FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

DOCTORAL THESIS

Jaroslav Horáček

Interval linear and nonlinear systems

Department of Applied Mathematics

Supervisor of the doctoral thesis:	Doc. Mgr. Milan Hladík, Ph.D.
Study programme:	Informatics
Study branch:	Discrete Models and Algorithms

Prague 2019

Title: Interval linear and nonlinear systems

Author: RNDr. Jaroslav Horáček

Department: Department of Applied Mathematics

Supervisor: Doc. Mgr. Milan Hladík, Ph.D., Department of Applied Mathematics

Abstract: First, basic aspects of interval analysis, roles of intervals and their applications are addressed. Then, various classes of interval matrices are described and their relations are depicted. This material forms a prelude to the unifying theme of the rest of the work – solving interval linear systems.

Several methods for enclosing the solution set of square and overdetermined interval linear systems are covered and compared. For square systems the new shaving method is introduced, for overdetermined systems the new subsquares approach is introduced. Detecting unsolvability and solvability of such systems is discussed and several polynomial conditions are compared. Two strongest conditions are proved to be equivalent under certain assumption. Solving of interval linear systems is used to approach other problems in the rest of the work.

Computing enclosures of determinants of interval matrices is addressed. NP-hardness of both relative and absolute approximation is proved. New method based on solving square interval linear systems and Cramer's rule is designed. Various classes of matrices with polynomially computable bounds on determinant are characterized. Solving of interval linear systems is also used to compute the least squares linear and nonlinear interval regression. It is then applied to real medical pulmonary testing data producing several potentially clinically significant hypotheses. A part of the application is a description of the new breath detection algorithm. Regarding nonlinear systems an approach to linearizing a constraint satisfaction on an interval box problem into a system of real inequalities is shown. Such an approach is a generalization of the previous work by Araya, Trombettoni and Neveu. The features of this approach are discussed.

At the end computational complexity of selected interval problems is addressed and their feasible subclasses are captured. The interval toolbox LIME for Octave and its interval package, which implements most of the tested methods, is introduced.

Keywords: interval matrix, interval linear system, interval linear algebra, constraint satisfaction problem, interval regression, interval determinant, computational complexity

First of all, I would like to express my deep gratitude to Milan Hladík. I am very grateful for his support, for many opportunities he introduced me to, for never refusing to help me, for his patience and human kindness. I also deeply thank to my scientific colleagues and friends. I enjoyed working with you very much. Big thanks also goes to the students that I met during many lectures and tutorials I gave and also the students I supervised. Even though, I was the teacher, they have taught me many amazing things. I am very grateful to my friends for adding always something new to the way I view the world. A huge thanks goes to my parents and brother Jan, their influence has led me to the point where I stand now. Finally, I would like to thank my partner for her endless support, care and love.

Contents

1 Introduction

"To develop a complete mind: Study the art of science; study the science of art. Learn how to see. Realize that everything connects to everything else."

A quote attributed to Leonardo DaVinci.

In applications, many problems can be transformed to solving a system of linear equations. That is why linear systems often play a prominent role. There are various reasons why to incorporate intervals into such systems (rounding errors, inaccuracy of measurement, uncertainty, etc.). Similarly, in this work the main theme that weaves through all the chapters are interval linear systems. Of course, during the work we also meet nonlinear problems. However, it will be possible to deal with them using the linear means.

The first goal of this work is to present our contributions to several areas of interval analysis. Moreover, the work is submitted as a doctoral thesis of the author.

Most chapters are based on reworked and extended journal or conference papers published with other co-authors; mainly Michal Černý, Milan Hladík, Jan Horáček and Václav Koucký. Some of the results were also a product of joint work with defended students that were supervised by the author of this work; namely Josef Matějka and Petra Pelikánová. Some chapters contain also unpublished results and new material.

Parts of the text keep a survey book style with links to other works to enable the reader (either a professional or a student) to quickly pick up basics of the addressed area. That is the third goal of this work.

The material of this work is built in a cumulative way, hence a chapter usually uses the material from the previous chapters. However, we believe that each chapter could be read in a stand-alone manner with only occasional turning of pages. Most of the chapters are concluded with references to broader literature on various topics.

The work is divided into 12 chapters. Below is the brief content of each chapter:

Chapter 2. Roles of intervals. We introduce our understanding of intervals and their roles • We show simple examples comprehensible without knowledge of interval analysis • We discuss properties and advantages of intervals • The literature concerning applications and various ares of interval analysis referenced.

Chapter 3. Basic notation and ideas. Basic noninterval notation is introduced • Interval notation, concepts and structures we use are introduced • We briefly discuss the relation between intervals and rounded arithmetics • We discuss testing of interval methods and how to compare interval results.

Chapter 4. Interval matrix ZOO. Known classes of interval matrices, their properties and examples are presented • Their relations are depicted.

Chapter 5. Square interval linear systems. Various known direct and iterative methods are discussed • We discuss related topics such as preconditioning, finding initial enclosures and stopping criteria • We introduce the shaving method that enables further improvement of an enclosure • The methods are briefly compared.

Chapter 6. Overdetermined interval linear systems. The least squares solution is discussed • Various known methods for solving square interval linear systems are adapted for solving overdetermined interval systems • We introduce some known methods for solving overdetermined systems. • We introduce the subsquares approach and its variants.

Chapter 7. (Un)solvability of interval linear systems. Various conditions for detecting unsolvability are introduced • Checking full column rank is discussed and two sufficient conditions are proved to be equivalent under certain assumption • Checking solvability is addressed • The mentioned methods are compared.

Chapter 8. Determinant of an interval matrix. Known results about interval determinants are addressed • NP-hardness of absolute and relative approximation is proved • Known methods are refined to compute determinants of interval matrices • A method based on Cramer's rule is designed • Determinants of symmetric matrices are addressed • Classes of matrices with polynomially computable tasks related to interval determinants are explored • The methods are tested.

Chapter 9. Application of intervals to medical data. The Multiple breath washout procedure for lung function testing is introduced • Our algorithm for finding breath ends is introduced • Special type of regression where matrix is integer is discussed • Interval regression is applied to clinical data • Hypothetical conclusions are derived from the results.

Chapter 10. A linear approach to CSP. Linearization of nonlinear constraints is discussed • Linear programming approach is introduced • Vertex selection for linearization is discussed • Nonvertex selection for linearization is discussed • Properties of the proposed linearization are analyzed.

Chapter 11. Complexity of selected interval problems. Computational complexity in relation to intervals is explained • Complexity of various problems is addressed • Polynomially computable cases or classes of problems are characterized.

Chapter 12. LIME^2: interval toolbox. Interval toolbox LIME is introduced • Properties and goals of LIME are specified • Features and methods of LIME are listed • Installation and use is described.

1.1 Main results of the work

Here, we briefly summarize the main results of the work:

- **Chapter** 4. In this chapter we restructure results by Neumaier and others. New examples are added and relations between classes of interval matrices are analyzed and clearly visualized.
- **Chapter** 5. Many methods for solving interval linear systems need to use preconditioning. However, such an operation typically enlarges the original solution set. Methods applied to a preconditioned system return an enclosure of the enlarged solution set. In such cases a method that can further improve such an enclosure is of high importance. We introduce the shaving method that takes an enclosure and iteratively tries to shave off slices of the enclosure to get closer to the original solution set.
- **Chapter** 6. We shed more light on known methods for solving overdetermined interval linear systems. For overdetermined systems of interval linear equations we designed a new subsquares approach and its variants that can be easily parallelized and most of all can detect unsolvability of the system.
- **Chapter** 7. We describe several conditions for checking unsolvability and solvability of interval linear systems. Two conditions for detecting unsolvability that are based on full column rank detection are proved to be equivalent under certain assumption. Range of application of all conditions is visualized using heat maps.
- **Chapter** 8. In this chapter we prove that computing both the relative and absolute approximation of the exact determinant of an interval matrix is NP-hard. We characterize several classes of matrices with polynomially computable bounds on interval determinant. We design a new faster algorithm, based on Cramer's rule, for computing enclosure of the determinant of an interval matrix.
- **Chapter** 9. The interval least squares regression is applied to real world medical data from lung function testing. We show how to improve computation speed for certain regression input. We developed a new algorithm for detecting breath ends in clinical data. Such an algorithm can outperform the state-of-the-art algorithms even a commercial one. Based on the results we derive several hypotheses. If they turn to be true, it would have a significant impact on the area of current lung assessment methods.
- **Chapter** 10. Nonlinear constraint satisfaction problems are part of many practical problems. In this chapter we show how to linearize the nonlinear constraints and solve them using linear programming. For such a linearization an expansion

point is needed. Older approaches used vertex points of the initial box, we show how to use an arbitrary point from the box. We prove that such a linearization is never worse then Jaulin's bounding with two parallel affine functions.

- **Chapter** 11. We discuss the complexity issues related to interval linear algebra. Then we provide a concise survey of complexity of selected problems from interval linear algebra.
- **Chapter** 12. We briefly introduce LIME, our interval package for Octave. Such package contains most of the methods mentioned in this work and some more.

For the list of author's publications and defended students see Chapter 13.

2 Roles of intervals

- Various roles of intervals
- Early use of intervals
- Properties and advantages of intervals
- Literature and sources on intervals

In this chapter various roles of intervals are demonstrated. They are introduced via examples that do not require a proper definition of an interval arithmetics yet. Later, useful properties and advantages of intervals are pointed out. We slightly mention the early works concerning intervals. The chapter is concluded with references to applications and other aspects of intervals.

2.1 Examples of intervals

Let us start with six simple examples. They illustrate various roles intervals can play.

Example 2.1. One of the earliest works on intervals was probably by Archimedes (287–212 BC). In his treatise Measurement of a Circle he gave the following verified bounds for π

$$3\frac{10}{71} < \pi < 3\frac{1}{7}.$$

Note that he proved that π indeed lies in the given bounds.

Example 2.2. Let us say, we want to know what time t it will take an object (simulated as a mass point) to fall from height $h = 50$ meters. If we take the h as a constant, then time t can be simply expressed as a function of a gravitational acceleration g as

$$t(g) = \sqrt{\frac{2h}{g}} = \frac{10}{\sqrt{g}} \text{ seconds.}$$

However, gravitational acceleration differs at various places on Earth as elaborated in the work [66]. The lowest estimated value $\underline{g}$ is on the Nevado Huascarán summit, Peru and the highest value $\overline{g}$ is on the surface of the Arctic Ocean

$$\underline{g} = 9.76392 \; ms^{-2} \quad \overline{g} = 9.83366 \; ms^{-2}.$$

If we do not know the exact g of our area we should simultaneously evaluate the formula for all g's of all measured surface points. However, since the function t is decreasing in g it is enough to evaluate it for $\overline{g}$ to obtain the shortest time and for $\underline{g}$ to obtain the longest time. Hence, when not computing with a specific g the time lies in the interval

$$[t(\overline{g}), t(\underline{g})] = [3.1889\ldots, 3.2002\ldots].$$

To make the bounds to safely contain the value of the time t we can say

$$t \in [3.1889, 3.2003] \text{ seconds.}$$

Example 2.3. Let us take the continuous function

$$f(x) = x^3 - 10x^2 + 27x - 18,$$

and let us inspect the existence of a root on the interval $[2, 5]$. The function f is continuous on $[2, 5]$, hence the intermediate value theorem states that f takes any value between $f(2)$ and $f(5)$ on this interval. As $f(2) = 4$ and $f(5) = -8$ the function f must take zero for some point in $[2, 5]$. Therefore, $[2, 5]$ is a verified interval containing a zero of the function f.

We can go further and use bisection – splitting the initial interval into halves and applying the intermediate value theorem on the two halves separately. If the function values at the endpoints of one half do not have different signs, then we go on to inspect the other half. The procedure can be recursively repeated. Here is the list of examined intervals safely containing a root.

$$\begin{aligned}
&[2, 5]\\
&[2, 3.5]\\
&[2.75, 3.5]\\
&[2.75, 3.125]\\
&[2.9375, 3.125]\\
&[2.9375, 3.03125]
\end{aligned}$$

If we properly handle the rounding errors the intervals introduce verified bounds on the location where the root lies. With each step the width of a resulting enclosure decreases. Since $f(x) = (x - 1)(x - 3)(x - 6)$ we know that the exact root is 3. Such a method is very simple and can be further improved.

Example 2.4. A patient is connected to a breathing mask and instructed to breathe normally. During the breathing session various physical characteristics are measured by sensors in the mask. One of the variables measured is actual flow of air inside the mask. The sensor measures the flow value every given time slice. Let the length of a time-slice be d (usually $d = 5$ms). Moreover, the flow sensor has accuracy 5%. Hence, each measured flow in each time slice t denoted as φ_t becomes an interval

$$[0.95 \cdot \varphi_t,\ 1.05 \cdot \varphi_t].$$

Instead of a sequence of real numbers we get a sequence of intervals as depicted in Figure 2.1.

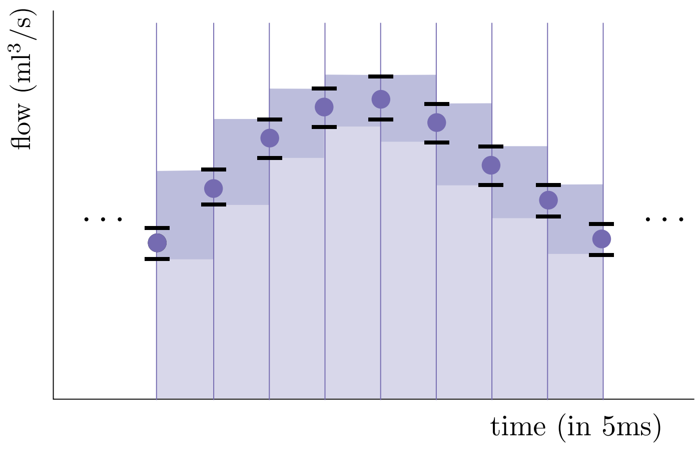


Figure 2.1: Simple verified volume computation. The circles represent flow measured in each time slice (vertical bars), short horizontal bars depict upper and lower bound on each measured flow incorporating 5 % measurement accuracy. Darker and lighter area depict the upper and lower bound on the volume respectively.

Many approaches to clinical assessment of lung function require the knowledge of the total volume of inhaled/exhaled air. Volume can be obtained as integration of flow – computing the area of the surface under the flow data. Since the time slice is small enough, the bounds for the volume can be computed as

$$\left[d \cdot \sum_{i=1}^{n-1} 0.95 \cdot \min(\varphi_i, \varphi_{i+1}),\ d \cdot \sum_{i=1}^{n-1} 1.05 \cdot \max(\varphi_i, \varphi_{i+1})\right].$$

Such an approach can be used for other integration applications. The example is based on the real medical background later explained in Chapter 9. It is a philosophical question whether these bounds are verified (whether the measurement accuracy covers all phenomena that can occur). However, it is a safe way of using the measured data.

Example 2.5. Let us take the function f from the previous example. We want to inspect whether it is increasing on the interval $[5, 5.9]$. Since the first derivative of $f(x)$ is

$$f'(x) = 3x^2 - 20x + 27,$$

which is greater than 0 on $[5, 5.9]$. Thus f is increasing on this interval.

Example 2.6. Let us have one nonlinear constraint

$$x^2 - \cos(y) = 0,$$

where $x \in [-1, 1]$ and $y \in [-1, 1]$. Let us bound the feasible solutions of the constraint. The initial bounds on x and y can be further reduced.

For $y \in [-1, 1]$ the range of the function cos is included in $[0.54, 1]$. The maximum value is $\cos(0) = 1$ and the minimum value is $\cos(-1) = \cos(1) = 0.540302 \cdots > 0.54$.

Now, by expressing x as $|x| = \sqrt{\cos(y)}$ for $\cos(y) \in [0.54, 1]$ we get from monotonicity of $\sqrt{\cdot}$ that $|x| \in [0.73, 1]$, i.e,

$$x \in [-1, -0.73] \cup [0.73, 1].$$

Note that we actually proved that no solution has x, in the interval $[-0.7, 0.7]$,

In the above mentioned examples an interval played the following four roles:

1. interval in which a phenomenon occurs everywhere (Example 2.5),
2. interval in which a phenomenon occurs for sure, but we cannot tell where exactly (Example 2.1 and 2.3),
3. interval in which a phenomenon might occur (Example 2.2, 2.4 and 2.6),
4. interval in which a phenomenon does not occur (Example 2.6).

Such a perception of intervals is nothing new, we as people do it every day. The 1. is used when speaking of interval training (a form of training requiring to keep doing an exercise for a given period of time), a training when during an interval one must keep doing a prescribed activity followed by a short break. We use the 2. when watching Perseids or eclipse of the sun in the sky (these phenomenons have a known interval in which they occur). We use the 3. when placing a bet on a goal during a given period of a game. The 4. is used when referring to an amount of time between meals, a gap between objects, a break between two halves of a match.

In this work we are going to exploit these roles of intervals in various ways.

2.2 Application of intervals

Except from using the intervals in the way explained in the first section. The intervals can be, more specifically, used for various purposes:

- **To handle rounding errors.** By proper outward rounding of intermediate calculation results a verified interval containing the proper desired values can be obtained.
- **To express uncertainty.** In some situations we are not sure about the proper distribution of a phenomenon. Note that the situation is a bit different from having a uniform distribution on the interval. By uniform distribution we model the situation on an interval, however, in reality the obtained value can come from outside of the interval. Nevertheless, in the case of intervals the lower and upper bounds are verified to keep the value in between.
- **To cover measurement errors.** Machines have usually given operating accuracy in a form of ±error which produces interval bounds.
- **To proof a property for all representatives.** For example, in a dynamical system it is possible to prove that all points starting from a given initial area will reach an equilibrium.

Intervals can be used everywhere where the problems evince the kind of uncertainty already described – computer assisted proofs, economics, medicine, solving numerical systems and differential equations, constraint satisfaction problems and global optimization, computing physical constants, robotics, etc. It would be redundant to list all the possible applications, since it has been done many times. More uses of intervals can be found in [104, 132]. For more applications see, e.g., [5], [104]. The applications of intervals lies on many foundations.

2.3 Early works on intervals

We have already mentioned Archimedes and his approach to enclosing π with intervals. If we fast-forward to 20th century we encounter the following names in relation to intervals:

- **1931** – Rosalind Cecily Young published her paper *The algebra of multi-valued quantities* [222].
- **1951** – Paul Sumner Dweyer in his book *Linear computations* discusses range numbers and their use to measure rounding errors [36].
- **1956** – Mieczyslaw Warmus in his paper *Calculus of approximations* builds an interval apparatus for formulation of numerical problems [219].
- **1958** – Teruo Sunaga in his paper *Theory of interval algebra and its application to numerical analysis* develops interval calculus and shows its properties and examples in order to solve problems [209].
- **1961** – Ramon E. Moore published his Ph.D. thesis *Interval arithmetic and automatic error analysis in digital computing* [130].

This list is just to give a reader a brief peek into the historical connections of interval analysis. We are aware that this list is possibly very incomplete. History related to interval arithmetics is an interesting subject and would need much more space than we can afford here. More information regarding history of intervals can be found in [5, 196].[1]

Although the intervals were known early in 20th century it took some time before they were used in computers. There were possibly two reasons: interval operations were considered too slow in comparison with their real counterparts and the resulting intervals were huge. However, this comparison with real numbers was a bit unfair because interval computations solve a different problem – instead of "some" solution of unknown quality interval arithmetics gives us rigorous bounds for the solution. Regarding the widths of intervals, with the successive developments of new methods the resulting intervals have started to be of applicable quality.

[1]Many early papers on intervals are accessible at `http://www.cs.utep.edu/interval-comp/early.html` (Accessed February 10, 2019).

2.4 More on intervals

There are a lot of works to start with for better knowledge of intervals. A very short introduction is, e.g, [215] by Tucker or [104] by Kearfott. A classical book on introduction to interval analysis is [133] by Moore, Kearfott and Cloud. Another Moore's book on more mathematical applications of interval analysis is [132]. It contains large list of interval-related publications. Many key concepts are shown in another classical books – [3] by Alefeld and Herzberger and [139] by Neumaier. A book with applications mostly in robotics and control is [99] by Jaulin and et al. There is a work on verified numerics by Rump [193]. Regarding global optimization there is a book [59] by Hansen and Walster. A list of interval related publications is [51, 52]. All problems can be viewed from the computational complexity point of view. There is a thorough book [111] or one can read our survey paper on computational complexity and interval linear algebra [85]. Also Rohn's hanbook [176] can serve as a useful signpost to other interval topics. For introduction to computer (interval) arithmetic see, e.g., [115] or the IEEE interval standard [162].

