinCL	0.061	0.221	96
LinLL	0.063	0.205	90
LinQL	0.065	0.218	98
LinLN	0.087	0.266	72
LinEL	0.080	0.257	87
LinEN	0.084	0.341	72

As the test show introducing new variables instead of approximating the bilinear terms by constant linear or quadratic ones and the quadratic terms by constant or linear ones yields more gain but is slower even in low dimensions.

Remark: The the idea of the Gauss-Jordan precoditioning and Bounding a polytope came from Arnold Neumaier. The remaining Sections are my work.

List of Figures

Bibliography

[1] J. Adams. Computer computations in representation theory III: Unitary representation of real Lie groups. Technical report, University of Maryland, 2002. URL www.math.umd.edu/~jda/minicourse.

[2] C. S. Adjiman, I. P. Androulakis, and C. A. Floudas. A global optimization method, αBB, for general twice-differentiable constrained NLPs - II. implementation and computational results. *Computers and Chemical Engineering*, 22:1159–1179, 1998.

[3] C. S. Adjiman, I. P. Androulakis, C. D. Maranas, and C. A. Floudas. A global optimization method αBB for process design. *Computers and Chemical Engineering*, 20:419–424, 1996.

[4] C. S. Adjiman, S. Dallwig, C. A. Floudas, and A. Neumaier. A global optimization method, αBB, for general twice-differentiable constrained NLPs - I. theoretical advances,. *Computers and Chemical Engineering*, 22:1137–1158, 1998.

[5] C. S. Adjiman and C. A. Floudas. Rigorous convex underestimators for general twice-differentiable problems. *Journal of Global Optimization*, 9:23–40, 1996.

[6] F. Al-Khayyal. Jointly constrained biconvex programming and related problems: An overview. *Comput. Math. Appl*, 19:53–62, 1990.

[7] E. D. Anderson and K. D. Anderson. Presolving in linear programming. *Math. Program.*, 71:221–245, 1995.

[8] I. P. Androulakis, C. D. Maranas, and C. A. Floudas. αBB: a global optimization method for general constrained nonconvex problems. *Journal of Global Optimization*, 7:337–363, 1995.

[9] K. R. Apt. The essence of constraint propagation. *Theoretical Computer Science*, 221(1):179–210, 1999.

[10] A. B. Babichev, O. B. Kadyrova, T. P. Kashevarova, A. S. Leshchenko, and A. L. Semenov. UniCalc, a novel approach to solving systems of algebraic equations. *Interval Computations*, 3:29–47, 1993.

[11] F. Benhamou. Heterogeneous constraint solving. In M. Hanus and M. Rodrguez-Artalejo, editors, *Proceedings of ALP'96, 5th International Conference on Algebraic and Logic Programming*, vol. 1139 of *Lecture Notes in Computer Science*, pp. 62–76, Aachen, Germany, 1996. Springer-Verlag.

[12] F. Benhamou, F. Goualard, L. Granvilliers, and J. F. Puget. Revising hull and box consistency. In *International Conference on Logic Programming*, pp. 230–244, 1999. URL citeseer.ist.psu.edu/benhamou99revising.html.

[13] F. Benhamou, L. Granvilliers, and F. Goualard. Interval constraints: Results and perspectives. In *New Trends in Constraints*, pp. 1–16, 1999. URL citeseer.ist.psu.edu/benhamou99interval.html.

[14] F. Benhamou, D. McAllister, and P. Van Hentenryck. CLP(intervals) revisited. In *Proc. International Symposium on Logic Programming*, pp. 124–138. MIT Press, 1994.

[15] F. Benhamou and W. J. Older. Applying interval arithmetic to real, integer, and boolean constraints. *Journal Logic Program*, 32:1–24, 1997.

[16] M. Berkelaar, J. Dirks, K. Eikland, and P. Notebaert. LpSolve, 1991 - 1999. URL http://lpsolve.sourceforge.net/5.5/.

[17] C. Bliek, P. Spelucci, L. N. Vicente, A. Neumaier, L. Granvilliers, E. Monfro, F. Benhamou, E. Huens, P. Van Hentenryck, D. Sam-Haround, and B. Faltings. Algorithms for solving nonlinear constrained and optimization problems: the state of the art, 2000. URL http://www.mat.univie.ac.at/~neum/ms/StArt.pdf.

[18] A. Brooke, D. Kendrick, and A. Meeraus. GAMS: A user's guide, 1992. URL citeseer.ist.psu.edu/brooke92gams.html.

[19] M. Ceberio and L. Granvilliers. Solving nonlinear systems by constraint inversion and interval arithmetic. In *AISC*, pp. 127–141, 2000. URL http://link.springer.de/link/service/series/0558/bibs/1930/19300127.htm.

[20] G. Chabert and L. Jaulin. Hull consistency under monotonicity. In *Principle and practices of constraint programming - CP2009*, pp. 188–195, 2009.

[21] H. M. Chen and M. H. van Emden. Adding interval constraints to the Moore–Skelboe global optimization algorithm. In V. Kreinovich, editor, *Extended Abstracts of APIC'95 of the International Workshop on Applications of Interval Computations*, pp. 54–57, 1995.

[22] J. G. Cleary. Logical arithmetic. *Future Computing Systems*, 2:125–149, 1987.

[23] J. Cruz and P. Barahona. Constraint reasoning in deep biomedical models. *Journal of Artificial Intelligence in Medicine*, 34:77–88, 2005. URL http://ssdi.di.fct.unl.pt/~pb/papers/ludi_constraints.pdf.

[24] J. Czyzyk, M. Mesnier, and J. Moré. The NEOS Server. *IEEE Journal on Computational Science and Engineering*, 5:68–75, 1998. URL http://www-neos.mcs.anl.gov/.

[25] S. Dallwig, A. Neumaier, and H. Schichl. GLOPT - a program for constrained global optimization. In I. Bomze, T. Csendes, R. Horst, and P. M. Pardalos, editors, *Developments in Global Optimization*, pp. 19–36. Kluwer, Dordrecht, 1997.

[26] N. S. Dimitrova and S. M. Markov. Über die intervall-arithmetische Berechnung des Wertebereichs einer Funktion mit Anwendungen, Freiburger Intervall-Berichte. *Univ. Freiburg*, 81:1–22, 1981.

[27] F. Domes. GloptLab – a configurable framework for the rigorous global solution of quadratic constraint satisfaction problems. *Optimization Methods and Software*, 24:727–747, 2009. URL http://www.mat.univie.ac.at/~dferi/publ/Gloptlab.pdf.

[28] F. Domes, M. Fuchs, and H. Schichl. The Optimization Test Environment. *Optimization Methods and Software*, 2010. submitted. URL http://www.mat.univie.ac.at/~dferi/testenv.html.

[29] F. Domes and A. Neumaier. A scaling algorithm for polynomial constraint satisfaction problems. *Journal of Global Optimization*, 43:327–345, 2008. URL http://www.mat.univie.ac.at/~dferi/publ/Scaling.pdf.

[30] F. Domes and A. Neumaier. Constraint propagation on quadratic constraints. *Constraints*, 2009. ISSN 1383–7133. URL http://www.mat.univie.ac.at/~dferi/publ/Propag.pdf.

[31] F. Domes and A. Neumaier. Rigorous enclosures of ellipsoids and directed Cholesky factorizations. submitted, 2009. URL http://www.mat.univie.ac.at/~dferi/publ/Cholesky.pdf.

[32] F. Domes and A. Neumaier. Finding and verifying feasible points of polynomial constraint satisfaction problems. in preparation, 2010. URL http://www.mat.univie.ac.at/~dferi/publ/.