3 Basic notation and ideas

- ▶ Basic notation
- ▶ Basic interval notation, arithmetics, operations and relations
- ▶ Interval structures, expressions and functions
- ▶ Intervals and rounding
- ▶ Comparison of interval structures
- ▶ Testing of interval algorithms

This is a preliminary chapter containing the elementary building blocks for this work. We start with the basic notation for real mathematical objects. Then we move towards interval related material. We briefly introduce interval arithmetics and other operations on intervals. Later, we explain how to work with more complex interval structures – vectors, matrices, expressions and functions. Relation of intervals and computer rounded arithmetics is discussed. Because in almost every subsequent chapter we compare various algorithms with interval outputs, we explain how to compare quality of interval results here. We also state what software and computational power we use for such testing..

3.1 Notation

For the sake of clarity we provide a list of notation that we are going to use for real structures:

notation	**explanation**
A	a real matrix
x, b	a real column vector
I or I_n	identity matrix of the corresponding size
e_i	ith column of I
E	all-ones matrix of the corresponding size

notation	explanation
A_{ij}	the coefficient in ith row and jth column of a matrix A
a_{ij} or $a_{(i,j)}$	the coefficient in ith row and jth column of a matrix A
A_{i*}	ith row of a matrix A
A_{*j}	ith column of a matrix A
$A_{1,(1:3)}$	a vector (a_{11}, a_{12}, a_{13}) (notation borrowed from Matlab)
A^T	A transposed
$\lvert \cdot \rvert$	absolute value (for vectors and matrices works element-wise)
$\lVert \cdot \rVert_p$	vector or matrix p-norm
A^+	the Moore-Penrose pseudoinverse of A
A^{-1}	inverse matrix
A^{-T}	inverse of A^T
$\varrho(A)$	spectral radius of A
S^n	a set of all vectors of length n with coefficients from the set S
Y_n	the set $\{\pm 1\}^n$

For every vector $x \in \mathbb{R}^n$ we define its sign vector $\operatorname{sign}(x) \in \{\pm 1\}^n$ as

$$\operatorname{sign}(x)_i = \begin{cases} 1, & \text{if } x_i \geq 0, \\ 0, & \text{if } x_i = 0. \\ -1, & \text{if } x_i < 0. \end{cases}$$

Functions max, min applied on a vector are understand in a similar way as in Matlab, they choose maximum/minimum of the vector coefficients. For a given vector $x \in \mathbb{R}^n$ we denote

$$D_x = \operatorname{diag}(x_1, \ldots, x_n) = \begin{pmatrix} x_1 & 0 & \ldots & 0 \\ 0 & x_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & x_n \end{pmatrix}.$$

By writing $|x - y| < \epsilon$ for two vectors x, y of the same length we mean $|x_i - y_i| < \epsilon$ for each i. Hence, when relation operators such as $>, <, \leq, \geq, =$ are applied to vectors or matrices, then, unless not stated otherwise, they are understood component-wise.

3.2 Interval

The key notion of this work is an *interval*. Even though, there are various types of intervals, here we understand it as a synonym for a real closed interval.

Definition 3.1 (Interval)**.** For $\underline{a}, \overline{a} \in \mathbb{R}$ a real closed interval $\boldsymbol{a}$ is defined as

$$\boldsymbol{a} = [\underline{a}, \overline{a}] = \{a \in \mathbb{R} \mid \underline{a} \leq a \leq \overline{a}\},$$

$\underline{a}, \overline{a}$ are called the lower and upper bound respectively.

If it holds that $\underline{a} = \overline{a}$, then we call the interval *degenerate.* If $\underline{a} = -\overline{a}$, then we call the interval *symmetric.* We denote the set of all real closed intervals by $\mathbb{IR}$. Open intervals will be only rarely needed and their use will be explicitly announced. They will be typeset with parentheses (i.e., (a, b)).

An interval can be also defined using a center and a distance from this center.

Definition 3.2 (Interval 2)**.** For $a_c \in \mathbb{R}$ and positive $a_\Delta \in \mathbb{R}$ a real closed interval $\boldsymbol{a}$ can be also defined as

$$\boldsymbol{a} = [a_c - a_\Delta, a_c + a_\Delta],$$

a_c and a_Δ are called the *midpoint* and *radius* respectively.

Sometimes it simplifies the notation to move the subscripts to the top, i.e., a^c, a^Δ, especially when other subscripts are used. We use this notation interchangeably. To be concise, when speaking about an interval $\boldsymbol{a}$ we implicitly assume that a_c, a_Δ are respectively its midpoint and radius.

Even though, the two definitions are obviously equivalent, using a proper definition may save excessive notation. Intervals and derived interval structures are denoted in boldface (i.e., $\boldsymbol{x}, \boldsymbol{A}, \boldsymbol{b}, \boldsymbol{f}$). Real numbers, vectors, matrices, functions, etc., are typeset in normal font (i.e., x, A, b, f).

3.3 Set operations

Intervals can be viewed as sets and therefore the typical set operations can be defined for them.

Definition 3.3 (Set operations)**.** Let us have two intervals $\boldsymbol{a} = [\underline{a}, \overline{a}]$ and $\boldsymbol{b} = [\underline{b}, \overline{b}]$. Then $\boldsymbol{a} \cap \boldsymbol{b} = \emptyset$ if $\overline{a} < \underline{b}$ or $\overline{b} < \underline{a}$. Otherwise

$$\begin{aligned} \boldsymbol{a} \cap \boldsymbol{b} &= [\max(\underline{a}, \underline{b}),\ \min(\overline{a}, \overline{b})], \\ \boldsymbol{a} \cup \boldsymbol{b} &= \{x \in \boldsymbol{a} \vee x \in \boldsymbol{b}\}. \end{aligned}$$

Since the result of the operation $\cup$ is not always a single interval we define the hull as

$$\square(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a} \sqcup \boldsymbol{b} = [\min(\underline{a}, \underline{b}),\ \max(\overline{a}, \overline{b})].$$

Note that for the hull we use two different notations, that can be interchanged. Generally, the hull is understood as the interval of the minimal width containing the sets $\boldsymbol{a}$ and $\boldsymbol{b}$. The set operations can be easily extended to take more intervals as arguments.

3.4 Interval arithmetics

An arithmetics can be defined on intervals. We are going to use a standard definition mentioned in, e.g, [133]. What we need from an arithmetical operation $\circ$ on two intervals $\boldsymbol{a}, \boldsymbol{b}$ is

$$\boldsymbol{a} \circ \boldsymbol{b} = \square\{a \circ b \mid a \in \boldsymbol{a}, b \in \boldsymbol{b}\}.$$

The following definition of the basic operations satisfies such a demand.

Definition 3.4 (Interval arithmetics)**.** Let us have two intervals $\boldsymbol{a} = [\underline{a}, \overline{a}]$ and $\boldsymbol{b} = [\underline{b}, \overline{b}]$. Arithmetical operations $+, -, \cdot, /$ are defined as

$$\begin{aligned}
\boldsymbol{a} + \boldsymbol{b} &= [\underline{a} + \underline{b}, \overline{a} + \overline{b}],\\
\boldsymbol{a} - \boldsymbol{b} &= [\underline{a} - \overline{b}, \overline{a} - \underline{b}],\\
\boldsymbol{a} \cdot \boldsymbol{b} &= [\min(M), \max(M)], \quad \text{where } M = \{\underline{a}\underline{b},\ \underline{a}\overline{b},\ \overline{a}\underline{b},\ \overline{a}\overline{b}\},\\
\boldsymbol{a}/\boldsymbol{b} &= \boldsymbol{a} \cdot (1/\boldsymbol{b}), \quad \text{where } 1/\boldsymbol{b} = [1/\overline{b}, 1/\underline{b}],\ 0 \notin \boldsymbol{b}.
\end{aligned}$$

In the definition of division we presume $\boldsymbol{b}$ does not contain 0. When we need to divide with intervals containing zero, an extended version of interval arithmetics can be used [114, 155].

The set $\mathbb{IR}$ with the defined interval arithmetics does not form a field. Only some properties of a field hold. There exist distinct zero element $\mathbf{0} = [0, 0]$ and unit element $\mathbf{1} = [1, 1]$ (we will denote them just 0 and 1 respectively). Moreover, for all $\boldsymbol{a} \in \mathbb{IR}$ it holds that

$$\begin{aligned}
0 + \boldsymbol{a} &= \boldsymbol{a},\\
1 \cdot \boldsymbol{a} &= \boldsymbol{a},\\
0 \cdot \boldsymbol{a} &= 0.
\end{aligned}$$

By definition, the addition and multiplication are commutative and associative.

$$\begin{aligned}
\boldsymbol{x} + \boldsymbol{y} &= \boldsymbol{y} + \boldsymbol{x}, & \boldsymbol{x} + (\boldsymbol{y} + \boldsymbol{z}) &= (\boldsymbol{x} + \boldsymbol{y}) + \boldsymbol{z},\\
\boldsymbol{x}\boldsymbol{y} &= \boldsymbol{y}\boldsymbol{x}, & \boldsymbol{x}(\boldsymbol{y}\boldsymbol{z}) &= (\boldsymbol{x}\boldsymbol{y})\boldsymbol{z}.
\end{aligned}$$

Unfortunately, there is no inverse element with respect to addition and multiplication.

Proposition 3.5. *For a nondegenerate interval $\boldsymbol{a} = [\underline{a}, \overline{a}]$ there does not exist an inverse element with respect to addition.*

Proof. Let $\boldsymbol{a}$ be a nondegenerate interval and let $\boldsymbol{b}$ be its inverse element with respect to addition. According to the definition of the zero interval we get

$$\mathbf{0} = [0, 0] = \boldsymbol{a} + \boldsymbol{b} = [\underline{a}, \overline{a}] + [\underline{b}, \overline{b}] = [\underline{a} + \underline{b}, \overline{a} + \overline{b}].$$

Thus $\underline{a} + \underline{b} = 0$ and $\overline{a} + \overline{b} = 0$. It follows that

$$\underline{b} = -\underline{a},\ \overline{b} = -\overline{a}.$$

Hence

$$\boldsymbol{b} = [-\underline{a}, -\overline{a}].$$

For the bounds of the interval $\boldsymbol{a}$ it holds that $\underline{a} \leq \overline{a}$. However, the bounds of $\boldsymbol{b}$ are contradiction to the definition of interval since $-\underline{a} \geq -\overline{a}$. □

According to the definition of an interval the inverse element with respect to addition exists only for a degenerate interval. The proof for nonexistence of inverse element with respect to multiplication can be provided similarly, but requires more tedious elaboration.

Moreover, the distributivity does not hold either. Generally,

$$\boldsymbol{a}(\boldsymbol{b} + \boldsymbol{c}) \neq \boldsymbol{ab} + \boldsymbol{ac}.$$

Example 3.6. For intervals $\boldsymbol{a} = [1, 2], \boldsymbol{b} = [1, 1]$ and $\boldsymbol{c} = [-1, -1]$ we obtain the following results.

$$\begin{aligned} \boldsymbol{a}(\boldsymbol{b} + \boldsymbol{c}) &= [0, 0], \\ \boldsymbol{ab} + \boldsymbol{ac} &= [-1, 1]. \end{aligned}$$

However, the *subdistributivity* always holds

$$\boldsymbol{a}(\boldsymbol{b} + \boldsymbol{c}) \subseteq \boldsymbol{ab} + \boldsymbol{ac}.$$

Such an overestimation caused by the second formula is a result of a so-called *dependency problem.* Whenever real number is chosen from $\boldsymbol{a}$ the same value should be fixed for the second occurrence of the second $\boldsymbol{a}$. However, the interval arithmetics does not see both $\boldsymbol{a}$'s as one and the same variable, but rather as two different variables. We will touch dependency in Section 3.8 in more detail.

3.5 Relations

Basic relations of intervals can be defined in the following way.

Definition 3.7 (Relations)**.** For two intervals $\boldsymbol{a} = [\underline{a}, \overline{a}]$ and $\boldsymbol{b} = [\underline{b}, \overline{b}]$.
The relation $\boldsymbol{a} = \boldsymbol{b}$ holds if

$$\underline{a} = \underline{b} \quad \text{and} \quad \overline{a} = \overline{b}.$$

The relation $\boldsymbol{a} \leq \boldsymbol{b}$ holds if

$$\overline{a} \leq \underline{b}.$$

The relation $\boldsymbol{a} < \boldsymbol{b}$ holds if

$$\overline{a} < \underline{b}.$$

Similarly for the relations $\geq, <, >$.

Note, that some intervals are incomparable, e.g.,

$$[1, 3] \not\leq [2, 4] \quad \text{and} \quad [1, 3] \not\geq [2, 4].$$

3.6 More interval notation

Regarding intervals we need to define more notation:

notion	**formula**	**explanation**
$\mathrm{wid}(\boldsymbol{a})$	$\overline{a} - \underline{a}$	width of an interval
$\mathrm{mid}(\boldsymbol{a})$	$a_c = (\underline{a} + \overline{a})/2$	midpoint of an interval
$\mathrm{rad}(\boldsymbol{a})$	$\mathrm{wid}(\boldsymbol{a})/2$	radius of an interval
$\mathrm{mig}(\boldsymbol{a})$	$\min(\lvert\underline{a}\rvert, \lvert\overline{a}\rvert)$ or 0 when $0 \in \boldsymbol{a}$	mignitude of an interval
$\mathrm{mag}(\boldsymbol{a})$	$\max(\lvert\underline{a}\rvert, \lvert\overline{a}\rvert)$	magnitude of an interval
$\lvert\boldsymbol{a}\rvert$	$\{\lvert a\rvert,\ a \in \boldsymbol{a}\}$	absolute values of an interval

Note the difference between the absolute value and magnitude. Sometimes these two notions are used interchangably. Nevertheless, here in our work, we are going to strictly distinguish between them. The magnitude of an interval is a number while the absolute value of an interval is an interval:

$$\begin{aligned}
\text{If } \boldsymbol{a} > 0 \quad & |\boldsymbol{a}| = [\underline{a}, \overline{a}]\,, \\
\text{If } \boldsymbol{a} < 0 \quad & |\boldsymbol{a}| = [|\overline{a}|, |\underline{a}|]\,, \\
\text{If } 0 \in \boldsymbol{a} \quad & |\boldsymbol{a}| = [0, \max\{|\underline{a}|, |\overline{a}|\}]\,.
\end{aligned}$$

For many important properties of the introduced functions and operations see [139].

3.7 Vectors and matrices

Intervals can be used as building blocks for more complex structures. In this section we address interval vectors and matrices. An interval matrix (or an interval vector as its special case) can be defined as a matrix having intervals as its coefficients.

Definition 3.8 (Interval vector and matrix)**.** Let $\boldsymbol{b}_i, \boldsymbol{a}_{ij}$ for $i = 1, \ldots, m$ and $j = 1, \ldots, n$ be intervals then an m-dimensional interval vector $\boldsymbol{b}$ and an $m \times n$ interval matrix $\boldsymbol{A}$ are defined as

$$\boldsymbol{b} = \begin{pmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \\ \vdots \\ \boldsymbol{b}_m \end{pmatrix}, \quad \boldsymbol{A} = \begin{pmatrix} \boldsymbol{a}_{11} & \ldots & \boldsymbol{a}_{1n} \\ \vdots & & \vdots \\ \boldsymbol{a}_{m1} & \ldots & \boldsymbol{a}_{mn} \end{pmatrix},$$

When we talk about *square* matrices, we always assume the size of $\boldsymbol{A}$ is $n \times n$. Otherwise, the size $m \times n$ is assumed. Note that an n-dimensional interval vector actually represents an n-dimensional box aligned with axes. That is why we use the phrases “interval vector” and “interval box” interchangeably.

The relations $=, \leq, \geq, <, >, \subseteq, \in$ are understood component-wise. So are the set operations $\cup, \cap, \square, \sqcup$. Hence an $m \times n$ interval matrix can be also defined using two real $m \times n$ matrices $\underline{A}, \overline{A}$ as

$$\boldsymbol{A} = \{A \mid \underline{A} \leq \overline{A}\}.$$

Formally, it is slightly different from Definition 3.8, however it is simple to transit between the two points of view. We can also define an interval matrix using its midpoint A_c and radius A_Δ matrix as

$$\boldsymbol{A} = [A_c - A_\Delta, A_c + A_\Delta].$$

For the sake of concise notation, when speaking about $\boldsymbol{A}$ we always implicitly assume that $\underline{A}, \overline{A}$ are its lower and upper bound respectively and that A_c, A_Δ are its midpoint and radius respectively.

For two interval matrices $\boldsymbol{A}, \boldsymbol{B}$ of the same size the interval arithmetics operations $+$ and $-$ are performed component-wise as

$$(\boldsymbol{A} + \boldsymbol{B})_{ij} = \boldsymbol{a}_{ij} + \boldsymbol{b}_{ij},$$
$$(\boldsymbol{A} - \boldsymbol{B})_{ij} = \boldsymbol{a}_{ij} - \boldsymbol{b}_{ij}.$$

For an $m \times n$ matrix $\boldsymbol{A}$ and an $n \times p$ matrix $\boldsymbol{B}$ the matrix multiplication $\boldsymbol{AB}$, can be carried out similarly as in the case of real matrices.

$$(\boldsymbol{AB})_{ij} = \sum_{k=1}^{n} \boldsymbol{a}_{ik}\boldsymbol{b}_{kj}.$$

Even though, the result gives sharp bounds on the matrix product, it can contain matrices that cannot be obtained by any product of $A \in \boldsymbol{A}, B \in \boldsymbol{B}$. Here is an example from [133].

Example 3.9. For two matrices

$$\boldsymbol{A} = \begin{pmatrix} [1,2] & [3,4] \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} [5,6] & [7,8] \\ [9,10] & [11,12] \end{pmatrix}$$

the product is $\boldsymbol{AB} = \begin{pmatrix} [32,52] & [40,64] \end{pmatrix}$. Let us take the matrix $\begin{pmatrix} 32 & 64 \end{pmatrix}$; the element 32 is obtained by multiplying $\underline{A}$ by lower bound of the right column of $\boldsymbol{B}$ and the element 64 is obtained by multiplying $\overline{A}$ by upper bound of the right column of $\boldsymbol{B}$.

The operation $+$ is for interval matrices commutative and associative. There are cases when associativity of multiplication multiplication fails [139]. As in the case of intervals, for matrices we again get subdistributivity [139].

$$\boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C}) \subseteq \boldsymbol{AB} + \boldsymbol{AC},$$
$$(\boldsymbol{A} + \boldsymbol{B})\boldsymbol{C} \subseteq \boldsymbol{AB} + \boldsymbol{AC}.$$

The already mentioned functions and operations wid$(\cdot)$, mid$(\cdot)$, rad$(\cdot)$, mig$(\cdot)$, mag$(\cdot)$ and $|\cdot|$ are for interval vectors and matrices understood component-wise. They posses several useful properties:

$$\begin{aligned} \operatorname{mag}(\boldsymbol{A}) &= |A_c| + A_\Delta, && (3.1)\\ \operatorname{mid}(\boldsymbol{A} \pm \boldsymbol{B}) &= A_c \pm B_c, && (3.2)\\ \operatorname{mid}(\boldsymbol{AB}) &= \operatorname{mid}(\boldsymbol{A})\operatorname{mid}(\boldsymbol{B}), \quad \text{if } \boldsymbol{A} \text{ or } \boldsymbol{B} \text{ is thin}, && (3.3)\\ \operatorname{rad}(\boldsymbol{AB}) &= |A|\operatorname{rad}(\boldsymbol{B}) \quad \text{if } \boldsymbol{A} \text{ is thin or } B_c = 0, && (3.4)\\ \operatorname{rad}(\boldsymbol{A} + \boldsymbol{B}) &= \operatorname{rad}(\boldsymbol{A}) + \operatorname{rad}(\boldsymbol{B}). && (3.5) \end{aligned}$$

Interval version of other notation such as $\varrho(\cdot)$ or $(\cdot)^{-1}$ will be introduced in the corresponding chapters later when needed. Next, we introduce the useful concept of interval matrix norms.

Definition 3.10 (Interval matrix norm)**.** For interval matrices a matrix norm $\|\cdot\|$ can be defined as

$$\|\boldsymbol{A}\| = \max\{\|A\|,\ A \in \boldsymbol{A}\}.$$

Regarding the computation of matrix norms, there are easily computable matrix norms:

$$\begin{aligned} \|\boldsymbol{A}\|_1 &= \max_j \sum_i \operatorname{mag}(\boldsymbol{A}_{ij}),\\ \|\boldsymbol{A}\|_\infty &= \max_i \sum_j \operatorname{mag}(\boldsymbol{A}_{ij}). \end{aligned}$$

Furthermore, we can use a so-called *scaled maximum norm* as a generalization of the maximum norm $\|\cdot\|_\infty$. For any vector $\boldsymbol{x} \in \mathbb{IR}^n$ and a vector $0 < u \in \mathbb{R}^n$ we define

$$\|\boldsymbol{x}\|_u := \max\{\operatorname{mag}(\boldsymbol{x}_i)/u_i \mid i = 1, \dots, n\},$$

and

$$\|\boldsymbol{A}\|_u := \|\operatorname{mag}(\boldsymbol{A})u\,\|_u.$$

Note that for $u = (1, \dots, 1)^T$ we get the maximum norm. The following holds for such a norm [139]

$$\begin{aligned} \|\boldsymbol{A}\|_u < \alpha &\iff \operatorname{mag}(\boldsymbol{A})u < \alpha u, && (3.6)\\ \|\boldsymbol{A}\|_u \le \alpha &\iff \operatorname{mag}(\boldsymbol{A})u \le \alpha u. && (3.7) \end{aligned}$$

In the further text, many of our results will be in terms of matrix norms. We will use only *consistent* matrix norms, i.e, those that satisfy

$$\|\boldsymbol{A} \cdot \boldsymbol{x}\| \le \|\boldsymbol{A}\| \cdot \|\boldsymbol{x}\|.$$

All the mentioned norms satisfy this property [126, 139]. Note all norms were defined for interval matrices. To define $\|\cdot\|_1, \|\cdot\|_\infty, \|\cdot\|_u$ for real matrices it is enough to replace $\operatorname{mag}(\cdot)$ with $|\cdot|$.