[33] F. Domes and A. Neumaier. Rigorous filtering using linear relaxations. in preparation, 2010. URL http://www.mat.univie.ac.at/~dferi/publ/.

[34] F. Domes and A. Neumaier. Using conic programs to solve quadratic constraint satisfaction problems. in preparation, 2010. URL http://www.mat.univie.ac.at/~dferi/publ/.

[35] R. Fourer, D. M. Gay, and B. W. Kernighan. AMPL – a modeling language for mathematical programming, 2002. Software. URL http://www.ampl.com/.

[36] J. Garloff, C. Jansson, and A. Smith. Lower bound functions for polynomials. *Journal of Computational and Applied Mathematics*, 157:207–225, 2003. URL citeseer.ist.psu.edu/534450.html.

[37] C. Grandon, D. Daney, and Y. Papegay. Combining CP and interval methods for solving the direct kinematic of a parallel robot under uncertainties. IntCP 06 Workshop, 2006. URL ftp://ftp-sop.inria.fr/coprin/daney/articles/intcp06.pdf.

[38] L. Granvilliers and F. Benhamou. Realpaver: An interval solver using constraint satisfaction techniques. *ACM Transactions on Mathematical Software*, 32:38–156, 2006. URL http://www.sciences.univ-nantes.fr/info/perso/permanents/granvil/realpaver/.

[39] G. D. Hager. Solving large systems of nonlinear constraints with application to data modeling. *Interval Computations*, 3:169–200, 1993.

[40] E. R. Hansen and G. W. Walster. Sharp bounds on interval polynomial roots. *Reliable Computing*, 8:115–122, 2002.

[41] P. Van Hentenryck. A Gentle Introduction to Numerica. *Artifical Intelligence*, 103:209–235, 1998.

[42] P. Van Hentenryck, L. Michel, and F. Benhamou. Newton: constraint programming over non-linear constraints. *Sci. Program*, 30:83–118, 1997.

[43] P. Van Hentenryck, L. Michel, and Y. Deville. *Numerica. A modeling language for global optimization*. MIT Press, 1997.

[44] N. J. Higham. *Accuracy and stability of numerical algorithms*, chapter 10. Siam, Philadelphia, 1996.

[45] E. Hyvönen and S. De Pascale. Interval computations on the spreadsheet. In R. B. Kearfott and V. Kreinovich, editors, *Applications of Interval Computations*, pp. 169–209. Kluwer, 1996.

[46] C. Jansson. VSDP: A MATLAB software package for verified semidefinite programming. In *Conference paper of NOLTA 2006*, p. 327330, 2006. URL http://www.ti3.tu-harburg.de/paper/jansson/Nolta06.pdf.

[47] C. Jansson and S. M. Rump. Rigorous solution of linear programming problems with uncertain data. *ZOR Methods and Models of Operations Research*, 35:87111, 1991. URL http://www.ti3.tu-harburg.de/paper/rump/JaRu91.pdf.

[48] L. Jaulin. Interval constraint propagation with application to bounded-error estimation. *Automatica*, 36:1547–1552, 2000. URL https://www.ensieta.fr/e3i2/Jaulin/hull.pdf.

[49] L. Jaulin. Interval constraints propagation techniques for the simultaneous localization and map building of an underwater robot, 2006. URL http://www.mat.univie.ac.at/~neum/glopt/gicolag/talks/jaulin.pdf.

[50] L. Jaulin, M. Kieffer, I. Braems, and E. Walter. Guaranteed nonlinear estimation using constraint propagation on sets. *International Journal of Control*, 74:1772–1782, 1999. URL https://www.ensieta.fr/e3i2/Jaulin/observer.pdf.

[51] C. Jermann, Y. Lebbah, and D. Sam-Haroud. Interval analysis, constraint propagation and applications. In F. Benhamou, N. Jussien, and B. O'Sullivan, editors, *Trends in Constraint Programming*, chapter 4, pp. 223–259. ISTE, 2007.

[52] N. Jussien and V. Barichard. The PaLM system: explanation-based constraint programming. In *Proceedings of TRICS: Techniques for Implementing Constraint programming Systems, a post-conference workshop of CP 2000*, pp. 118–133, September 2000. URL http://www.emn.fr/jussien/publications/jussien-WCP00.pdf.

[53] R. B. Kearfott. Decomposition of arithmetic expressions to improve the behavior of interval iteration for nonlinear systems. *Computing*, 47:169–191, 1991.

[54] R. B. Kearfott. GlobSol user guide. *Optimization Methods and Software*, 24: 687–708, 2009.

[55] C. Keil. Lurupa - rigorous error bounds in linear programming. In *Algebraic and Numerical Algorithms and Computer-assisted Proofs*, 2005.

[56] C. T. Kelley. Iterative methods for optimization – Matlab codes, 1999. Software. URL http://www4.ncsu.edu/~ctk/matlab_darts.html.

[57] L. V. Kolev. Automatic computation of a linear interval enclosure. *Reliable Computing*, 7(1):17–28, 2001.

[58] L. Krippahl and P. Barahona. PSICO: Solving protein structures with constraint programming and optimization. *Constraints*, 7:317–331, 2002. URL http://ssdi.di.fct.unl.pt/~pb/papers/ludi_constraints.pdf.

[59] Y. Lebbah. iCOs – Interval COnstraints Solver, 2003. URL http://ylebbah.googlepages.com/icos.

[60] Y. Lebbah, C. Michel, and M. Rueher. A rigorous global filtering algorithm for quadratic constraints. *Constraints*, 10:47–65, 2005. URL http://ylebbah.googlepages.com/research.

[61] Y. Lebbah, C. Michel, M. Rueher, D. Daney, and J-P. Merlet. Efficient and safe global constraints for handling numerical constraint systems. *SIAM Journal on Numerical Analysis*, 42(5):2076–2097, 2005. URL http://ylebbah.googlepages.com/research.

[62] W. A. Lodwick. Constraint propagation, relational arithmetic in ai systems and mathematical programs. *Ann. Oper. Res*, 21:143–148, 1989.

[63] M. C. Markót. SMPL - A Simplified Modeling Language for Mathematical Programming. Technical report, University of Vienna, 2008. URL http://www.mat.univie.ac.at/~dferi/Gloptlab/index.html.

[64] M. C. Markót and T. Csendes. A new verified optimization technique for the "packing circles in a unit square" problems. *SIAM Journal on Optimization*, 16: 193–219, 2005. URL http://www.mat.univie.ac.at/~markot/impcirc.pdf.

[65] G. McCormick. Computability of global solutions to factorable non-convex programs part I Convex underestimating problems. *Math. Prog.*, 10:147175, 1976.

[66] J-P. Merlet. Solving the forward kinematics of a Gough-type parallel manipulator with interval analysis. *International Journal of Robotics Research*, 23(3):221–235, 2004. URL http://www-sop.inria.fr/coprin/equipe/merlet/Papers/IJRR2004.pdf.

[67] J-P. Merlet. Solving the forward kinematics of a Gough-type parallel manipulator with interval analysis. *International Journal of Robotics Research*, 23(3):221–236, 2004. URL citeseer.ist.psu.edu/merlet04solving.html.

[68] A. Morgan. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems*. Prentice-Hall, 1987.

[69] T. S. Motzkin. *Beitrge zur Theorie der linearen Ungleichungen*. PhD thesis, Basel, Jerusalem, 1936.

[70] A. Neumaier. *Interval methods for systems of equations*, vol. 37 of *Encyclopedia of Mathematics and its Applications*. Cambridge Univ. Press, Cambridge, 1990.

[71] A. Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. *SIAM Rev.*, 39:407–460, 1997. URL http://www.mat.univie.ac.at/~neum/ms/protein.pdf.