3.8 Interval expressions and functions

One of the important tasks is to enclose the range of a real-valued function. This section is loosely inspired by [139] and [215]. Let us consider a function $f : D \mapsto \mathbb{R}$, where $D \subseteq \mathbb{R}^n$, the range is then

$$f(D) = \{f(x) \mid x \in D\}.$$

For a monotone (or piece-wise monotone) function the range can be expressed exactly. Elementary functions such as $\cos(x), \sin(x), |x|, a^x, \log(x)$ satisfy this property. We can use these functions as building blocks for more complex functions.

Generally, we want to extend a real-valued function f to an interval function $\boldsymbol{f}$

$$\boldsymbol{f} : \mathbb{IR}^n \mapsto \mathbb{IR}.$$

Such a generalization should poses some favorable characteristics. First, it would be useful if

$$\boldsymbol{f}(x) = f(x), \ \forall x \in D.$$

Here, x can be seen as a degenerate interval vector. Or an interval function should at least satisfy

$$f(x) \in \boldsymbol{f}(\boldsymbol{x}), \ \text{for } x \in \boldsymbol{x} \subseteq \square D.$$

Such a function is called *interval extension.* Another favorable property is *inclusion monotonicity*, i.e.,

$$\boldsymbol{x} \subseteq \boldsymbol{y} \Rightarrow \boldsymbol{f}(\boldsymbol{x}) \subseteq \boldsymbol{f}(\boldsymbol{y}).$$

For a function a *natural interval extension* can be obtained by viewing its variables as intervals and its operators/subfunctions as interval operators/subfunctions. In [131] we can find the following theorem by Moore.

Theorem 3.11. *The natural interval extension $\boldsymbol{f}$ associated with a real function f that is a combination of only constants, variables, arithmetical operations and elementary functions (*$\sin(x), \cos(x), |x|, a^x, \log(x) \ldots$*) is an inclusion monotone interval extension such that*

$$\{f(x) \mid x \in \boldsymbol{x}\} \subseteq \boldsymbol{f}(\boldsymbol{x}),$$

for any $\boldsymbol{x}$, where $\boldsymbol{f}(\boldsymbol{x})$ is defined.

Note that there is a difference between $f(\boldsymbol{x})$ and $\boldsymbol{f}(\boldsymbol{x})$. The first denotes computation of range of a function over $\boldsymbol{x}$ and the second is an interval function. The following example demonstrates that not every interval extension is narrow.

Example 3.12. For $x \in \boldsymbol{x} = [-1, 2]$ compare the following:

$$\begin{aligned} \boldsymbol{x} - \boldsymbol{x} &= [-3, 1] \quad \text{vs.} \quad 0, \\ \boldsymbol{x} \cdot \boldsymbol{x} &= [-2, 4] \quad \text{vs.} \quad \boldsymbol{x}^2 = [0, 4]. \end{aligned}$$

The previous examples suffered from a so-called *dependency problem* – the interval arithmetic as is defined does not see the double occurrence of $\boldsymbol{x}$ and treats both occurrences as separate variables. We have actually met this phenomenon when talking about subdistributivity of interval arithmetic operations or interval matrix multiplication. Not surprisingly, we have the following theorem by Moore [131].

Theorem 3.13. *Let $f(x_1, \ldots, x_n, y_1, \ldots, y_m)$ be an real function from the previous theorem with $n + m$ variables and let $\boldsymbol{f}$ be its natural extension. Suppose that the variables $y_1, \ldots, y_m$ occur only once in f. Then*

$$\square\{f(x, y) \mid x \in \boldsymbol{x},\ y \in \boldsymbol{y}\} = \bigcup_{x \in \boldsymbol{x}} \boldsymbol{f}(x, \boldsymbol{y}),$$

for $(\boldsymbol{x}, \boldsymbol{y})$ where $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$ is defined.

Especially, if each variable in arithmetical expression occurs only once, then the following holds.

$$\boldsymbol{f}(\boldsymbol{x}) = \square\{f(x) \mid x \in \boldsymbol{x}\}.$$

There are many other methods for enclosing the range of $f(x)$ for $x \in \boldsymbol{x}$. One way is to use the mean value form

$$f(x) = f(x_c) + f'(\psi)(x - x_c),$$

where ψ lies on a line segment between x and x_c. Since $\psi \in \boldsymbol{x}$ we have the interval extension

$$\boldsymbol{f}(\boldsymbol{x}) \subseteq f(x_c) + f'(\boldsymbol{x})(\boldsymbol{x} - x_c).$$

The function f' is a gradient of f. Its range can be estimated in various ways e.g., using slopes [139]. We are going to use such methods in Chapter 10. For more on range of real-valued functions and polynomials see, e.g., [46, 134, 194, 196].

3.9 Rounded interval arithmetic

Now, we briefly touch the topic that we will address only rarely in the text. It is a well-known fact that computers cannot represent all numbers from $\mathbb{R}$. Let us denote the set of machine representable numbers by $\mathbb{R}^{pc}$. When a number cannot be represented it is necessary to round it to some representable number. Speaking of rounding procedure for a real number a, we are interested in the two main types: rounding to $+\infty$ ($\uparrow a$) and rounding to $-\infty$ ($\downarrow a$).

These roundings preserve the property of the relation $\leq$ on $\mathbb{R}$. Thus, for $a, b \in \mathbb{R}$

$$a \leq b \Rightarrow \uparrow a \leq \uparrow b.$$

Also for every $a \in \mathbb{R}$ it holds that

$$\uparrow\uparrow a = \uparrow a.$$

We have already defined the set $\mathbb{IR}$ as a set of real closed intervals. We can also define the set $\mathbb{IR}^{pc}$, the set of real closed intervals with machine representable endpoints. We can switch from $\mathbb{IR}$ to $\mathbb{IR}^{pc}$ with use of directed rounding:

$$[a, b] \in \mathbb{IR} \quad \mapsto \quad [\downarrow a, \uparrow b] \in \mathbb{IR}^{pc}.$$

Such an implementation needs switching of rounding mode. If a being machine representable implies $-a$ is also machine representable, then only one directed rounding is enough:

$$[\downarrow a, \uparrow b] = [\downarrow a, - \downarrow -b] = [- \uparrow -a, \uparrow b].$$

The interval arithmetics can be defined also on $\mathbb{IR}^{pc}$. For example addition of two intervals $\boldsymbol{a} = [\underline{a}, \overline{a}], \boldsymbol{b} = [\underline{b}, \overline{b}] \in \mathbb{IR}^{pc}$ can be defined as

$$\boldsymbol{a} +^{pc} \boldsymbol{b} = [\downarrow(\underline{a} + \underline{b}), \uparrow(\overline{a} + \overline{b})].$$

In the following text, we will keep working with $\mathbb{IR}$, however, we keep in mind that to obtain verified results algorithms must be implemented via $\mathbb{IR}^{pc}$. That is, when we talk about the hull or enclosure we implicitly assume that its end-points are machine representable.

There are many packages that handle computing with $\mathbb{IR}^{pc}$, e.g., Intlab for Matlab and Octave [188], Octave interval package [62], `libieee1788` for C++ [135] and many others [113]. However, not all of them conform to the interval arithmetics standard IEEE 1788-2015 [162]. More on rounding and interval arithmetics can be found in, e.g. [132, 133, 139].

3.10 Comparing quality of interval results

In this work we need to compare intervals or interval vectors (boxes) returned by various methods. If two methods `a` and `b` return single intervals $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively (e.g., methods for computing the determinant of an interval matrix), the returned solutions can be compared as

$$rat(\boldsymbol{a}, \boldsymbol{b}) = \frac{\operatorname{wid}(\boldsymbol{a})}{\operatorname{wid}(\boldsymbol{b})}. \tag{3.8}$$

If the two methods return n-dimensional interval vectors $\boldsymbol{a} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$ and $\boldsymbol{b} = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_n)$ respectively, their quality is compared as the average ratio of widths of the corresponding elements

$$\frac{\sum_{i=1}^{n} rat(\boldsymbol{a}_i, \boldsymbol{b}_i)}{n}. \tag{3.9}$$

Only rarely will it be compared as

$$\frac{\sum_{i=1}^{n} \operatorname{wid}(\boldsymbol{a}_i)}{\sum_{i=1}^{n} \operatorname{wid}(\boldsymbol{b}_i)}. \tag{3.10}$$

In each comparison we use a reference method, i.e., a method to which other methods are compared. In the text, the previous formulas are used in the following way. The method `b` (the second one) is always the reference method and `a` is the method compared to it. Hence, if the ratio is > 1, then the method `a` is worse than `b`, if the ratio is < 1, then the intervals returned by `a` are tighter than the ones by `b`.

3.11 How we test

Most of the chapters need to compare more methods for solving a certain interval problem. Features of each method can be demonstrated by special cases (particular interval matrix or system, etc.). Nevertheless, to compare methods more thoroughly, we test them on larger sets of random problems. Of course, the problems in real applications are not exactly random, however, in some cases the testing on random systems gives us a hint about the natural behavior of the methods.

If not stated otherwise, the tests are computed using the two settings:

1. DESKTOP – computationally demanding tests run on a desktop machine with 8-CPU machine Intel(R) Core(TM) i7-4790K, 4.00GHz, 15937 MB RAM, Octave 4.0.3., Octave Interval package 3.0.0.
2. LAPTOP – computationally not so exhaustive tests are executed on laptop with Intel Core i5-7200U – 2.5GHz, TB 3.1GHz, HyperThreading; 8GB DDR4 memory. Octave 4.2.2, Octave Interval package 3.2.0.

Most of the methods tested here are implemented in interval toolbox LIME (see more in Chapter 12), which is built on Oliver Heimlich's [62] interval package for Octave.

4 Interval matrix ZOO

- M-matrices and inverse nonnegative matrices
- Strictly diagonally dominant matrices
- H-matices
- Regular and strongly regular matrices
- Relations between matrix classes
- Other types of matrices

We defined general interval matrices in the previous chapter. However, there is a large variety of special types of interval matrices. Many of them emerged as generalization of notions from the classical linear algebra. Nevertheless, in this chapter we focus only on interval matrices. For more insight into real matrices see, for example, the works [19, 41, 91, 150].

Some of the classes of interval matrices have favorable properties (easily computable inverse, regularity etc.) and algorithms usually work well for them. We feel the necessity of characterizing the distinct classes of interval matrices, their features and links between them, since we believe it would increase the understanding of the rest of the work. This chapter is loosely based on Chapter 3 and 4 from Neumaier's book [139]. We focus on the most common types of matrices that are usually used in connection with quality of solving interval linear systems and related problems. In this short chapter we re-structure Neumaier's material and add some new examples and comments to make the relations between the classes of matrices more visible and clear. The final Figure 4.1 illustrates the relationships between the classes of interval matrices.

4.1 Regular matrices

Regular matrices are of special importance.

Definition 4.1 (Regular matrix)**.** A square interval matrix $\boldsymbol{A}$ is *regular* if every $A \in \boldsymbol{A}$ is nonsingular.

Note that there is a slight terminology confusion in addressing the similar quality of real and interval matrices.

real matrices	$\rightarrow$	nonsingular
interval matrices	$\rightarrow$	regular

The interval matrices that are not regular are called *singular* as in the case of real matrices.

In Chapter 11 we show that checking regularity is generally a coNP-complete problem. There exist a lot of sufficient and necessary conditions for regularity of interval matrices [179]. All of them are of exponential nature.

Fortunately, there are some polynomially computable sufficient conditions and not explicitly exponential algorithms for checking regularity [38, 96, 163, 164]. The following useful condition is from [164].

Theorem 4.2. *A square interval matrix* $\boldsymbol{A}$ *is regular if for some real matrix* R *the following condition holds*

$$\varrho(|I - RA_c| + |R|A_\Delta) < 1.$$

Particularly, if A_c *is regular, then for* $R = A_c^{-1}$ *the condition reads* $\varrho(|A_c^{-1}|A_\Delta) < 1$.

It can be shown that if the first condition holds for some R then

$$\varrho(|A_c^{-1}|A_\Delta) \leq \varrho(|I - RA_c| + |R|A_\Delta),$$

which makes the midpoint inverse a kind of optimal choice. Later we use the following simple consequence of Theorem 4.2.

Corollary 4.3. *A square interval matrix* $\boldsymbol{A}$ *with* $A_c = I$ *is regular if*

$$\varrho(A_\Delta) < 1.$$

4.2 M-matrices

In many applications matrices of a special form, called Z-matrices, appear [7, 18, 19, 39].

Definition 4.4 (Z-matrix)**.** A square real matrix A is called a *Z-matrix* if $a_{ij} \leq 0$ for every $i \neq j$. A square interval matrix $\boldsymbol{A}$ is called a *Z-matrix*, if every $A \in \boldsymbol{A}$ is a Z-matrix.

By adding more restriction to Z-matrices we obtain an important subclass of interval matrices.

Definition 4.5 (M-matrix)**.** An interval matrix $\boldsymbol{A}$ is an *M-matrix* if it is a Z-matrix and there exists $0 < u \in \mathbb{R}^n$ such that $\boldsymbol{A}u > 0$ (understood component-wise).

According to [150] the term “M-matrix” was first used by Ostrowski in [146] where he studied such matrices extensively. They are often connected to various problems in mathematics, biology, physics, etc. For more applications and properties of real M-matrices one can see [19, 41, 150]. Another feature of M-matrices is computational, since many algorithms behave well when working with an M-matrix.

Before stating the equivalent characterization of M-matrices, we need to specify what do we mean by an inverse interval matrix, a principal minor and a P-matrix.

Definition 4.6 (Inverse interval matrix)**.** Let us have a regular interval matrix $\boldsymbol{A}$. We define its interval inverse matrix $\boldsymbol{A}^{-1}$ as

$$\boldsymbol{A}^{-1} = [\underline{B}, \overline{B}],$$

$$\underline{B}_{ij} = \min\{(A^{-1})_{ij} \mid A \in \boldsymbol{A}\},$$
$$\overline{B}_{ij} = \max\{(A^{-1})_{ij} \mid A \in \boldsymbol{A}\},$$

for $i, j = 1, \dots, n$.

Definition 4.7 (Principal minor)**.** For a square matrix a *principal matrix* occurs when deleting some rows of the matrix and also the corresponding columns with the same indices. A determinant of a principal matrix is called a *principal minor.*

Definition 4.8 (P-matrix)**.** A square real matrix is a *P-matrix* if its every principal minor is positive. A square interval matrix $\boldsymbol{A}$ is a *P-matrix* if every $A \in \boldsymbol{A}$ is a P-matrix.

Theorem 4.9. *The following statements are equivalent*

1. $\boldsymbol{A}$ *is an M-matrix,*
2. *every* $A \in \boldsymbol{A}$ *is an M-matrix,*
3. $\underline{A}, \overline{A}$ *are M-matrices,*
4. $\boldsymbol{A}$ *is a regular Z-matrix and* $\boldsymbol{A}^{-1} = [\overline{A}^{-1}, \underline{A}^{-1}] \geq 0,$
5. $\boldsymbol{A}$ *is a Z-matrix and P-matrix.*

The statements *1.–4.* come from [139]. The statement *2.* implies that if $\boldsymbol{A}$ is an M-matrix and $\boldsymbol{B} \subseteq \boldsymbol{A}$, then $\boldsymbol{B}$ is also an M-matrix. From *4.* we can see that M-matrices are inverse nonnegative matrices (see the next section). The statement *5.* is a simple generalization of the similar claim for real matrices [150]. To check that a matrix is an M-matrix, the statement *4.* gives a hint how to find a positive vector u proving that $\boldsymbol{A}$ is an M-matrix. First, solve the system $\underline{A}u = e$. Because $\boldsymbol{A}^{-1}$ should be nonnegative, for the solution u it should hold that $u = \underline{A}^{-1}e > 0$. Second, check whether $\boldsymbol{A}u > 0$. The check needs to be performed in a verified way. It is also possible to exploit the statement *3.* If both $\underline{A}, \overline{A}$ are Z-matrices and their verified inverse is nonnegative, then $\boldsymbol{A}$ is an M-matrix. From *4.* it can be seen that M-matrices are regular. For a more detailed proof see e.g., [139] Regarding *5.*, computation of determinants of interval matrices can be used. For example, a tight enclosure of a determinant of a 2×2 interval matrix can be expressed as

$$\det \begin{pmatrix} \boldsymbol{a}_{11} & \boldsymbol{a}_{12} \\ \boldsymbol{a}_{21} & \boldsymbol{a}_{22} \end{pmatrix} = \boldsymbol{a}_{11} \cdot \boldsymbol{a}_{22} - \boldsymbol{a}_{12} \cdot \boldsymbol{a}_{21}.$$

This topic is further elaborated in Chapter 8.

Example 4.10. Let us have the matrix

$$\boldsymbol{A} = \begin{pmatrix} 2 & -1 \\ [-2, 0] & 2 \end{pmatrix}.$$

Clearly $\boldsymbol{A}$ is a Z-matrix. Furthermore, $\boldsymbol{A}$ is an M-matrix since all principal minors are positive $\det(\boldsymbol{A}_1) = 2, \det(\boldsymbol{A}_2) = 2, \det(\boldsymbol{A}_{12}) = [2, 4]$.

Example 4.11. Let us show another way to prove that the Z-matrix $\boldsymbol{A}$ from the previous example is an M-matrix. For

$$u = \begin{pmatrix} 1 \\ 1.5 \end{pmatrix} \text{ we see that } \boldsymbol{A}u = \begin{pmatrix} 0.5 \\ [1, 3] \end{pmatrix} > 0.$$

4.3 Inverse nonnegative matrices

From previous section it is already known that every M-matrix has a nonnegative inverse. M-matrices are part of a larger class of interval matrices.

Definition 4.12 (Inverse nonnegative)**.** A square interval matrix $\boldsymbol{A}$ is called *inverse nonnegative* if $\boldsymbol{A}$ is regular and $\boldsymbol{A}^{-1} \geq 0$.

For such a class of matrices there is a theorem by Kuttler in [117] which gives us explicit bounds on a matrix inverse.

Theorem 4.13 (Kuttler)**.** *Let $\boldsymbol{A}$ be an interval matrix. If its lower and upper bounds $\underline{A}, \overline{A}$ are regular and $\underline{A}^{-1}, \overline{A}^{-1} \geq 0$ then $\boldsymbol{A}$ is regular and*

$$\boldsymbol{A}^{-1} = [\overline{A}^{-1}, \underline{A}^{-1}] \geq 0.$$

Example 4.14. If we take the already known matrix

$$\boldsymbol{A} = \begin{pmatrix} 2 & -1 \\ [-2, 0] & 2 \end{pmatrix},$$

using the algebraic formula for inverse of a real 2×2 matrix we can inspect both inverses of $\underline{A}$ and $\overline{A}$

$$\underline{A}^{-1} = \begin{pmatrix} 1 & 0.5 \\ 1 & 1 \end{pmatrix}, \quad \overline{A}^{-1} = \begin{pmatrix} 0.5 & 0.25 \\ 0 & 0.5 \end{pmatrix},$$

and according to Kuttler's theorem we get

$$\boldsymbol{A}^{-1} = [\overline{A}^{-1}, \underline{A}^{-1}] = \begin{pmatrix} [0.5, 1] & [0.25, 0.5] \\ [0, 1] & [0.5, 1] \end{pmatrix},$$

which confirms that $\boldsymbol{A}$ is an inverse nonnegative matrix. Notice that $\boldsymbol{A}^{-1}$ is not regular since it contains, for example, the singular matrix

$$\begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}.$$

Example 4.15. According to Kuttler's theorem, the matrix

$$\boldsymbol{B} = \begin{pmatrix} -2 & 1 \\ [5, 6] & -2 \end{pmatrix}$$

has the inverse

$$\boldsymbol{B}^{-1} = \begin{pmatrix} [1, 2] & [0.5, 1] \\ [2.5, 5] & [1, 2] \end{pmatrix}.$$

which proves that $\boldsymbol{B}$ is inverse nonnegative, although it is not a Z-matrix (also not an M-matrix). Hence, not every inverse nonnegative matrix must be an M-matrix.

4.4 H-matrices

H-matrices are a generalization of M-matrices by lifting the condition on signs of matrix off-diagonal elements. The class of H-matrices inherits some favorable properties from M-matrices; regularity, for example (see [139]). We define an H-matrix using a comparison matrix.

Definition 4.16 (Comparison matrix)**.** For a square real matrix A its *comparison matrix* $\langle A \rangle$ is defined as

$$\begin{aligned} \langle A \rangle_{ii} &= A_{ii}, \\ \langle A \rangle_{ij} &= -|A_{ij}| \quad \text{for } i \neq j. \end{aligned}$$

For a square interval matrix $\boldsymbol{A}$ its *comparison matrix* $\langle \boldsymbol{A} \rangle$ is defined as

$$\begin{aligned} \langle \boldsymbol{A} \rangle_{ii} &= \operatorname{mig}(\boldsymbol{A}_{ii}), \\ \langle \boldsymbol{A} \rangle_{ij} &= -\operatorname{mag}(\boldsymbol{A}_{ij}) \quad \text{for } i \neq j. \end{aligned}$$

Note that $\langle \boldsymbol{A} \rangle$ is forced to be a Z-matrix.

Definition 4.17 (H-matrix)**.** A square real matrix A is an *H-matrix* if $\langle A \rangle$ is an M-matrix. A square interval matrix $\boldsymbol{A}$ is an *H-matrix* if $\langle \boldsymbol{A} \rangle$ is an M-matrix.

Hence checking of H-matrix property can be transformed to checking M-matrix property. The following equivalent conditions can be found in Neumaier [139].

Theorem 4.18. *The following statements are equivalent:*

1. $\boldsymbol{A}$ *is an H-matrix,*
2. *every* $A \in \boldsymbol{A}$ *is an H-matrix,*
3. $\langle \boldsymbol{A} \rangle$ *is regular and* $\langle \boldsymbol{A} \rangle^{-1} e > 0$.

Example 4.19. If $\boldsymbol{A}$ is an M-matrix, then $\langle A \rangle = \underline{A}$, which is according to Theorem 4.9 also an M-matrix. Therefore, every M-matrix is also an H-matrix.

Example 4.20. The slightly changed matrix from Example 4.10

$$\boldsymbol{A} = \begin{pmatrix} 2 & 1 \\ [0,2] & 2 \end{pmatrix}$$

is not an M-matrix, however $\langle \boldsymbol{A} \rangle = \begin{pmatrix} 2 & -1 \\ -2 & 2 \end{pmatrix}$ which is an M-matrix, hence $\boldsymbol{A}$ is an H-matrix because its inverse is

$$\langle \boldsymbol{A} \rangle^{-1} = \begin{pmatrix} 1 & 0.5 \\ 1 & 1 \end{pmatrix} \geq 0.$$

Example 4.21. Every regular lower or upper triangular matrix is an H-matrix [139].

Example 4.22. Every matrix that is sufficiently close to the identity matrix is also an H-matrix, i.e., every matrix that satisfies

$$\|I - \boldsymbol{A}\| < 1,$$

for some consistent matrix norm is an H-matrix [139].

Nevertheless, there exist inverse nonnegative matrices that are not H-matrices.

Example 4.23. The inverse nonnegative matrix from Example 4.15 is not even an H-matrix, because its comparison matrix is not an M-matrix (its determinant is -2).

4.5 Strictly diagonally dominant matrices

The condition for H-matrices $\langle \boldsymbol{A} \rangle u > 0$ can be rewritten for $u = (1, \ldots, 1)^T$ as

$$\operatorname{mig}(\boldsymbol{a}_{ii}) > \sum_{k \neq i} \operatorname{mag}(\boldsymbol{a}_{ik}), \quad \text{for } i = 1, \ldots, n. \tag{4.1}$$

Definition 4.24 (Strictly diagonally dominant matrix)**.** A square interval matrix $\boldsymbol{A}$ satisfying the condition 4.1 is called *strictly diagonally dominant.*

Clearly, according to its definition, every strictly diagonally dominant matrix is an H-matrix. Therefore it is also regular. Whenever a (preconditioned) matrix is close to the identity matrix then it is strictly diagonally dominant (and also an H-matrix).

Example 4.25. If $\|I - \boldsymbol{A}\|_\infty < 1$ then $\boldsymbol{A}$ is strictly diagonally dominant.

Example 4.26. The matrix

$$\boldsymbol{A} = \begin{pmatrix} 2 & [-1, 0] \\ [-1, 0] & 2 \end{pmatrix}$$

is strictly diagonally dominant and also an M-matrix (hence also inverse nonnegative).

Example 4.27. Not every strictly diagonally dominant matrix is an M-matrix. The strictly diagonally dominant matrix

$$\boldsymbol{A} = \begin{pmatrix} -2 & [0, 1] \\ [0, 1] & -2 \end{pmatrix}$$

is not an M-matrix because it is not a Z-matrix. Moreover, $\boldsymbol{A}$ is not inverse nonnegative since

$$\underline{A}^{-1} = \begin{pmatrix} -2 & 0 \\ 0 & -2 \end{pmatrix}^{-1} = \begin{pmatrix} -0.5 & 0 \\ 0 & -0.5 \end{pmatrix}.$$

Example 4.28. There exists an H-matrix that is not an M-matrix, not strictly diagonally dominant and not inverse nonnegative. The matrix

$$\boldsymbol{A} = \begin{pmatrix} 2 & 1 \\ [0, 2] & 2 \end{pmatrix}$$

is not an M-matrix (it is not a Z-matrix), but it is an H-matrix. It is clearly not strictly diagonally dominant. And since

$$\underline{A}^{-1} = \begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix}^{-1} = \begin{pmatrix} 0.5 & -0.25 \\ 0 & 0.5 \end{pmatrix},$$

it is not inverse nonnegative.

Example 4.29. The slightly adapted matrix from Example 4.26

$$\boldsymbol{A} = \begin{pmatrix} 2 & [-1,0] \\ [-1,0] & 1 \end{pmatrix},$$

is an M-matrix, however it is not strictly diagonally dominant.

Example 4.30. There exists an H-matrix that is neither an M-matrix nor strictly diagonally dominant, but it is inverse nonnegative. The matrix

$$\boldsymbol{A} = \begin{pmatrix} [1, 1+\varepsilon] & -1 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}, \quad \text{for } \varepsilon > 0,$$

is not an M-matrix (because it is not a Z-matrix), but it is an H-matrix since

$$\langle \boldsymbol{A} \rangle^{-1} = \begin{pmatrix} 1 & 1 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \geq 0.$$

$\boldsymbol{A}$ is clearly not strictly diagonally dominant. It is inverse nonnegative since

$$\underline{A}^{-1} = \begin{pmatrix} 1 & 1 & 1 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \geq 0$$

and, according to the Sherman–Morison formula,

$$\overline{A}^{-1} = \begin{pmatrix} \frac{1}{1+\varepsilon} & \frac{1}{1+\varepsilon} & \frac{1}{1+\varepsilon} \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \geq 0.$$

Example 4.31. There exists an H-matrix, that is not an M-matrix and is both strictly diagonally dominant and inverse nonnegative. The matrix

$$\boldsymbol{A} = \begin{pmatrix} [11/30, 11/30+\varepsilon] & -0.1 & 1/30 \\ -0.1 & 0.3 & -0.1 \\ 1/30 & -0.1 & 11/30 \end{pmatrix}, \quad \text{for some } \varepsilon > 0,$$

is not an M-matrix (because it is not a Z-matrix). The matrix is clearly strictly diagonally dominant. It is an H-matrix since for its comparison matrix

$$\langle \boldsymbol{A} \rangle = \begin{pmatrix} 11/30 & -0.1 & -1/30 \\ -0.1 & 0.3 & -0.1 \\ -1/30 & -0.1 & 11/30 \end{pmatrix},$$

and $u = (1,1,1)^T > 0$ it holds that $\langle \boldsymbol{A} \rangle u > 0$. Finally, it is inverse nonnegative since

$$\underline{A}^{-1} = \begin{pmatrix} 3 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 3 \end{pmatrix} \geq 0.$$

and, according to the Sherman–Morrison formula,

$$\overline{A}^{-1} = \begin{pmatrix} \frac{3}{1+3\varepsilon} & \frac{1}{1+3\varepsilon} & 0 \\ \frac{1}{1+3\varepsilon} & \frac{4+11\varepsilon}{1+3\varepsilon} & 1 \\ 0 & 1 & 3 \end{pmatrix} \geq 0.$$

4.6 Strongly regular matrices

Usually, before computing with an interval matrix some kind of preconditioning is applied. It means a matrix $\boldsymbol{A}$ is multiplied with a real regular matrix C

$$\boldsymbol{A} \mapsto C\boldsymbol{A}.$$

Such a resulting matrix might possess properties more suitable for further processing (for example it will prevent growth of intervals' widths). A usual preconditioner is $C = A_c^{-1}$. Of course, in finite arithmetic we can not often get precise midpoint inverse. Nevertheless, we can use $C \approx A_c^{-1}$. If we precondition with the midpoint inverse, first thing we want is $A_c^{-1}\boldsymbol{A}$ to be regular.

Definition 4.32 (Strongly regular matrix)**.** Let $\boldsymbol{A}$ be a square interval matrix. Let A_c be regular. If $A_c^{-1}\boldsymbol{A}$ is regular, then $\boldsymbol{A}$ is called *strongly regular.*

In [139] there are many useful conditions for deciding strong regularity.

Theorem 4.33. *Let $\boldsymbol{A}$ be a square interval matrix and A_c be regular, then the following statements are equivalent:*

1. *$\boldsymbol{A}$ is strongly regular,*
2. *$\boldsymbol{A}^T$ is strongly regular,*
3. *$\varrho(|A_c^{-1}|A_\Delta) < 1$,*
4. *$\|I - A_c^{-1}\boldsymbol{A}\| < 1$ for some consistent matrix norm,*
5. *$A_c^{-1}\boldsymbol{A}$ is an H-matrix.*

Note that the statement *3.* is a sufficient condition for regularity, hence every strongly regular matrix is regular. Let us comment on the statement *4.* When, assuming the exact arithmetics, $\boldsymbol{A}$ is preconditioned by A_c^{-1}, the resulting matrix has an identity matrix I as its midpoint. Hence we can view the resulting interval matrix as wrapping around I. To maintain its regularity, such a matrix cannot reach too far from I, which we can formulate as $\|I - A_c^{-1}\boldsymbol{A}\| < 1$.

We can also apply preconditioning from both sides

$$\boldsymbol{A} \mapsto C_1\boldsymbol{A}C_2.$$

However, that does not extend the class of strongly regular matrices [139].

Theorem 4.34. *Let $\boldsymbol{A}$ be a square interval matrix and C_1, C_2 be square real matrices. Suppose that $C_1\boldsymbol{A}C_2$ is an H-matrix, then $\boldsymbol{A}$ is strongly regular.*

Example 4.35. If we set $C_1 = C_2 = I$ in Theorem 4.34, then it implies that every H-matrix is strongly regular.

Here we have an example from [139] that not every regular matrix is strongly regular.

Example 4.36. The matrix

$$\boldsymbol{A} = \begin{pmatrix} [0,2] & 1 \\ -1 & [0,2] \end{pmatrix}$$

is regular (e.g., since $\det(\boldsymbol{A}) = [1,5] > 0$). We have

$$A_c^{-1} = \begin{pmatrix} 0.5 & -0.5 \\ 0.5 & 0.5 \end{pmatrix} \quad \text{and} \quad A_\Delta = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

hence,

$$\varrho(|A_c^{-1}|A_\Delta) = \varrho\begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix} = 1,$$

which, according to Theorem 4.33, means $\boldsymbol{A}$ is not strongly regular.

Example 4.37. The matrix from Example 4.15 is strongly regular.

Example 4.38. Regular matrices with inverse nonnegative midpoint are strongly regular [139]. Hence inverse nonnegative matrices are strongly regular too.

However, not every strongly regular matrix is inverse nonnegative.

Example 4.39. The matrix

$$\boldsymbol{A} = \begin{pmatrix} [0,1] & 1 \\ -1 & [0,1] \end{pmatrix},$$

is regular (because $\det(\boldsymbol{A}) = [1,2]$), and it is strongly regular because $\varrho(|A_c^{-1}|A_\Delta) = 0.6 < 1$. Nevertheless, its inverse is

$$\boldsymbol{A}^{-1} = \begin{pmatrix} [0.5,1] & [-1,-0.5] \\ [0.5,1] & [0,1] \end{pmatrix},$$

which means $\boldsymbol{A}$ is not inverse nonnegative.

4.7 Mutual relations

For the sake of clarity, the relations between the mentioned classes of interval matrices are captured in Figure 4.1.

4.8 More on interval matrices

There is a survey on properties of matrices that are computable in polynomial time is [75]. Discussion about the relationship between regularity and singularity can be found in [186]. We saw that when upper and lower bound matrix is an M-matrix then the whole interval matrix is an M-matrix. When checking a certain property for certain boundary matrices implies the property for all matrices included in the interval matrix, it is called *interval property*. There is a survey paper on such matrices [50]. A survey devoted to checking various matrix properties is [173]. Results regarding positive definiteness, stability and P-matrices can be found there. More on interval P-matrices can be found, e.g., in [20, 73, 185]. Other results regarding stability are [31, 63, 199]. For more on totally nonnegative interval matrices see [1, 44]. To know more about sign regular matrices see [2, 45]. For information about matrices with parametric dependencies see [76, 153]. Complexity issues related to interval matrices can be found in [112, 85]. More about inverse interval matrix can be found in [169, 183].

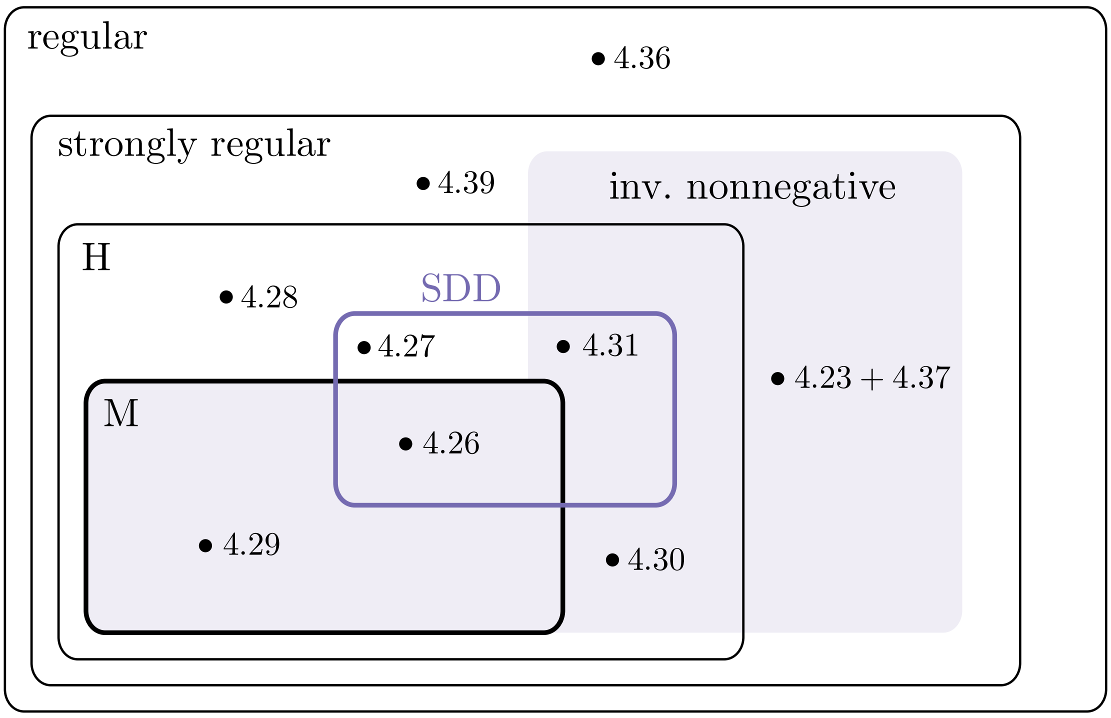


Figure 4.1: Inclusion relations between the mentioned classes of interval matrices; **SDD** (strictly diagonally dominant matrices), **M** (M-matrices), **H** (H-matrices). The numbers refer to examples that show existence of an interval matrix lying in the intersection of two particular classes. The darker area corresponds to the set of inverse nonnegative matrices.

5 Square interval linear systems

- Solution set of an linear system
- Interval hull and enclosure
- Verified solution of a real system
- Preconditioning
- Direct and iterative methods for enclosures
- Comparison of methods
- Shaving method

Interval linear systems form a crucial part of interval linear algebra. Moreover, many linear algebraic problems can be transformed to solving a square interval system. That is why in this chapter we deal with solving square interval systems first. We define what do we mean by the solution set of an interval linear system. We discuss a characterization of a solution set. As this set might be of a complex shape it is usually enclosed with an n-dimensional box for further processing. We address computation of the tightest n-dimensional box enclosing this set (the hull). However, computing the hull is an NP-hard problem, that is why we sometimes need to be satisfied with a larger box (an enclosure). Of course, the tighter the enclosure is the better. There are various methods for computing enclosures of the solution set. We divide them into two groups – the iterative methods and direct methods. We introduce some representatives for each group – Krawczyk's method, the Jacobi and Gauss–Seidel method as iterative methods and the Hansen–Bliek–Rohn–Ning–Kearfott–Neumaier method and Gaussian elimination as direct methods. Sometimes a verified solution of a real system is needed, hence we describe Rump's ε-inflation method, which can be also used as an enclosure method. The related topics such as preconditioning, finding an initial enclosure and stopping criteria are discussed as well. At the end we compare the mentioned methods, since we need to know which method to use when solving of square interval systems is later needed as a subtask of a problem. We also introduce and demonstrate our shaving method introduced in [81] that is able to further improve obtained enclosures. In this chapter we deal only with square interval systems, overdetermined systems are described in the next chapter. We conclude the chapter with a list of further references.

5.1 Solution set and its characterization

For the sake of clarity let us first define a system of interval linear equations or, as we abbreviate it, an *interval linear system.* It can be defined as a set of all real systems that are contained within bounds given by an interval matrix and an interval vector.

Definition 5.1 (Interval linear system)**.** For an $m \times n$ interval matrix $\boldsymbol{A}$ and an m-dimensional interval vector $\boldsymbol{b}$ we call the following structure an *interval linear system*

$$\{Ax = b \mid A \in \boldsymbol{A}, b \in \boldsymbol{b}\}.$$

Note that when a matrix (vector) is selected from an interval matrix (vector) each coefficient is selected independently. For the sake of simplicity it will be denoted by

$$\boldsymbol{A}x = \boldsymbol{b}.$$

When $m = n$ holds, in another words a system has the same number of variables and equations, we call it a *square system.* In practical applications, descriptions of problems often lead to square systems. If $m > n$ then a system is called *overdetermined* and if $m < n$, then a system is called *underdetermined.* Moreover, solving of square systems will be in later chapters useful for dealing with other problems, e.g., solving overdetermined systems, computing determinants, constructing the least squares regression, etc.

First, it is necessary to define what is meant by the solution set of an interval linear system.

Definition 5.2 (Solution set)**.** The solution set Σ of an interval linear system $\boldsymbol{A}x = \boldsymbol{b}$ is the defined as follows

$$\Sigma = \{\, x \mid Ax = b \text{ for some } A \in \boldsymbol{A}, b \in \boldsymbol{b}\,\}.$$

Example 5.3. The following examples are inspired by [116]. The solution set of the system

$$\begin{pmatrix} [2,4] & [-2,1] \\ [-1,2] & [2,4] \end{pmatrix} x = \begin{pmatrix} [-2,2] \\ [-2,2] \end{pmatrix},$$

forms four spikes. To obtain its top left spike we take its subsystem

$$\begin{pmatrix} [2,4] & [0,1] \\ [0,2] & [2,4] \end{pmatrix} x = \begin{pmatrix} [-2,0] \\ [0,2] \end{pmatrix}.$$

Both solution sets are depicted in Figure 5.1.

If the solution set Σ corresponding to $\boldsymbol{A}x = \boldsymbol{b}$ is nonempty, we call the system *(weakly) solvable.* If it is empty, we call the system *unsolvable.* If every real system $(Ax = b) \in (\boldsymbol{A}x = \boldsymbol{b})$ is solvable, we call the interval system *strongly solvable.* We deal with unsolvability and solvability more in Chapter 7.

It can be seen that a solution set may be of a complicated shape – it is generally nonconvex, however, it is convex in each orthant. The shape of the solution set is described by the following theorem stated in [144].