[72] A. Neumaier. Enclosing clusters of zeros of polynomials. *Journal Comput. Appl. Math.*, 156:389–401, 2003.

[73] A. Neumaier. Complete search in continuous global optimization and constraint satisfaction. *Acta Numerica*, 1004:271–369, 2004.

[74] A. Neumaier. Computer-assisted proofs. In *Proc. 12th GAMM-IMACS (SCAN 2006)*. IEEE Computer Society, 2007. URL http://www.mat.univie.ac.at/~neum/ms/caps.pdf.

[75] A. Neumaier, O. Shcherbina, W. Huyer, and T. Vinkó. A comparison of complete global optimization solvers. *Math. Programming B*, 103:335–356, 2005.

[76] W. Older and A. Vellino. Constraint arithmetic on real intervals. In F. Benhameou and A. Colmerauer, editors, *Constrained Logic Programming: Selected Research*. MIT Press, 1993.

[77] F. Rendl, G. Rinaldi, and A. Wiegele. Solving Max-Cut to Optimality by Intersecting Semidefinite and Polyhedral Relaxations, 2007. URL http://biqmac.uni-klu.ac.at/rrw.pdf.

[78] A. D. Rikun. A convex envelope formula for multilinear functions. *Journal of Global Optimization*, 10:425437, 1997.

[79] S. M. Rump. Solving algebraic systems with high accuracy. In *A New Approach to Scientific Computation*, pp. 51–120. Academic Press, 1993.

[80] S. M. Rump. Validated solution of large linear systems. In *Validation Numerics*. Springer, 1993.

[81] S. M. Rump. Verification methods for dense and sparse systems of equations. In *Topics in validated computations*. North Holland, 1994.

[82] S. M. Rump. INTLAB – INTerval LABoratory, 1998 - 2008. URL http://www.ti3.tu-harburg.de/~rump/intlab/.

[83] S. M. Rump. Verification of positive definiteness. *BIT Numerical Mathematics*, 46:433–452, 2006.

[84] S. M. Rump and T. Ogita. Super-fast validated solution of linear systems. *Journal Comput. Appl. Math.*, 199(2), 2007.

[85] N. Sahinidis and M. Tawarmalani. *Convexification and global optimization in continuous and mixed–integer nonlinear programming: theory, algorithms, software, and applications*. Kluwer Academic Pub., 2003.

[86] N. V. Sahinidis and M. Tawarmalani. *BARON 7.2.5: global optimization of mixed-integer nonlinear programs,* User's Manual, 2005. URL http://www.gams.com/dd/docs/solvers/baron.pdf.

[87] H. Schichl. Mathematical modeling and global optimization, habilitation thesis, 2003. URL http://www.mat.univie.ac.at/~herman/papers/habil.pdf.

[88] H. Schichl, M. C. Markót, A. Neumaier, Xuan-Ha Vu, and C. Keil. The COCONUT Environment, 2000-2010. Software. URL http://www.mat.univie.ac.at/coconut-environment.

[89] H. Schichl and A. Neumaier. Interval Analysis on Directed Acyclic Graphs for Global Optimization. *Journal of Global Optimization*, 33(4):541–562, 2005.

[90] H. Schichl and A. Neumaier. Transposition theorems and qualification-free optimality conditions. *Siam Journal Optimization*, 17:1035–1055, 2006. URL http://www.mat.univie.ac.at/~neum/ms/trans.pdf.

[91] O. Shcherbina, A. Neumaier, D. Sam-Haroud, Xuan-Ha Vu, and Tuan-Viet Nguyen. Benchmarking global optimization and constraint satisfaction codes. In Ch. Bliek, Ch. Jermann, and A. Neumaier, editors, *Global Optimization and Constraint Satisfaction*, pp. 211–222. Springer, 2003. URL http://www.mat.univie.ac.at/~neum/ms/bench.pdf.