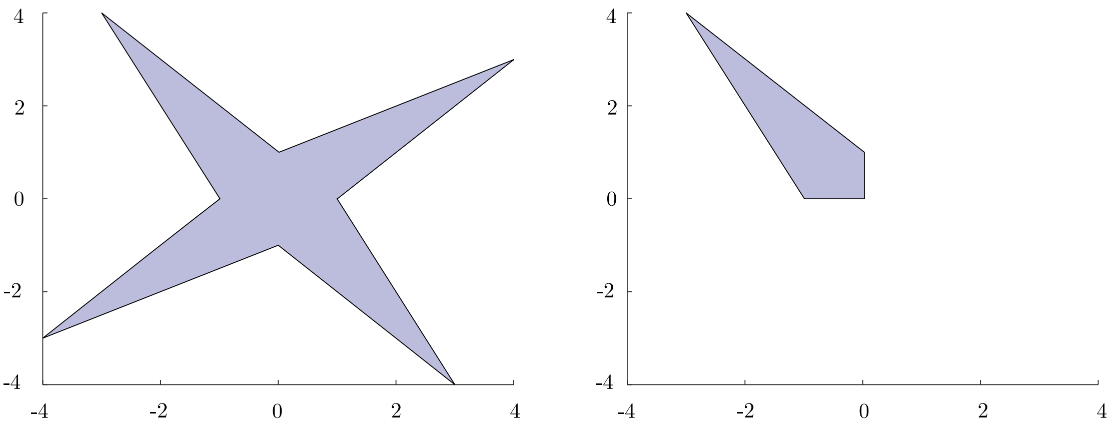


Figure 5.1: The solution sets of the interval linear systems from Example 5.3.

Theorem 5.4 (Oettli–Prager)**.** *Let us have an interval linear system* $\boldsymbol{A}x = \boldsymbol{b}$*. Vector* $x \in \mathbb{R}^n$ *is a solution of this system (*$x \in \Sigma$*) if and only if*

$$|A_c x - b_c| \leq A_\Delta |x| + b_\Delta.$$

Proof. There are various proofs of this theorem. The first proof was given in [144], a constructive proof is given in [178] and a simple proof can be found in [139]. □

Such nonlinear inequalities can be for each orthant transformed into a set of linear inequalities. That explains the convexity of the solution set in each orthant. We will show the transformation in the next section.

5.2 Interval hull

Because a solution set might be of a complicated shape, for practical use its simplified representation is more suitable. The simplest idea is to enclose it by an n-dimensional box aligned with axes. If such a box is the tightest possible we call it the interval hull. Let us define it more formally.

Definition 5.5 (Interval hull)**.** When $\boldsymbol{A}$ is regular and Σ is the solution set of $\boldsymbol{A}x = \boldsymbol{b}$, the interval vector $\boldsymbol{h} = [\underline{h}, \overline{h}]$ given by

$$\underline{h}_i = \min_{x \in \Sigma} x_i,$$
$$\overline{h}_i = \max_{x \in \Sigma} x_i \quad i = (1, \ldots, n),$$

is called the *interval hull.*

The formula in the Oettli–Prager theorem can be rewritten using linear inequalities only. The absolute values can be rewritten in the following way. We can get rid of the first one by breaking it down into the two cases

$$A_c x - b_c \leq A_\Delta |x| + b_\Delta, \tag{5.1}$$
$$-(A_c x - b_c) \leq A_\Delta |x| + b_\Delta. \tag{5.2}$$

The second absolute value can be rewritten with the use of knowledge of the orthant we currently work with. The following holds

$$|x| = D_z x, \quad \text{where } z = \operatorname{sign} x.$$

That gives rise to the condition

$$0 \leq D_z x. \tag{5.3}$$

For every orthant the conditions (5.1), (5.2) and (5.3) form a system of linear inequalities. Therefore we can use linear programming. Generally, we have to solve $(2^n \times 2n)$ linear programming problems (for each orthant in each coordinate compute the upper and lower bound). That is obviously too much computing. However, if we know some enclosure of the solution set (known in advance or computed with some of the later mentioned methods), then we can apply linear programming to only those orthants across which the enclosure stretches.

Of course, in both cases the linear programming needs to be verified. For information about its verification see, e.g., [16, 106].

If we are unlucky, we must test all the exponentially many orthants. It is no surprise since computing the exact hull is an NP-hard problem [184]. However, as we will further see, for certain classes of matrices (or systems), the orthant search and even linear programming could be avoided and there is much more convenient way to compute the interval hull.

There are other methods for computing the hull which are also possibly of exponential nature [3, 59, 94, 139, 176].

5.3 Enclosure of Σ

As computing the hull is generally a computationally difficult task, we must sometimes lower our demands and compute only an interval box containing the solution set Σ. Of course, the tighter is the box the better.

Definition 5.6 (Enclosure)**.** For an interval system $\boldsymbol{A}x = \boldsymbol{b}$ any $\boldsymbol{x} \in \mathbb{IR}^n$ such that

$$\Sigma \subseteq \boldsymbol{x}$$

is called an *enclosure*.

As enclosing the solution set of an interval linear system is a crucial task applicable to many other problems, in the next two chapters the main goal will be the following:

Problem: Compute a tight enclosure of the solution set of $\boldsymbol{A}x = \boldsymbol{b}$.

In another words our goal is always to compute the tightest enclosure in a reasonable amount of time. Note that in this work we address both the computation of hull and the computation of enclosure as *solving*. Sometimes, we refer to an enclosure of the solution set of an interval linear system just as *enclosure of the interval linear system*.

In order to obtain a finite enclosure we need the solution set to be bounded. Rohn proved in [171] that the solution set is bounded if and only if $\boldsymbol{A}$ is regular. Regularity can be checked by means discussed in Chapter 4.

In the rest of the chapter we are going to introduce various approaches to this problem. Before that, we briefly explain the concept of preconditioning borrowed from numerical mathematics.

5.4 Preconditioning of a square system

In the case of interval systems preconditioning means transforming the system into a more feasible form for further processing. Mostly, to overcome uncontrollable growth of widths of intervals during computation. Here we assume that $\boldsymbol{A}$ is a square matrix. The general transformation is

$$\boldsymbol{A}x = \boldsymbol{b} \quad \mapsto \quad (C\boldsymbol{A})x = (C\boldsymbol{b}),$$

where C is a real square matrix of a corresponding size.

The most promising choice is usually $C = A_c^{-1}$. It is also optimal from a certain viewpoint [139]. Such a preconditioning leads to a new system where the matrix has I as a midpoint (if we assume exact arithmetics). Such matrices are theoretically nice. As we saw in the previous chapter our goal is to transform $\boldsymbol{A}$ into an H-matrix. That is why strongly regular matrices play a prominent role in our problems, specially in solving of interval linear systems.

Because we work only with finite arithmetics a typical choice is preconditioning with approximate midpoint inverse, i.e., $C \approx A_c^{-1}$.

Unfortunately, beside the positive effect of preconditioning, it will often enlarge the solution set of the new system. This is the cost we need to pay.

Example 5.7. Let us take the first system from Example 5.3 and use two preconditioning matrices

$$C_1 \approx A_c^{-1} = \begin{pmatrix} 3 & -0.5 \\ 0.5 & 3 \end{pmatrix}^{-1}, \quad C_2 \approx \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}^{-1}.$$

The solution sets of the two new resulting systems are depicted in Figure 5.2. In the first case the hull of the new system is

$$\boldsymbol{h}_1 = \begin{pmatrix} [-14, 14] \\ [-14, 14] \end{pmatrix}.$$

However, in the second case the hull remains the same.

$$\boldsymbol{h}_2 = \begin{pmatrix} [-4, 4] \\ [-4, 4] \end{pmatrix}.$$

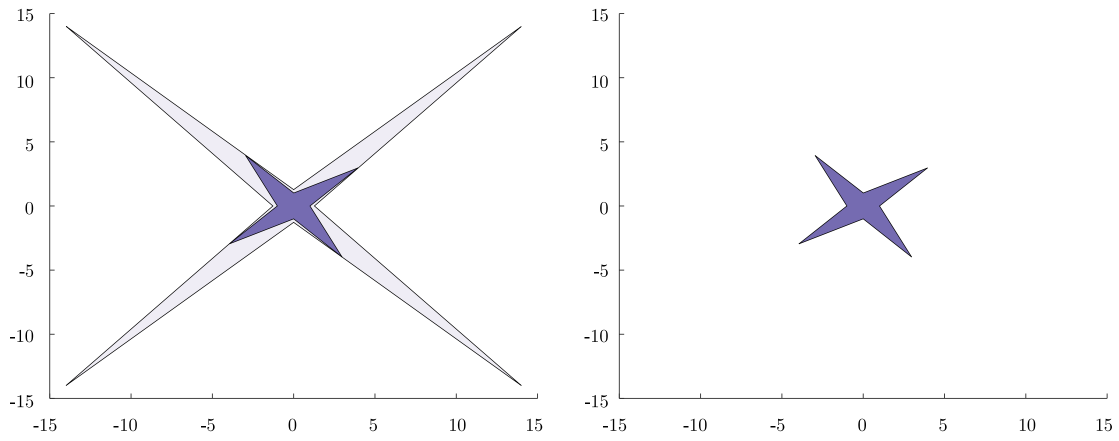


Figure 5.2: The two preconditionings from Example 5.7 – C_1(left) and C_2 (right). The darker area is the original solution set, the lighter area is the solution set of the preconditioned system.

This is no coincidence since preconditioning with a diagonal matrix D where $D_{ii} \neq 0$ preserves the original solution set [143]. This can be used for example when $\boldsymbol{A}$ is strictly diagonally dominant [189].

The preconditioning by A_c^{-1} is not always the optimal choice [74, 103]. There are other possibilities [59, 105] of preconditioning. In some cases the preconditioning is not favorable, e.g., when applying Gaussian elimination on a system where the matrix is an M-matrix [139]. In some cases preconditioning can even be avoided [201].

5.5 ε-inflation method

In further text we are also going to need a verified enclosure of a solution of a real square linear system. That is why we start with this topic first. It seems to be the same problem as solving an interval system because we need to enclose the coefficients of the real system with intervals to prevent rounding errors anyway. This is basically true. However, the radii of intervals are extremely small and hence specific methods can be used that return tight enclosures and are fast. We chose to present an efficient method introduced by Rump in his dissertation thesis [190]. In English it is described, e.g., in [191, 196]. Here we use the version described in [196].

Let us have a square real system $Ax = b$ with a nonsingular A. Our goal is to compute a tight verified enclosure of $x = A^{-1}b$. Many methods introduced later start with some initial enclosure of the solution and try to contract it or shave it. This method follows a rather opposite approach. It starts with some approximation of the solution. Let us say

$$\tilde{x} = Cb,$$

where $C \approx A^{-1}$. The initial degenerated enclosure $\boldsymbol{x}^{(0)} = [\tilde{x}, \tilde{x}]$ is then being inflated until a certain condition is met

$$\begin{aligned} \boldsymbol{y} &:= \boldsymbol{x}^{(k)} \cdot [0.9, 1.1] + [-\varepsilon, \varepsilon], \\ \boldsymbol{x}^{(k+1)} &:= Cb + (I - CA)\boldsymbol{y}. \end{aligned}$$

Relative inflation by the interval $[0.9, 1.1]$ and absolute inflation by the interval $[-\varepsilon, \varepsilon]$ for some small ε (Rump chooses $\varepsilon = 10^{-20}$) are used. Both of the intervals are set empirically. If at some iteration point

$$\boldsymbol{x}^{(k+1)} \subseteq \text{interior}(\boldsymbol{y})$$

holds, then according to the fix point theorem [189] we know that A and C are regular and

$$A^{-1}b \in \boldsymbol{x}^{(k+1)}.$$

If $\varrho(I - CA) < 1$, then the algorithm converges [196]. The algorithm works also for interval matrices (one can just replace A, b with $\boldsymbol{A}, \boldsymbol{b}$ respectively). The algorithm works also for multiple right-hand sides at once, hence it can be used to compute a verified inverse of a real matrix.

5.6 Direct computation

As we implied in the introduction, in this work we distinguish between methods with direct computation and methods with iterative computation. We start with direct methods first. The number of steps that a direct method executes can be counted in advance and for every size of input they basically provide the same number of steps.

5.6.1 Gaussian elimination

Interval version of Gaussian elimination has already been described many times, see [3, 58, 139]. It works similarly to real Gaussian elimination. Only some minor changes are needed. First, for elimination into row echelon form we use interval arithmetics instead of the real one. Second, if we view the elimination process as simultaneous elimination on all real systems contained in an interval system, then we can put an interval $[0, 0]$ instead of each eliminated element under a pivot.

Example 5.8. Notice the elimination of the element under the pivot by subtraction of the first row from the second one.

$$\begin{pmatrix} [1,2] & [1,2] \\ [1,2] & [3,3] \end{pmatrix} \sim \begin{pmatrix} [1,2] & [1,2] \\ [0,0] & [1,2] \end{pmatrix}.$$

Solving of an interval system consists of two phases – elimination and backward substitution (described as Algorithm 5.9 and 5.10 respectively).

The elimination assumes that pivot intervals do not contain zero. Sometimes a matrix can be rearranged in such a form. However, as shown in [3], such a rearrangement might not exist even if $\boldsymbol{A}$ is regular. Nevertheless, the elimination can be carried out without row interchange for H-matrices and tridiagonal matrices [3].

Algorithm 5.9 (Elimination phase)**.** The algorithm takes an interval matrix $\boldsymbol{A}$ and an interval vector $\boldsymbol{b}$ of the corresponding size and eliminates the matrix $(\boldsymbol{A} \mid \boldsymbol{b})$ into row echelon form.

1. For rows $i = 1, \ldots, (n-1)$ do the following steps.
2. For $j \in \{i, \ldots, n\}$ a find row with $0 \notin \boldsymbol{a}_{ji}$.
3. If such row cannot be found, notify that $\boldsymbol{A}$ is possibly singular.
4. For every row $j > i$ set

$$\begin{aligned} \boldsymbol{a}_{ji} &:= [0,0], \\ \boldsymbol{a}_{(j,i+1:n)} &:= \boldsymbol{a}_{(j,i+1:n)} - \frac{\boldsymbol{a}_{ji}}{\boldsymbol{a}_{ii}} \cdot \boldsymbol{a}_{(i,i+1:n)}, \\ \boldsymbol{b}_j &:= \boldsymbol{b}_j - \frac{\boldsymbol{a}_{ji}}{\boldsymbol{a}_{ii}} \cdot \boldsymbol{b}_i. \end{aligned}$$

The step 2. can be combined with some kind of pivoting. As in [61] we select a pivot with the smallest mignitude (*mignitude pivoting*).

Algorithm 5.10 (Backward substitution)**.** The algorithm takes $(\boldsymbol{A} \mid \boldsymbol{b})$ in the row echelon form and computes enclosures of all variables by systematic substitution from below.

1. For each row $i = n, \ldots, 1$ do the following steps.
2. Compute enclosure of $\boldsymbol{x}_i$ as

$$\boldsymbol{x}_i = \frac{1}{\boldsymbol{a}_{ii}} \left(\boldsymbol{b}_i - \sum_{j=i+1}^{n} \boldsymbol{a}_{ij} \boldsymbol{x}_i \right).$$

Gaussian elimination without preconditioning may work when intervals are small. However Gaussian elimination generally suffers from multiple use of interval coefficients during elimination. The widths of resulting interval enclosures tend to grow exponentially. For more information on such a phenomenon see, e.g., [133, 196].

Example 5.11. Let A_c be the 10×10 Toeplitz matrix with the first row equal to $(1, 2, 3, 4, \ldots, 9, 10)^T$. Let b_c be $(1, 1, \ldots, 1)^T$. The radii of all intervals are set to 10^{-6}. Widths of variable enclosures returned by Gaussian elimination without preconditioning are shown in Table 5.1. In each next coefficient the width of enclosure widens "roughly" by 3.

Fortunately, for special classes of matrices this algorithm works (at least theoretically) well. It can be performed without preconditioning on H-matrices with certain overestimation and it returns the hull for M-matrices and $\boldsymbol{b} > 0$ or $0 \in \boldsymbol{b}$ or $\boldsymbol{b} > 0$ [139]. For other classes of matrices use of preconditioning might be needed. Gaussian elimination with preconditioning can be proved to work better than the later introduced Jacobi and Gauss–Seidel method [139]. Gaussian elimination was also a subject to various improvements, e.g., [42, 49].

Table 5.1: Overestimation of variable enclosures by Gaussian elimination and backward substitution without preconditioning from Example 5.11. The first column indicates a variable; the width of each variable enclosure is $(\alpha \cdot 10^e)$.

variable	α	10^e
x_{10}	1.08	10^{-2}
x_9	2.62	10^{-2}
x_8	8.69	10^{-2}
x_7	2.97	10^{-1}
x_6	1.01	10^0
x_5	3.46	10^0
x_4	1.18	10^1
x_3	4.03	10^1
x_2	1.38	10^2
x_1	1.67	10^2

5.6.2 The Hansen–Bliek–Rohn–Ning–Kearfott–Neumaier method

This method was first developed by Hansen in [57] and also independently by Bliek in his dissertation thesis. The stronger results were reformulated by Rohn in [167] using only one matrix inverse. In [143], Ning and Kearfott generalized the method for H-matrices. A simpler proof was given by Neumaier in [140]. The following version of the theorem is from [143]. For simplicity we refer to this method as the HBR method.

Theorem 5.12 (HBR)**.** *Let $\boldsymbol{A}x = \boldsymbol{b}$ a square interval system, with $\boldsymbol{A}$ being an H-matrix of order n,*

$$u = \langle \boldsymbol{A} \rangle^{-1} \operatorname{mag}(\boldsymbol{b}), \quad d_i = (\langle \boldsymbol{A} \rangle^{-1})_{ii},$$

and

$$\alpha_i = \langle \boldsymbol{A} \rangle_{ii} - 1/d_i, \quad \beta_i = u_i/d_i - \operatorname{mag}(\boldsymbol{b}_i),$$

for $i = 1, \ldots, n$. Then Σ is contained in $\boldsymbol{x}$ with components

$$\boldsymbol{x}_i = \frac{\boldsymbol{b}_i + [-\beta_i, \beta_i]}{\boldsymbol{A}_{ii} + [-\alpha_i, \alpha_i]},$$

for $i = 1, \ldots, n$.

This method has a nice feature; when A_c is a diagonal matrix, then the returned $\boldsymbol{x}$ is the hull, the proof can be found found in [140, 143]. In this theorem only one computation of a verified matrix inverse is needed. The verified bounds on $\langle \boldsymbol{A} \rangle^{-1}$ can

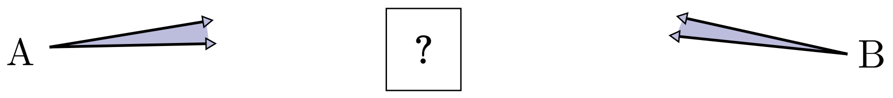


Figure 5.3: Two colliding projectiles and the area of interest.

be computed using the ε-inflation method. Another approach for finding the upper bound on $\langle \boldsymbol{A} \rangle^{-1}$ can be found in [140].

5.7 Iterative computation

In iterative computation we usually start with an initial enclosure $\boldsymbol{x}^{(0)}$ containing the solution set. Nevertheless, some methods do not require that. If they are given a box containing only a part of the solution, they compute an enclosure of this part, if they are given a box with no solution, they can usually recognize that. Such methods generate a sequence of enclosures

$$\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(k)}, \boldsymbol{x}^{(k+1)}.$$

Such a sequence is often nested

$$\boldsymbol{x}^{(0)} \supseteq \boldsymbol{x}^{(1)} \supseteq \cdots \supseteq \boldsymbol{x}^{(k)} \supseteq \boldsymbol{x}^{(k+1)}.$$

Regarding the sequence, three issues need to be addressed — how to determine $\boldsymbol{x}^{(0)}$, how to derive $\boldsymbol{x}^{(k+1)}$ from $\boldsymbol{x}^{(k)}$ and how to stop the iteration. We start with the first and the third issue. The second one depends on a particular method – we later introduce approaches by Krawczyk's method, the Jacobi method and the Gauss–Seidel method.

5.7.1 Initial enclosure

All the next methods rely on some existing enclosure $\boldsymbol{x}^{(0)}$ of the solution set. There are some ways how to determine an initial enclosure. First, we might guess it from the nature of a problem to be solved.

Example 5.13. Let us have two projectiles A, B as depicted in Figure 5.3. Both projectiles move with a known constant velocity in directions having some added uncertainty. We might be interested in whether it is possible that the two projectiles collide in the marked area. For example, when we know that there is a city in this area, we are extremely interested in solutions only in this area.

Second, we can compute an initial enclosure using some direct method and try improve it iteratively. This approach will be later used for overdetermined interval linear systems.

And third, we can use an explicit formula giving us an initial enclosure. For example the next proposition comes from [139].

Proposition 5.14. *Let $\boldsymbol{A}x = \boldsymbol{b}$ be a square interval system and C be a square real matrix of the corresponding size. If it holds that*

$$\langle C\boldsymbol{A}\rangle u \geq v > 0, \quad \textit{for some } u \geq 0,$$

then

$$\Sigma \subseteq \|C\boldsymbol{b}\|_v[-u, u].$$

A good candidate for such a vector u may be an approximate solution of the system $\langle C\boldsymbol{A}\rangle u = e$ and v can be set to $v := \langle C\boldsymbol{A}\rangle u$. The argumentation behind this can be also found in [139]. As usual, we use $C \approx A_c^{-1}$.

Sometimes much computationally easier initial enclosure can be obtained by using just maximum norm [133], when we set $\boldsymbol{A}' = I - C\boldsymbol{A}$ and $\|\boldsymbol{A}'\| < 1$ ($C\boldsymbol{A}$ is an H-matrix), then

$$\Sigma \subseteq \frac{\|C\boldsymbol{b}\|}{1 - \|\boldsymbol{A}'\|}[-e, e]. \tag{5.4}$$

5.7.2 Stopping criteria

The stopping criterion reflects similarity of two consequent enclosures in a nested sequence. Unless stated otherwise stopping criterion will be a combination of maximum number of steps (usually 20) and the following condition which takes into account the difference of lower and upper bounds separately. For two subsequent enclosures $\boldsymbol{x}^{(k)}, \boldsymbol{x}^{(k+1)}$ we stop when

$$|\underline{x}^{(k)} - \underline{x}^{(k+1)}| < \varepsilon \text{ and } |\overline{x}^{(k)} - \overline{x}^{(k+1)}| < \varepsilon,$$

where ε is a vector with all coefficients equal to some small positive number. It can be heuristically preset with respect to widths of intervals,

$$\varepsilon \approx \min_{ij}(\mathrm{wid}(\boldsymbol{A}_{ij})) \times 10^{-5}. \tag{5.5}$$

5.7.3 Krawczyk's method

The method is described in e.g., [133, 139]. For a given interval linear system $\boldsymbol{A}x = \boldsymbol{b}$ let us suppose there is an initial enclosure $\boldsymbol{x}^{(0)} \supseteq \Sigma$. For every $x = A^{-1}b$, where $A \in \boldsymbol{A}, b \in \boldsymbol{b}$ and C being a suitable real matrix it holds that

$$x = A^{-1}b = Cb - (CA - I)A^{-1}b \in C\boldsymbol{b} - (C\boldsymbol{A} - I)\boldsymbol{x}^{(0)}.$$

Hence, the iteration is

$$\begin{aligned}\boldsymbol{y}^{(i+1)} &:= C\boldsymbol{b} - (C\boldsymbol{A} - I)\boldsymbol{x}^{(i)},\\ \boldsymbol{x}^{(i+1)} &:= \boldsymbol{y}^{(i+1)} \cap \boldsymbol{x}^{(i)}.\end{aligned}$$

Due to the intersection the algorithm creates a sequence of nested interval vectors. Another point of view on Krawczyk's method is that it is a restriction of a more general method for nonlinear systems to linear systems only [110]. This method is very simple and better enclosures can be obtained by other methods. An advantage is that when a preconditioner C is available there are no divisions in the algorithm (unlike in the further introduced Jacobi and Gauss–Seidel method).

5.7.4 The Jacobi and Gauss–Seidel method

In this subsection we discuss the well-known iterative algorithms – the Jacobi method and the Gauss–Seidel method. Both methods need some initial enclosure $\boldsymbol{x}^{(0)}$. First, let us start with the Jacobi method. The ith equation of a real system $Ax = b$ is the following

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ii}x_i + \cdots + a_{in}x_n = b_i.$$

The variable x_i can be expressed as

$$x_i = \frac{1}{a_{ii}}\Big[b_i - (a_{i1}x_1 + \cdots + a_{i(i-1)}x_{i-1} + a_{i(i+1)}x_{i+1} + \cdots + a_{in}x_n)\Big].$$

When we have an interval system $\boldsymbol{A}x = \boldsymbol{b}$ and a known enclosure $\boldsymbol{x}$ we get

$$\boldsymbol{x}_i \subseteq \frac{1}{\boldsymbol{a}_{ii}}\Big[\boldsymbol{b}_i - (\boldsymbol{a}_{i1}\boldsymbol{x}_1 + \cdots + \boldsymbol{a}_{i(i-1)}\boldsymbol{x}_{i-1} + \boldsymbol{a}_{i(i+1)}\boldsymbol{x}_{i+1} + \cdots + \boldsymbol{a}_{in}\boldsymbol{x}_n)\Big].$$

This formula gives rise to iterative Algorithm 5.15.

Algorithm 5.15 (Jacobi method)**.** Input is a square system $\boldsymbol{A}x = \boldsymbol{b}$ and some initial enclosure of the solution set $\boldsymbol{x}^{(0)}$. It returns an enclosure $\boldsymbol{x}$ of a solution set.

1. For each variable $\boldsymbol{x}_i$ compute a new enclosure as

$$\boldsymbol{y}_i^{(k+1)} = \frac{1}{\boldsymbol{a}_{ii}}\left(\boldsymbol{b}_i - \sum_{j\neq i}\boldsymbol{a}_{ij}\boldsymbol{x}_j^{(k)}\right) \quad \text{for } i = (1,\ldots,n). \tag{5.6}$$

2. Intersect with the old enclosure

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k \cap \boldsymbol{y}^{k+1}.$$

3. Repeat steps 1. to 2. until stopping criteria are not met.

From the prescription of iterative improvement it is clear that it can be computed in parallel for each variable. It is possible to rewrite (5.6) in a form that helps mathematical software with optimized matrix multiplication:

$$\boldsymbol{y}^{(k+1)} = \boldsymbol{D}^{-1}(\boldsymbol{b} - \boldsymbol{J}\boldsymbol{x}^k), \tag{5.7}$$

where $\boldsymbol{D}$ is the main diagonal of $\boldsymbol{A}$ and $\boldsymbol{J}$ is $\boldsymbol{A}$ with the intervals on the main diagonal set to $[0, 0]$.

Example 5.16. In this example we show the differences between the two versions of the Jacobi iteration (5.6) and (5.7). They were compared on random interval linear systems. The midpoint coefficients of a system were taken independently uniformly from interval $[-10, 10]$ and then wrapped with intervals with radii 10^{-3}. The simple initial enclosure (5.4) was used and we set $\epsilon = 10^{-6}$. We tested using the DESKTOP setting. Both methods returned identical enclosures and did identical number of iterations, only the computation times differed. For each size we took the average computation time on 100 systems. The average computation times in seconds are displayed in Table 5.2. The average times include preconditioning. The matrix version of the Jacobi method is clearly faster.

Table 5.2: The two implementations of the Jacobi method (5.6) and (5.7) – average computation times (in seconds), n is the number of variables of a system.

n	Jacobi (5.6)	matrix Jacobi (5.7)
10	0.26	0.07
20	0.52	0.07
30	0.92	0.08
40	1.31	0.10
50	1.72	0.11
60	2.25	0.14
70	2.67	0.17
80	3.21	0.20
90	3.76	0.25
100	4.37	0.30

The previous results need to be taken with caution, because they may be partly system/software dependent. However, when optimized matrix multiplication is accessible, we recommend to use the (5.7) version of the Jacobi algorithm.

The Gauss–Seidel method is an improvement of the Jacobi method. The only difference is that it immediately uses the newly computed enclosures of variables.

Algorithm 5.17 (Gauss–Seidel method)**.** Substitute the formula (5.6) in step 2. of the Jacobi method by

$$\boldsymbol{y}_i^{(k+1)} = \frac{1}{\boldsymbol{a}_{ii}}\left(\boldsymbol{b}_i - \sum_{j<i}\boldsymbol{a}_{ij}\boldsymbol{x}_j^{(k+1)} - \sum_{j>i}\boldsymbol{a}_{ij}\boldsymbol{x}_j^{(k)}\right) \quad \text{for } i = (1, \ldots, n).$$

The advantage of the Gauss–Seidel method is its faster convergence, the drawback is that it cannot be parallelized. Since the the number of operations per one iteration is the same as for the Jacobi method (5.6), when comparing the Gauss–Seidel and the parallel Jacobi we could expect the similar results regarding computation time as in Example 5.16. During experiments testing the fewer iterations did compensate for much larger time needed per one iteration.

Both methods assume there are no intervals containing 0 on the main diagonal of $\boldsymbol{A}$. If this is not the case, then extended interval arithmetic can be used [133]. This does not happen for Krawczyk's method.

It is proved that when used with a preconditioner C, the Gauss–Seidel method never yields worse bounds than any method based on matrix splitting of $C\boldsymbol{A}$ (i.e., the Jacobi, Krawczyk's, etc.) [139]. However, the Jacobi and the Gauss–Seidel practically converge to the same enclosures.

When applied to a system with an M-matrix, both methods can be used without preconditioning and return the hull. Similarly to Krawczyk's method, for both methods, if some $\emptyset \neq \boldsymbol{y}^{(k+1)} \subseteq$ interior $\boldsymbol{x}^{(0)}$ (the initial enclosure is strictly improved during iterations) then it proves that $\boldsymbol{A}$ is an H-matrix [139].

5.8 Small comparison of methods

In upcoming problems we are going to exploit existence of methods for solving square interval linear systems. That is why at the end of this chapter we provide a small comparison of the mentioned methods. We are going to compare:

- `ge` – Gaussian elimination with mignitude pivoting,
- `jacobi` – matrix version of the Jacobi method (5.7) with maximum number of iterations set to 20 and ε chosen according to (5.5); for initial enclosure we use the formula (5.4),
- `krawczyk` – Krawczyk's method with the same setting as `Jacobi`,
- `hbr` – the Hansen–Bliek–Rohn method; the enclosure on $\langle A\rangle^{-1}$ is computed using the inflation method.

The suffix `+pre` means that the method is used with preconditioning by midpoint inverse.

In [143] they compare several methods for computing enclosures of interval linear systems; mostly variants of the HBR-method and Gaussian elimination. We borrow their three examples and demonstrate also properties of other methods.

Example 5.18. Let us have the system $\boldsymbol{A}x = \boldsymbol{b}$, where

$$\boldsymbol{A} = \begin{pmatrix} [4,6] & [-1,1] & [-1,1] & [-1,1] \\ [-1,1] & [-6,-4] & [-1,1] & [-1,1] \\ [-1,1] & [-1,1] & [9,11] & [-1,1] \\ [-1,1] & [-1,1] & [-1,1] & [-11,-9] \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} [-2,4] \\ [1,8] \\ [-4,10] \\ [2,12] \end{pmatrix}.$$

Note that $\boldsymbol{A}$ is a strictly diagonally dominant, hence Gaussian elimination can be used without preconditioning [139]. The resulting enclosure is

$$\boldsymbol{x}^{\texttt{ge}} = \begin{pmatrix} [-2.60, 3.10] \\ [-3.90, 1.50] \\ [-1.43, 2.15] \\ [-2.35, 0.60] \end{pmatrix}.$$

When using the Jacobi method we obtain

$$\boldsymbol{x}^{\texttt{jacobi}} = \begin{pmatrix} [-2.60, 3.10] \\ [-3.90, 1.65] \\ [-1.48, 2.15] \\ [-2.35, 0.79] \end{pmatrix}.$$

Notice that in each interval coefficient at least one bound is exactly the same as in $\boldsymbol{x}^{\mathtt{ge}}$. The HBR method returns the narrowest enclosure

$$\boldsymbol{x}^{\mathtt{hbr}} = \begin{pmatrix} [-2.50, 3.10] \\ [-3.90, 1.20] \\ [-1.40, 2.15] \\ [-2.35, 0.60] \end{pmatrix},$$

and since the midpoint matrix A_c of $\boldsymbol{A}$ is diagonal, it is the interval hull.

When $\boldsymbol{A}$ is an H-matrix and A_c is diagonal then it can be proved that $\boldsymbol{x}^{\mathtt{ge}} \subseteq \boldsymbol{x}^{\mathtt{jac}}$ and in each interval coefficient has at least one bound the same [139]. Let us use another example from [143], in which $\boldsymbol{A}$ is an M-matrix.

Example 5.19. Let us have a system $\boldsymbol{A}x = \boldsymbol{b}$, where

$$\boldsymbol{A} = \begin{pmatrix} [3.7, 4.3] & [-1.5, -0.5] & [0, 0] \\ [-1.5, -0.5] & [3.7, 4.3] & [-1.5, 0.5] \\ [0, 0] & [-1.5, -0.5] & [3.7, 4.3] \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} [-14, 14] \\ [-9, 9] \\ [-3, 3] \end{pmatrix}.$$

Using the previously tested four methods we get

$$\boldsymbol{x}^{\mathtt{ge}} = \boldsymbol{x}^{\mathtt{jacobi}} = \boldsymbol{x}^{\mathtt{hbr}} = \begin{pmatrix} [-6.38, 6.38] \\ [-6.40, 6.40] \\ [-3.40, 3.40] \end{pmatrix},$$

which is the interval hull.

In the next example Gaussian elimination gives better bounds.

Example 5.20. Using the previous example with a different right-hand side

$$\boldsymbol{b} = \begin{pmatrix} [-14, 0] \\ [-9, 0] \\ [-3, 0] \end{pmatrix}.$$

The tightest enclosure is returned by Gaussian elimination and the Jacobi method without preconditioning

$$\boldsymbol{x}^{\mathtt{ge}} = \boldsymbol{x}^{\mathtt{jacobi}} \begin{pmatrix} [-6.38, 0] \\ [-6.40, 0] \\ [-3.40, 0] \end{pmatrix}.$$

The enclosure returned by Gaussian elimination with preconditioning gives

$$\boldsymbol{x}^{\mathtt{ge+pre}} = \begin{pmatrix} [-6.38, 1.35] \\ [-6.40, 1.74] \\ [-3.40, 1.40] \end{pmatrix}.$$

Table 5.3: Square interval linear systems – comparison of enclosures. The reference method is `hbr`, the uniform radii was set to $r = 0.001$, n is the number of variables of a system.

n	`ge+pre`	`jacobi+pre`	`krawczyk+pre`
10	1.00083	1.00012	1.00187
20	1.00091	1.00005	1.00139
30	1.00122	1.00021	1.00222
40	1.00081	1.00025	1.00207
50	1.00091	1.00024	1.00200
60	1.00065	1.00021	1.00192
70	1.00058	1.00031	1.00232
80	1.00085	1.00032	1.00231
90	1.00086	1.00039	1.00238
100	1.00119	1.00038	1.00240

The HBR method gives wider bounds

$$\boldsymbol{x}^{\texttt{hbr}} = \begin{pmatrix} [-6.38, 1.12] \\ [-6.40, 1.54] \\ [-3.40, 1.40] \end{pmatrix}.$$

Sometimes the HBR method gives better bounds, sometimes Gaussian elimination does. In some cases both methods return the same bounds and in some cases the intersection of their results gives even shaper bounds [143].

Now let us test more thoroughly for larger random systems. The systems are generated as in Example 5.16. We test on 100 systems for each size (number of variables n). The reference method is `hbr`; other methods are compared to it using 3.9. The ratios of enclosures for radii $r = 0.001$ are in Table 5.3, computation times in Table 5.4 and number of finite enclosures returned in Table 5.5. Clearly `hbr` is the winner from both computational time and enclosure tightness perspective. The similar results for radii $r = 0.01$ are displayed in Tables 5.6, 5.7 and 5.8.

It can be seen, that the methods return similar results. It happens because all methods actually use preconditioning (explicitly or implicitly) after which the resulting matrix in system is an H-matrix. For slightly larger radii the ratios are still similar, however, some methods fail to produce finite enclosures for larger systems. This happens mostly because of failure of the initial enclosure (5.4).

5.9 Shaving method

Most of the previously mentioned methods use preconditioning. We saw that preconditioning can inflate the original solution set and even though we can get close to the

Table 5.4: Square interval linear systems – average computation times. The uniform radii was set to $r = 0.001$, the computation times are in seconds, n is the number of variables of a system.

n	`ge+pre`	`jacobi+pre`	`hbr`	`krawczyk+pre`
10	0.44	0.10	0.03	0.11
20	1.65	0.11	0.04	0.12
30	3.64	0.14	0.05	0.15
40	6.41	0.16	0.06	0.17
50	10.00	0.18	0.08	0.21
60	14.42	0.21	0.11	0.25
70	19.62	0.25	0.14	0.32
80	25.67	0.30	0.18	0.39
90	32.59	0.36	0.23	0.49
100	40.39	0.42	0.29	0.61

Table 5.5: Square interval linear systems – percentage of finite enclosures returned. The uniform radii was set to $r = 0.001$, n is the number of variables of a system.

n	`ge+pre`	`jacobi+pre`	`hbr`	`krawczyk+qpre`
10	100	98	100	98
20	100	93	100	93
30	97	90	97	91
40	96	91	96	91
50	95	84	95	85
60	97	88	97	90
70	94	80	94	85
80	94	71	94	78
90	89	68	89	75
100	90	55	90	65

Table 5.6: Square interval liner systems – enclosures comparison. The reference method is `hbr`, the uniform radii was set to $r = 0.01$, the symbol `'-'` means that a method returned no finite enclosure in all 100 test cases, n is the number of variables of a system.

n	`ge+pre`	`jacobi+pre`	`krawczyk+pre`
10	1.00430	1.00178	1.01213
20	1.00303	1.00247	1.01208
30	1.00444	1.00226	1.01004
40	1.00648	1.00251	1.01007
50	1.00678	1.00244	1.00911
60	1.00812	-	1.00939
70	1.00772	-	-
80	1.00842	-	-
90	1.00877	-	-
100	1.00749	-	-

Table 5.7: Square interval liner systems – average computation times. The uniform radii was set to $r = 0.01$, the computation times are in seconds, the symbol `'-'` means that a method returned no finite enclosure in all 100 test cases, n is the number of variables of a system.

n	`ge+pre`	`jacobi+pre`	`hbr`	`krawczyk+pre`
10	0.44	0.13	0.03	0.14
20	1.64	0.18	0.04	0.18
30	3.64	0.20	0.05	0.21
40	6.41	0.24	0.06	0.25
50	10.04	0.27	0.08	0.30
60	14.43	-	0.11	0.37
70	19.73	-	0.14	-
80	25.78	-	0.18	-
90	32.88	-	0.23	-
100	40.69	-	0.29	-

Table 5.8: Square interval liner systems – percentage of finite enclosures returned. The uniform radii is set to $r = 0.01$, n is the number of variables of a system.

n	`ge+pre`	`jacobi+pre`	`hbr`	`krawczyk+qpre`
10	98	90	98	90
20	92	80	92	82
30	83	56	83	60
40	70	19	70	28
50	59	6	59	11
60	43	0	43	1
70	31	0	31	0
80	10	0	10	0
90	5	0	5	0
100	3	0	3	0

interval hull of the preconditioned system, we still may get large overestimation, see Example 5.3.

The resulting enclosure can be further tightened/shaved. In this section we explain our method that provides such a shaving. The term "shaving" is borrowed from the area of solving constraint satisfaction problems [54, 214]. This section is an adapted version of our paper [81].

Let $\boldsymbol{x} \in \mathbb{IR}^n$ be an initial enclosure of the solution set Σ. The main idea behind shaving methods is to examine a slice of $\boldsymbol{x}$. An upper α-slice $\boldsymbol{x}(\uparrow, i, \alpha)$ is defined for ith variable and a nonnegative width α as

$$\boldsymbol{x}(\uparrow, i, \alpha)_j = \begin{cases} \boldsymbol{x}_j & \text{if } j \neq i, \\ [\overline{x}_j - \alpha, \overline{x}_j] & \text{if } j = i. \end{cases} \tag{5.8}$$

If we find that $\boldsymbol{x}(\uparrow, i, \alpha)$ contains no solution, then we cut off the slice and the tighter enclosure $\boldsymbol{x}'$ reads

$$\boldsymbol{x}'_j := \begin{cases} \boldsymbol{x}_j & \text{if } j \neq i, \\ [\underline{x}_j, \overline{x}_j - \alpha] & \text{if } j = i. \end{cases}$$

The situation is similar for the lower α-slice $\boldsymbol{x}(\downarrow, i, \alpha)$. The enclosure can be repeatedly shaved by choosing various variables for $i \in \{1, \dots, n\}$ and their lower and upper slices. Naturally, the larger the width of a slice is the more efficient is the shaving. To develop an efficient shaving method, we need a shaving condition that decides whether there is no solution contained in a given box $\boldsymbol{x}$. Let us start with a real system first.

Lemma 5.21 (Hladík, Horáček [81])**.** *Let $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$ and $\boldsymbol{x} \in \mathbb{IR}^n$. Then the linear system*

$$Ax = b, \quad x \in \boldsymbol{x}$$

has no solution if and only if the linear system

$$A^T w + y - z = 0, \quad b^T w + \overline{x}^T y - \underline{x}^T z = -1, \quad y, z \geq 0 \tag{5.9}$$

is solvable.