[92] H. Sherali and W. Adams. *A Reformulation-Linearization Technique for Solving Discrete and Continuous Nonconvex Problems*. Kluwer Academic Publ., 1999.

[93] M. Skutella. Convex quadratic and semidefinite programming relaxations in scheduling. *Journal ACM*, 48(2):206–242, 2001.

[94] J. Stoer and R. Bulirsch. *Introduction to numerical analysis*. Springer, 2002.

[95] J. F. Sturm, O. Romanko, and I. Pólik. SeDuMi, 1997 - 2008. URL http://sedumi.mcmaster.ca/.

[96] P. G. Szabó, M. C. Markót, T. Csendes, E. Specht, L. G. Casado, and I. Garca. *New Approaches to Circle Packing in a Square - With Program Codes*. Springer, Berlin, 2008.

[97] M. Tawarmalani and N. Sahinidis. Global optimization of mixed-integer nonlinear programs: A theoretical and computational study. *Math. Program.*, 99(3), 2004.

[98] K. C. Toh, M. J. Todd, and R. H. Tutuncu. SDPT3 – a Matlab software package for semidefinite programming, 1999. URL http://www.math.nus.edu.sg/~mattohkc/sdpt3.html.

[99] M. H. van Emden. Computing functional and relational box consistency by structured propagation in atomic constraint systems. *CoRR*, cs.PL/0106008, 2001.

[100] L. Vandenbergh and S. Boyd. Semidefinite programming. *SIAM Review*, 38:49–95, 1996.

[101] Xuan-Ha Vu, H. Schichl, and D. Sam-Haroud. Using directed acyclic graphs to coordinate propagation and search for numerical constraint satisfaction problems. In *In Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*, pp. 72–81, 2004. URL http://www.mat.univie.ac.at/~herman/papers/ICTAI2004.pdf.

[102] Xuan-Ha Vu, H. Schichl, and D. Sam-Haroud. Interval propagation and search on directed acyclic graphs for numerical constraint solving. *Journal of Global Optimization*, p. 39, 2007. to appear. URL http://www.mat.univie.ac.at/~herman/papers/FBPD-Hermann.pdf.

[103] L. T. Watson and L. Terry. HOMPACK: a suite of codes for globally convergent homotopy algorithms, 1985. URL http://deepblue.lib.umich.edu/dspace/bitstream/2027.42/8204/5/ban6930.0001.001.pdf.

[104] C. Zhu, R. H. Byrd, and J. Nocedal. L-BFGS-B: Algorithm 778: L-BFGS-B, FORTRAN routines for large scale bound constrained optimization, 1997. URL http://www.ece.northwestern.edu/~nocedal/lbfgsb.html.

CURRICULUM VITAE

Name: Ferenc Domes

Date of birth: 1975

Birthplace: Budapest, Hungary

E-mail: ferenc.domes@univie.ac.at

Homepage: http://www.mat.univie.ac.at/~dferi/

Education and Employment:

1982-1990	Keve Elementary School in Budapest
1990-1994	Toldy Ferenc Grammar School in Budapest
1994-1997	Working for the company Hydrotech
1997-1999	Studies of Mathematics at the University of Vienna, Austria
1999-2000	Mathematics and informatics teacher at the Batthyány László Római Catholic Blind Children's Home in Hungary
2000	Continued studies of Mathematics with main focus on Computational Mathematics
2002	Project: Interval Analysis and Global Optimization
2003	Project: Heuristic Global Search
2003	Diploma thesis: Chaos Control - Simulating and Controlling Chaotic Dynamical Systems
2004	Master of Science in Mathematics
2005-now	Member of the the research group CMA (Computational Mathematics) at the University of Vienna
2006-2007	Research assistant in the project: COCONUT – a Software Enironment for Global Optimization
2007-now	Research assistant in the Project: Advanced Modeling in Global Optimization

♢