Proof. The system can be rewritten as a system of inequalities

$$\begin{aligned} Ax &\le b, \\ -Ax &\le -b, \\ Ix &\le \overline{x}, \\ -Ix &\le -\underline{x}. \end{aligned}$$

By the well-known Farkas lemma (cf. [40]), the system $Ax = b$, $\underline{x} \le x \le \overline{x}$ has no solution if and only if the linear system

$$A^T w_1 - A^T w_2 + y - z = 0, \quad b^T w_1 - b^T w_2 + \overline{x}^T y - \underline{x}^T z < 0, \quad w_1, w_2, y, z \ge 0$$

is solvable. After substituting $w = w_1 - w_2$ we obtain

$$A^T w + y - z = 0, \quad b^T w + \overline{x}^T y - \underline{x}^T z < 0, \quad y, z \ge 0 \tag{5.10}$$

Since every positive multiple of $(w, y, z)^T$ also solves the system (5.10), the system (5.9) can be obtained after normalization. □

Now, we see that

$$Ax = b, \quad A \in \boldsymbol{A}, \ b \in \boldsymbol{b}, \ x \in \boldsymbol{x} \tag{5.11}$$

has no solution if and only if (5.9) is solvable for each $A \in \boldsymbol{A}$ and $b \in \boldsymbol{b}$. Checking such a type of solvability (so called *strong solvability*) is known to be computationally difficult (more precisely, coNP-complete); see Chapter 11. Below, we present an adaptation of the sufficient condition developed in [70].

5.9.1 A sufficient condition for strong solvability

In this section we show a heuristic way to show that (5.9) is strongly solvable. First we try to guess a vector (w, y, z) that satisfies a special one particular instance of (5.9), where $A \in \boldsymbol{A}, b \in \boldsymbol{b}, x \in \boldsymbol{x}$, i.e., for the midpoint system $A_c x = b_c, x \in \boldsymbol{x}$. Such a vector will give us a hint how to transform the system (5.9) into a square interval system that can be solved using the before mentioned means. Enclosure of the solution set in a certain shape can prove strong solvability of (5.9).

First, we solve the linear programming problem

$$\min b_c^T w + \overline{x}^T y - \underline{x}^T z$$

subject to

$$A_c^T w + y - z = 0, \ -e \le w \le e, \ y, z \ge 0,$$

and denote an optimal solution by w^*, y^*, z^*. The w is additionally bounded to prevent infinite optimal value. Notice that the solution needs not to be verified as it plays a role of a heuristic only. Suppose that the optimal value is negative. If it is not the case, then (5.9) is not solvable for $A = A_c$, $b = b_c$, and hence $\boldsymbol{x}$ contains a solution. If $y_i^* = 0$ for some i, then we fix the variable $y_i = 0$, and similarly for the entries of z^*. From now on, y, z denotes variables with the fixed values. This way we get rid of some columns (and also variables) of the system. After the fixation we obtain the potentially smaller system

$$A^T w + y - z = 0, \quad b^T w + \overline{x}^T y - \underline{x}^T z = -1, \tag{5.12}$$

where $A \in \boldsymbol{A}$ and $b \in \boldsymbol{b}$.

If it is an overdetermined system, then it has the form of $A^T w = 0$, $b^T w = -1$ (A is a square matrix, hence the only possibility to obtain an overdetermined system is $y = z = 0$). As a positive multiple of w^* solves $A_c^T w = 0$, we have that A_c is singular, which contradicts the assumption that Σ is bounded by $\boldsymbol{x}$. If (5.12) is underdetermined, then we add equations to the system to make it square. The left-hand side of the additional equations will be formed by an orthogonal basis of the null space of (5.12), and the right-hand side is calculated such that w^*, y^*, z^* solves the equations. Now we are sure that we have a square system. We denote it by

$$Cv = d, \quad C \in \boldsymbol{C}. \tag{5.13}$$

Let $v = (w, u)$ be the solution of the system, where u consists of the variables originating from (y, z). In a similar manner, let $\boldsymbol{v} = (\boldsymbol{w}, \boldsymbol{u})$ be an enclosure of the solution set of (5.13). If $\boldsymbol{u} \geq 0$, then (5.9) is solvable for each interval instance, which implies that (5.11) is not strongly solvable.

5.9.2 Computing the width of a slice

Now, we employ the above ideas to handle the problem of determining as large as possible slice of $\boldsymbol{x}$ containing no solution. Since the slice $\boldsymbol{x}$ has the form of (5.8), which depends on the parameter $\alpha \geq 0$, the interval system (5.12) and also (5.13) depend on α, too. Thus, we have to determine the largest value of α such that an enclosure to (5.13) still satisfies the nonnegativity condition $\underline{u} \geq 0$.

One possibility is to use a binary search for the optimal α. However, this would require solving plenty of interval linear systems. In the following, we rather describe a simple method for calculating a feasible, not necessary optimal, value of α.

Due to the way the lower and upper α-slice (5.8) are defined, when we take an α-slice of $\boldsymbol{x}$, such an α occurs only once in the system (5.13) (exactly in one coefficient of modified $\underline{x}$ or $\overline{x}$ after fixation). Moreover, the new system is

$$(C + \alpha E_{ij})v = d, \quad C \in \boldsymbol{C}, \tag{5.14}$$

where $E_{ij} = e_i e_j^T$ is the matrix with 1 at a certain position (i, j), and zeros elsewhere. The solution of the new system can be easily expressed.

Lemma 5.22 (Hladík, Horáček [81]). *Let $\widetilde{v}$ be a solution to $Cv = d$. Then the solution of $(C + \alpha E_{ij})v = d$ is*

$$\widetilde{v} - \frac{\alpha \widetilde{v}_j}{1 + \alpha C_{ji}^{-1}} C_{*i}^{-1}.$$

Proof. By the Sherman–Morrison formula for the inverse we get

$$\begin{aligned}(C+\alpha E_{ij})^{-1} &= (C+(\alpha e_i)e_j^T)^{-1}\\ &= C^{-1} - \frac{C^{-1}\left(\alpha e_i e_j^T\right)C^{-1}}{1+e_j^T C^{-1}\alpha e_i}\\ &= C^{-1} - \frac{\alpha}{1+\alpha C^{-1}_{ij}}\left(C^{-1}e_i\right)\left(e_j^T C^{-1}\right)\\ &= C^{-1} - \frac{\alpha}{1+\alpha C^{-1}_{ji}} C^{-1}_{*i} C^{-1}_{j*}.\end{aligned}$$

Multiplying by d we get

$$(C+\alpha E_{ij})^{-1}d = C^{-1}d - \frac{\alpha}{1+\alpha C^{-1}_{ji}} C^{-1}_{*i} C^{-1}_{j*} d = \widetilde{v} - \frac{\alpha \widetilde{v}_j}{1+\alpha C^{-1}_{ji}} C^{-1}_{*i}.$$

□

Suppose we already have an enclosure $\boldsymbol{v}$ of the solution set of (5.13) for a 0-slice. By the above lemma, an enclosure of the solution set of (5.14) is

$$\boldsymbol{v} - \frac{\alpha \boldsymbol{v}_j}{1+\alpha \boldsymbol{C}^{-1}_{ji}} \boldsymbol{C}^{-1}_{*i}.$$

Now we try to increase $\alpha > 0$ until it still satisfies several conditions of this formula. When $\alpha = 0$, then the denominator is 1. After inflating α the denominator should stay positive, otherwise it contains zero. If $\underline{\boldsymbol{C}^{-1}_{ji}} \geq 0$ then no restriction is forced on α. If $\underline{\boldsymbol{C}^{-1}_{ji}} < 0$ then it must hold that

$$-1 < \alpha \cdot \left[\underline{\boldsymbol{C}^{-1}_{ji}}, \overline{\boldsymbol{C}^{-1}_{ji}}\right],$$

which gives the first restriction on α

$$\alpha < -\frac{1}{\underline{\boldsymbol{C}^{-1}_{ji}}}. \tag{5.15}$$

In order to keep (5.11) unsolvable $\boldsymbol{u}$ must remain nonnegative after inflating the slice by α. For each variable corresponding to $\boldsymbol{u}$ it is necessary that

$$\boldsymbol{u}_k - \frac{\alpha \boldsymbol{u}_j}{1+\alpha \boldsymbol{C}^{-1}_{ji}} \boldsymbol{C}^{-1}_{ki} \geq 0.$$

Since the left-hand side is an interval, its lower bound is required to be nonnegative, i.e.

$$\underline{u}_k - \frac{\alpha}{1+\alpha \underline{\boldsymbol{C}^{-1}_{ji}}} \overline{\boldsymbol{u}_j \boldsymbol{C}^{-1}_{ki}} \geq 0.$$

By expressing α, we obtain

$$\alpha \leq \frac{\underline{u}_k}{\overline{\boldsymbol{u}_j \boldsymbol{C}^{-1}_{ki}} - \underline{u}_k \underline{\boldsymbol{C}^{-1}_{ji}}}. \tag{5.16}$$

for each k such that $\overline{\boldsymbol{u}_j \boldsymbol{C}^{-1}_{ki}} > \underline{u}_k \underline{\boldsymbol{C}^{-1}_{ji}}$.

Finally, from the formulas (5.15) and (5.16) we determine the maximal feasible α^* for inflating the α-slice. In order that the result is reliable, the formulas should be evaluated by interval arithmetic (even though they contain real variables only).

The computational cost of this method for computing α^* is low. We have to calculate $\boldsymbol{v}$, an enclosure to (5.13), and $\boldsymbol{C}^{-1}_{*i}$, which is an enclosure to the solutions set of the interval system

$$Cu = e_i, \quad C \in \boldsymbol{C}.$$

In total, we need to solve only two interval linear systems of equations. On the other hand, the computed α^* may not be the largest possible width of the slice.

5.9.3 Iterative improvement

Since α^* need not be optimal, we can think of improving it by repeating the whole process. We put $\alpha := \alpha^*$, and $\boldsymbol{v}$ will be an enclosure to (5.14). Similarly, $\boldsymbol{C}^{-1}_{*i}$ will be an enclosure to the solutions set of the interval system

$$(C + \alpha E_{ij})u = e_i, \quad C \in \boldsymbol{C}. \tag{5.17}$$

We determine the corresponding slice width α°, update $\alpha^* := \alpha^* + \alpha^\circ$ and repeat the process while improvement is significant (i.e., α° is large enough). Each iteration requires solving two interval systems, however, since the systems differ in one coefficient only, the new enclosures can be computed more effectively.

First, if we used the preconditioning by the (approximate) midpoint inverse, we can reuse the preconditioner from the previous iteration as the midpoint of (5.14) differs at the entry (i, j) only, its inverse is easily updated by using the Sherman–Morrison formula.

Updating the enclosure to (5.17) can be done even more efficiently. For a given $C \in \boldsymbol{C}$, we have by the Sherman–Morrison formula

$$(C + \alpha E_{ij})^{-1} = C^{-1} - \frac{\alpha}{1 + \alpha C^{-1}_{ji}} C^{-1}_{*i} C^{-1}_{j*}.$$

Its ith column draws

$$(C + \alpha E_{ij})^{-1}_{*i} = C^{-1}_{*i} - \frac{\alpha}{1 + \alpha C^{-1}_{ji}} C^{-1}_{*i} C^{-1}_{ji} = \frac{1}{1 + \alpha C^{-1}_{ji}} C^{-1}_{*i}.$$

Thus, $\boldsymbol{C}^{-1}_{*i}$ is updated as

$$\frac{1}{1 + \alpha \boldsymbol{C}^{-1}_{ji}} \boldsymbol{C}^{-1}_{*i}$$

without solving any system. Since the jth updated element $\boldsymbol{C}^{-1}_{ji}$ may be overestimated, we rather compute it by

$$\frac{1}{\alpha + 1/\boldsymbol{C}^{-1}_{ji}} \quad \text{instead of} \quad \frac{1}{1 + \alpha \boldsymbol{C}^{-1}_{ji}} \boldsymbol{C}^{-1}_{ji}.$$

In summary, while the first iteration needs to solve two interval systems, the others need to solve only one.

Table 5.9: Testing the shaving method without iterative improvement. The fixed radius is denoted by r, the computation time is in seconds, n is the number of variables of a system, the ratio column shows the improvement to `verifylss`, the last column shows the average number of shaved off slices.

n	r	time	ratio	# shavings
5	0.5	0.2568	0.7137	13.02
10	0.25	0.6375	0.7522	30.94
20	0.05	1.879	0.7848	61.09
50	0.025	14.58	0.8569	187.2
100	0.01	78.78	0.9049	373.8

5.9.4 Testing the shaving method

To give a hint about the cases for which the shaving method can help, let us present two tables from our previous work [81]. The method was tested on square interval systems with various fixed radii. Random square systems were generated and tested in the same way as in Example 5.16. The computations were carried out in Matlab 7.11.0.584 (R2010b) on a six-processor machine AMD Phenom(tm) II X6 1090T Processor, CPU 800 MHz, with 15579 MB RAM. Interval arithmetics and some basic interval functions were provided by the interval toolbox Intlab v6 [188]. The shaving method was run on an enclosure returned by Intlab method `verifylss`, which is a combination of a modified Krawczyk's method and the HBR method [61]. The quality of enclosures was compared using the formula (3.10). In Table 5.9 we see the results for shaving without iterative improvement of shaved slices widths (each variable is shaved only once from above and from below). In Table 5.10 the shaving method is tested on the same data but with added iterative improvement. For small interval radii the before mentioned methods return tight enough results and use of the shaving method is superfluous. However, for relatively large interval radii (such that the interval matrix is "nearly" singular) the shaving method pays off.

5.10 Some other references

There are other methods for solving interval linear systems, e.g., [14, 71]. Some methods can deal with matrices of a certain class [4, 93]. The results for systems with Toeplitz matrices are in [47]. To learn more about other concepts of solvability see, e.g., [178, 202]. More on block systems can be found in [48]. For verified solution of large systems see [192], for sparse systems see [193]. Methods for solving square interval linear systems were also compared in, e.g., [61, 143].

Table 5.10: Testing the shaving method with iterative improvement. The fixed radius is denoted by r, the computation time is in seconds, n is the number of variables of a system, the ratio column shows the improvement to `verifylss`, the last column shows the average number of shaved off slices.

n	r	time	ratio	# shavings
5	0.5	0.4977	0.6465	18.06
10	0.25	0.9941	0.6814	45.06
20	0.05	3.136	0.7161	87.77
50	0.025	26.65	0.8071	281.9
100	0.01	228.5	0.8693	946.3

6 Overdetermined interval linear systems

- ▶ The least squares solution
- ▶ Preconditioning of an overdetermined system
- ▶ Various methods for enclosing a solution set
- ▶ Subsquares method and its variations
- ▶ Comparison of methods

When a system has more equations than variables, we call it *overdetermined*. The previously described methods for solving square systems usually cannot be applied directly to them. In this chapter we first introduce the traditional approach to solving such systems via the least squares. We then discuss preconditioning for the overdetermined case. Then modification of some earlier known methods — Jacobi, Gaussian elimination — is shown. Rohn's method is introduced. All the methods are compared. The chapter is loosely based on our paper [83]. Similarly to introducing the shaving method in the previous chapter, here we introduce our subsquares method [84] that can further improve the obtained enclosure or can be used separately. Several variants of this method are developed. Its favorable properties are discussed. We end the chapter with more references to another methods for solving overdetermined and underdetermined systems.

6.1 Definition

Let us start with the formal definition.

Definition 6.1. (Overdetermined interval linear system) Let us have an interval matrix $\boldsymbol{A} \in \mathbb{IR}^{m \times n}$, where $m > n$ and an interval vector $\boldsymbol{b} \in \mathbb{IR}^{m}$. We call

$$\boldsymbol{A}x = \boldsymbol{b}$$

an *overdetermined* interval linear system.

To motivate the use of overdetermined interval systems see the following example.

Example 6.2. This example is borrowed from [165]. Let us have an $n \times n$ matrix A for which we want to compute an eigenvector corresponding to a known eigenvalue λ.

It is known that it can be computed as a solution of the following system

$$(A - \lambda I)v = 0.$$

However, the matrix on the left side is singular. To overcome this, let us "normalize" v. Either the first coefficient of v is 0 or v can be multiplied by a suitable scalar to make the first coefficient equal to 1. After setting $C = (A - \lambda I)$ for both cases the above system can be rewritten as the two following overdetermined systems:

$$\begin{pmatrix} c_{1,2} & \dots & c_{2,n} \\ \vdots & & \vdots \\ c_{n,2} & \dots & c_{n,n} \end{pmatrix} \cdot \begin{pmatrix} v_2 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \quad \text{or} \quad \begin{pmatrix} -c_{1,1} \\ \vdots \\ -c_{n,1} \end{pmatrix}.$$

If the second system is solvable and x' is the solution of the system, then $(1, x')^T$ is the desired eigenvector. If the second system is not solvable, we can recursively repeat the procedure for the first system. The algorithm can be accordingly applied to an interval matrix $\boldsymbol{A}$.

6.2 The least squares approach

When most people work with an overdetermined system they understand its solution in the least squares perspective.

Definition 6.3 (Interval least squares)**.** For an overdetermined interval linear system $\boldsymbol{A}x = \boldsymbol{b}$ the least squares solution is defined as

$$\Sigma_{lsq} = \{\, x \mid A^T A x = A^T b \text{ for some } A \in \boldsymbol{A}, b \in \boldsymbol{b} \,\}.$$

Such an approach can be found in [138] or [191] . It is easily seen that

$$\square(\Sigma) \;\subseteq\; \square(\Sigma_{lsq}).$$

Hence an enclosure of the set Σ_{lsq} is also an enclosure of the set Σ. For more information about this approach and relationship between Σ and Σ_{lsq} see [138]. The question is how to enclose Σ_{lsq}. The first idea is to solve the *interval normal equation*

$$\boldsymbol{A}^T \boldsymbol{A} x = \boldsymbol{A}^T \boldsymbol{b}.$$

This approach might not work because interval matrix multiplication can cause a huge overestimation (see Example 3.9; however later in Chapter 9 we will see that this approach can be used in some special cases). Even a use some preconditioner C

$$(C\boldsymbol{A})^T (C\boldsymbol{A}) x = (C\boldsymbol{A})^T \boldsymbol{b},$$

does often not work either. Anyway, we can use an equivalent expression for the least squares formula (again see [138, 191])

$$\begin{pmatrix} I & \boldsymbol{A} \\ \boldsymbol{A}^T & 0 \end{pmatrix} \begin{pmatrix} y \\ x \end{pmatrix} = \begin{pmatrix} \boldsymbol{b} \\ 0 \end{pmatrix}.$$

Such a system is a square system and hence the methods from the previous chapter can be applied. After computing an enclosure $(\boldsymbol{y}, \boldsymbol{x})^T \in \mathbb{IR}^{m+n}$ of its solution set, the second part $\boldsymbol{x}$ is an enclosure of Σ_{lsq}.

Since the returned interval vector contains the solution of the interval least squares, this method returns a nonempty enclosure even if the original system is unsolvable. Another drawback is that if the original system is of size $m \times n$ we have to solve a new one of size $(m+n) \times (m+n)$. That is why we often refer to this method as *supersquare* approach.

Example 6.4. For the overdetermined system $\boldsymbol{A}x = \boldsymbol{b}$ with

$$\boldsymbol{A} = \begin{pmatrix} [-0.8, 0.2] & [-20.1, -19.5] \\ [-15.6, -15.2] & [14.8, 16.7] \\ [18.8, 20.1] & [8.1, 9.5] \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} [292.1, 292.7] \\ [-361.9, -361.1] \\ [28.4, 30.3] \end{pmatrix},$$

the solution set, the hull of the original system, the hull of the supersquare system and the hull of the interval normal equation are displayed in Figure 6.1.

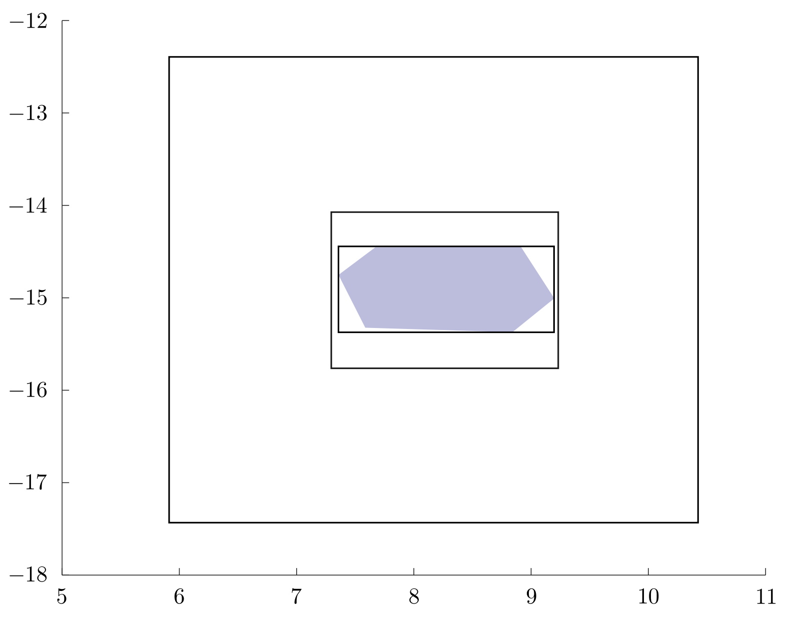


Figure 6.1: The interval least squares. Dark area is the solution set of the original system, the smallest rectangle is the hull of the original system, the intermediate rectangle is the hull of the supersquare system and the largest rectangle is the hull of the interval normal equation.

When solving a system by the supersquare approach, the new matrix is symmetric, hence dependencies occur in the new system (each interval coefficient from the original system is used twice in the new system, that is why when we choose one number from the first interval we should choose the same value in the second one to avoid overestimation). Here, methods dealing with dependencies between coefficients in interval linear systems could be used (e.g., [67, 152, 196]).

6.3 Preconditioning of an overdetermined system

Also an overdetermined system needs often to be transformed to a form that avoids expansion of intervals. It is achieved by multiplication with a preconditioner C of a corresponding size.

$$\boldsymbol{A}x = \boldsymbol{b} \quad \mapsto \quad C\boldsymbol{A}x = C\boldsymbol{b}.$$

A choice of C is proposed in [60]. Let $\boldsymbol{A}$ be of size $m \times n$. Transform its midpoint A^c into an upper trapezoidal form, Gaussian elimination with rounded arithmetics can be applied since we do not need exact result. The same elimination operations are simultaneously performed on an identity matrix of order m. Such a matrix is then taken as a preconditioner.

In [218] there is yet another slightly different possibility for preconditioning an overdetermined system by

$$C \approx \begin{pmatrix} A_1^c & 0 \\ A_2^c & I \end{pmatrix}^{-1}, \tag{6.1}$$

where A_1^c consists of the first n rows of A^c and A_2^c consists of the remaining $m-n$ rows of A^c, 0 is the $(m-n) \times n$ matrix of all zeros and I is the identity matrix of size $(m-n) \times (m-n)$. These preconditioners were designed for use with interval Gaussian elimination, that is why they might not be suitable for all methods. Later we will see that some methods use their own preconditioners (e.g., Rohn's method in Section 6.6). If not stated otherwise, when using a preconditioner for an overdetermined system, we prefer the second choice, since it is a generalization of the midpoint inverse preconditioning for square systems.

After an overdetermined system is preconditioned with C, the center of the resulting matrix is approximately of the shape

$$\begin{pmatrix} I \\ 0 \end{pmatrix},$$

where I is the $n \times n$ identity matrix and 0 is the $(m-n) \times n$ matrix of all zeros. The reasoning is illustrated by the Figure 6.2.

6.4 Gaussian elimination

Interval Gaussian elimination for overdetermined systems was proposed by Hansen in [60]. The idea is pretty the same as for square interval systems: rows are eliminated in the same way as explained in Section 5.6.1. The only difference is that the matrix $(\boldsymbol{A} \mid \boldsymbol{b})$ of size $m \times (n+1)$ corresponding to $\boldsymbol{A}x = \boldsymbol{b}$ is eliminated into the following shape:

$$(\boldsymbol{A} \mid \boldsymbol{b}) \quad \mapsto \quad \left(\begin{array}{cc|c} \boldsymbol{C} & \boldsymbol{d} & \boldsymbol{e} \\ 0 & \boldsymbol{u} & \boldsymbol{v} \end{array}\right),$$

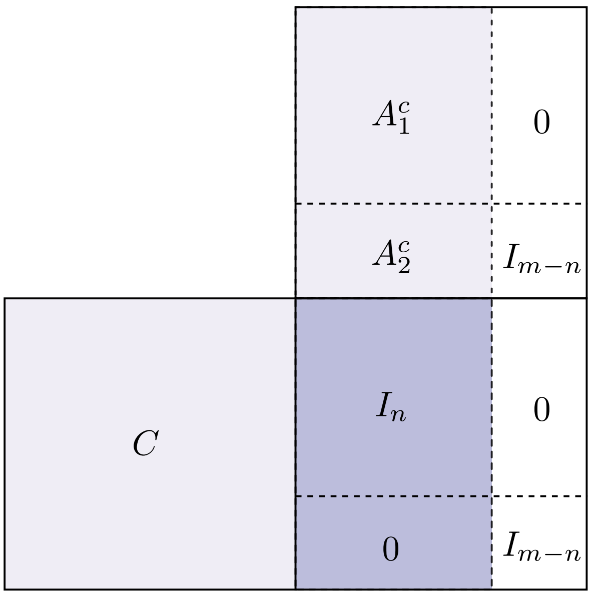


Figure 6.2: Illustration of the preconditioning by C computed by (6.1). The darkest area corresponds to the midpoint matrix of the preconditioned system.

where $\boldsymbol{C}$ is an $(n-1)\times(n-1)$ interval matrix in row echelon form, $\boldsymbol{d}, \boldsymbol{e}$ are $(n-1)\times 1$ interval vectors, 0 is an $(m-n+1)\times(n-1)$ matrix of all zeros and $\boldsymbol{u}, \boldsymbol{v}$ are $(m-n+1)\times 1$ interval vectors.

The vectors $\boldsymbol{u}, \boldsymbol{v}$ form $m-n+1$ interval equations in the shape

$$\boldsymbol{u}_i x_n = \boldsymbol{v}_i \quad \text{for } i = 1, \ldots, (m-n+1).$$

The solution of these equations gives the following enclosure for the variable x_n

$$\boldsymbol{x}_n = \bigcap_{i:\, 0 \notin \boldsymbol{u}_i} (\boldsymbol{v}_i / \boldsymbol{u}_i)\,.$$

If the intersection is empty, then the system has no solution. Nonetheless, if the intersection is unbounded, it can either mean that the solution set of the system is unbounded or huge overestimation due to large number of interval operations occurred. The enclosures for the other variables can be obtained using the backward substitution as described in Section 5.6.1.

6.5 Iterative methods

In the previous chapter we introduced three iterative methods for solving square interval linear systems – the Jacobi, the Gauss–Seidel and Krawczyk's method. After we apply preconditioning from Section 6.3 only the first n rows possibly do not contain zeros on the diagonal. That is why we can apply an iterative method to the square subsystem consisting of the first n equations of the preconditioned system. The solution set of the original overdetermined interval system must lie inside the solution set of a square subsystem (*subsquare*).

Proposition 6.5. *Let Σ be the solution set of $\boldsymbol{A}x = \boldsymbol{b}$ and let Σ_{subs} be the solution set of a square subsystem of $\boldsymbol{A}x = \boldsymbol{b}$. Then*

$$\Sigma \subseteq \Sigma_{subs}.$$

Proof. The original system $\boldsymbol{A}x = \boldsymbol{b}$ has more equations that can put no or more restriction on the solution set of the square subsystem. □

6.6 Rohn's method

We would like to mention the method introduced by Rohn [174]. In his paper more information and theoretical insight can be found. The following theorem is the basis of the method.

Theorem 6.6. *Let $\boldsymbol{A}x = \boldsymbol{b}$ be an overdetermined interval linear system with $\boldsymbol{A}$ being an $m \times n$ interval matrix and Σ being its solution set. Let R be an arbitrary real $n \times m$ matrix, let x_0 and $d > 0$ be arbitrary n-dimensional real vectors such that*

$$Gd + g < d, \tag{6.2}$$

where

$$G = |I - RA_c| + |R|A_\Delta,$$

and

$$g = |R(A_c x_0 - b_c)| + |R|(A_\Delta |x_0| + b_\Delta).$$

Then

$$\Sigma \subseteq [x_0 - d, x_0 + d].$$

The question is how to find the vector d, the matrix R and the vector x_o. To compute d, we can, for example, rewrite the inequality (6.2) as

$$d = Gd + g + \varepsilon, \tag{6.3}$$

for some small vector $\varepsilon > 0$. Then, start with $d = 0$ and iteratively refine d. This algorithm will stop after a finite number of steps if $\varrho(G) < 1$ holds.

In [82] we proposed another option for finding d. One can rewrite the equality 6.3) as

$$(I - G)d = g + \varepsilon,$$

Table 6.1: Rohn's method – testing of the iterative and direct approach for finding d from Example 6.7. The second column displays the average ratio of the vectors d returned by the two methods computed by the formula (6.4), the last two columns display the average computation times, $m \times n$ is the size of a system matrix.

$m \times n$	rat	t iterative	t direct
5 × 3	1.0000	0.0047	0.0012
15 × 10	1.0000	0.0067	0.0024
25 × 21	1.0000	0.0068	0.0023
35 × 23	1.0000	0.0066	0.0024
50 × 35	1.0000	0.0067	0.0024
73 × 55	1.0000	0.0072	0.0024
100 × 87	1.0000	0.0077	0.0028
200 × 170	1.0000	0.0066	0.0024

and solve the real system directly. After finding a solution, the vector d is tested for positivity. In the two following examples we test the two methods on random overdetermined systems.

Example 6.7. A solvable random overdetermined system is generated in the following way. First, a midpoint matrix A_c is generated by uniformly randomly and independently choosing its coefficients from interval $[-10, 10]$. Second, a random solution vector x is generated also with coefficients from interval $[-10, 10]$. The right-hand side b_c is then computed as $b = A_c x$. An interval system is obtained by wrapping the A_c, b_c with intervals having a fixed radius r. Here, we test the systems for $r = 10^{-3}$. The small positive vector ε has its coefficients 10^{-6}. The iteration limit is 50. The results of the comparison are shown in Table 6.1. The second column shows the average ratios of d^{it}, d^{dir} returned by the iterative and direct method respectively. The ratio is computed as average ratio of the coefficients of the two n-dimensional vectors

$$\text{rat} = \left(\sum_{i=1}^{n} \frac{d_i^{it}}{d_i^{dir}} \right) / n. \tag{6.4}$$

For each size we test on 100 random systems. The ratios in Table 6.1 show that both methods return basically identical results. The next two columns show average computation times (using the LAPTOP setting). Even thought, the measured times are very small, the results are in favor of the direct method. The results, however, depend on the method used for solving real linear systems (here we used Octave's `linsolve`). In conclusion, we rather prefer the direct method for computation of d because this way we do not have to care about properly setting ε.

We still have to determine x_0 and R. Rohn recommends to take

$$x_0 \approx Rb_c, \quad R \approx (A_c^T A_c)^{-1} A_c^T,$$

but not necessarily. Rohn suggests that Theorem 6.6 provides an instrument for iterative improvement of enclosure. We do not have to use only A_c to compute R, we can take any (e.g., random) matrix $A \in \boldsymbol{A}$, compute an enclosure and then intersect it with the old one. We can repeat this process as many times we want and provide an iterative improvement of the enclosure.

Example 6.8. For the overdetermined system from Example 6.4 we selected 20 random $A \in \boldsymbol{A}$ to compute $R \approx (A^T A)^{-1} A^T$. We also included $A = A_c$. For this system $A = A_c$ plays a prominent role, since for no other A was the enclosure better. The resulting boxes are displayed in Figure 6.3.

When the random systems were generated as in Example 6.7 for the same sizes, and the first enclosure was computed using $R \approx (A_c^T A_c)^{-1} A_c^T$, then no improvement of the enclosure was detected after 50 such iterations. Hence, it seems that testing other choices of R does not pay off in this case.

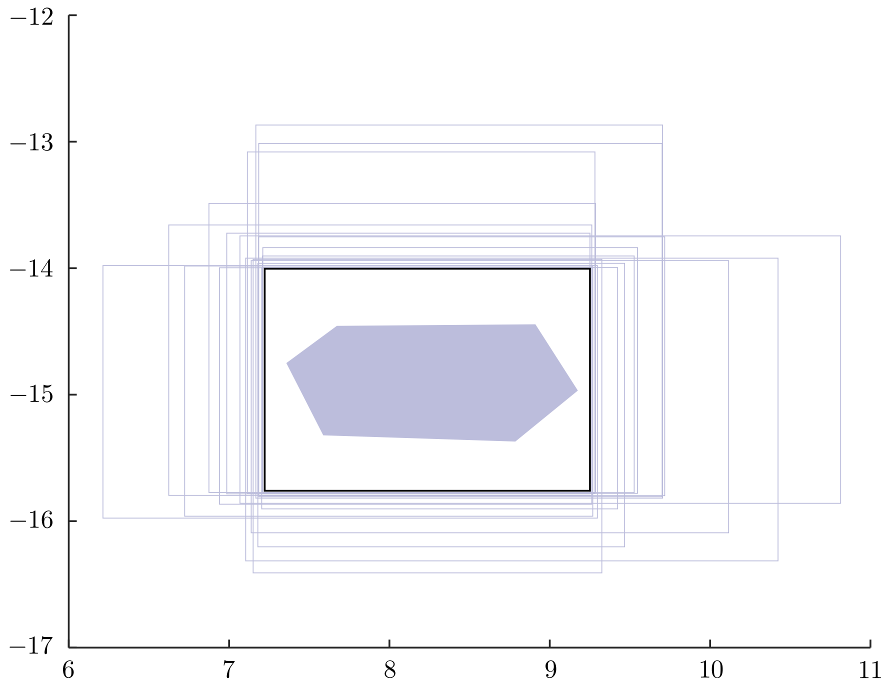


Figure 6.3: Result of Rohn's algorithm from Example 6.8 for various selections of $R \approx (A^T A)^{-1} A^T$ for $A \in \boldsymbol{A}$. The dark area is the solution set of the system. The darkest rectangle is the enclosure for $R \approx (A_c^T A_c)^{-1} A_c^T$. Other 20 enclosures (boxes) correspond to 20 random choices of $A \in \boldsymbol{A}$.

6.7 Comparison of methods

In this section the previously described methods for solving overdetermined systems are compared. First, we start with a simple examples revealing that the methods do not return empty enclosure for an unsolvable system. Next we compare the methods on random overdetermined systems with various fixed radii of interval coefficients. We are aware that it is not completely fair to compare direct and iterative methods together. However, a comparison will give us at least a hint of the properties of such methods. The tested methods are:

- `rohn` – Rohn's method for overdetermined systems with direct computation of d,
- `jacobi` – preconditioned system with the Jacobi iterative method implemented in matrix multiplication form applied to the first n equations, with maximum number of iterations set to 20, and $\varepsilon = 10^{-5}$,
- `lsq` – enclosing the least squares solution, by transforming a system into a supersquare and then solving with the HBR mehod,
- `ge` – Gaussian elimination for overdetermined systems with preconditioning and mignitude pivoting.

When a method name has the suffix `-pre`, it means that it is applied without preconditioning, the suffix `+pre` means the preconditioning with midpoint inverse was used. First let us test the methods for an unsolvable system.

Example 6.9. Let us have the unsolvable system with the following A_c, b_c and fixed radii of interval coefficients set to $r = 0.1$.

$$A_c = \begin{pmatrix} -6 & 2 & -9 \\ 0 & 8 & 6 \\ 7 & -9 & -5 \\ 4 & -5 & -8 \\ -5 & -7 & 6 \end{pmatrix}, \quad b_c = \begin{pmatrix} 9 \\ 54 \\ -120 \\ -95 \\ 57 \end{pmatrix}.$$

The returned enclosures are in Table 6.2. We can see that no method can detect unsolvability of the system.

Table 6.2: Comparison of enclosures returned by various methods applied on the unsolvable system from Example 6.9.

$\boldsymbol{x}$	$\boldsymbol{x}_1$	$\boldsymbol{x}_2$	$\boldsymbol{x}_3$
`rohn`	$[-9.4682, -8.6938]$	$[2.6762, 3.2171]$	$[5.2755, 5.7940]$
`jacobi+pre`	$[-10.804, -9.2545]$	$[1.4635, 2.8699]$	$[5.5646, 6.7552]$
`lsq`	$[-9.4951, -8.6841]$	$[2.6655, 3.2364]$	$[5.2681, 5.8091]$
`ge+pre`	$[-10.404, -7.9421]$	$[2.6365, 3.6731]$	$[5.0720, 5.8993]$
`ge-pre`	$[-10.804, -9.2545]$	$[1.4653, 2.8699]$	$[5.5671, 6.7552]$
`hull`	$\emptyset$	$\emptyset$	$\emptyset$

Table 6.3: Overdetermined interval linear systems – average ratios of enclosures returned by various methods compared to the hull, $m \times n$ is the size of a system matrix.

$m \times n$	`rohn`	`jacobi`	`lsq`	`ge`
5×3	1.114	7.961	1.114	7.961
15×13	1.038	4.538	1.039	4.538
35×23	1.116	14.963	1.116	14.962
50×35	1.101	11.946	1.101	11.945
100×87	1.043	11.043	1.047	17.562

Next, all the methods are compared on 100 random solvable systems for each size. Such systems were generated by using the same procedure as described in Example 6.7. The radii of interval coefficients were fixed to $r = 10^{-4}$. The average ratios of enclosures are compared using the formula (3.9). They are displayed in Table 6.3 and the average computation times are in Table 6.4. For such a "small" radii, the returned enclosures lie in one orthant, that is why we compared the quality of enclosures to the hull. The hull was computed as in Section 5.2, however, because of computation time reasons, using only nonverified linear programming. The results must be hence taken with caution, nevertheless, empirically for such systems a nonverified hull is nearly identical to the verified hull.

The `ge` and `jacobi` return comparable enclosures. So do `rohn` and `lsq`. We believe it is no coincidence. The way `jacobi` and `ge` are defined for overdetermined systems suggest that only the first n rows are basically used for computing an enclosure. We explain the similarity of `rohn` and `jacobi` by the use of the matrix R in `rohn`. Since it is basically the Moore-Penrose pseudoinverse of A_c the solution set Σ of the preconditioned system and Σ_{lsq} tend to coincide for small radii. In Table 6.5 we show that this is not the case for larger radii.

The `rohn` is the winner from computation time perspective, since the disadvantage of need to solve much larger supersquare system in `lsq` manifests itself. The `ge` shows excessive demands regarding computational time. In all cases methods returned finite enclosures, except for the size 100×87 `jacobi` returned infinite enclosures in 5 cases, `ge` in 2 cases.

6.8 Subsquares approach

In this section we present a scheme for solving overdetermined systems, which we developed in [84]. This method uses algorithms described in Chapter 5 and applies them on selected square subsystems of the original system. Although, we mentioned the term "subsquare" earlier in Section 6.5, we rather define it more formally here.

Definition 6.10 (Subsquare)**.** By a *square subsystem* or *subsquare* of an overdetermined system $\boldsymbol{A}x = \boldsymbol{b}$, where $\boldsymbol{A}$ is of size $m \times n$, we mean any choice of n equations

Table 6.4: Overdetermined interval linear systems – average computation times in seconds for various methods, $m \times n$ is the size of a system matrix.

$m \times n$	`rohn`	`jacobi`	`lsq`	`ge`
5×3	0.011	0.065	0.042	0.091
15×13	0.011	0.078	0.077	0.919
35×23	0.012	0.088	0.147	4.273
50×35	0.013	0.107	0.237	9.032
100×87	0.020	0.280	0.992	39.989

Table 6.5: Enclosures returned by `lsq` compared to enclosures by `rohn`. The symbol `'-'` means that no finite enclosure was returned by `lsq`, the symbol `'--'` means that no finite enclosure was returned by both methods, $m \times n$ is the size of a system matrix.

$m \times n$	$r = 0.01$	$r = 0.1$
5×3	1.005	1.061
15×13	1.054	3.502
35×23	1.061	18.721
50×35	1.103	-
100×87	2.140	--

(without repetition) from the original m ones.

Note that the original solution set lies in the solution set of each subsquare (see Proposition 6.5). For the sake of simplicity we will denote the square subsystem of $\boldsymbol{A}x = \boldsymbol{b}$ created by equations $i_1, i_2, \ldots, i_n$ as $\boldsymbol{A}_{\{i_1,i_2,\ldots,i_n\}}x = \boldsymbol{b}_{\{i_1,i_2,\ldots,i_n\}}$. When we use some order (e.g., dictionary order) of subsquares (here it does not depend which one) the jth square subsystem will be denoted by $\boldsymbol{A}_j x = \boldsymbol{b}_j$. Examples of subsquares can be seen in Example 6.11.

Example 6.11. Let us take again the system from Example 6.4. There are three possible subsquares:

$$\boldsymbol{A}_{\{1,2\}} = \begin{pmatrix} [-0.8, 0.2] & [-20.1, -19.5] \\ [-15.6, -15.2] & [14.8, 16.7] \end{pmatrix}, \quad \boldsymbol{b}_{\{1,2\}} = \begin{pmatrix} [292.1, 292.7] \\ [-361.9, -361.1] \end{pmatrix}.$$

$$\boldsymbol{A}_{\{1,3\}} = \begin{pmatrix} [-0.8, 0.2] & [-20.1, -19.5] \\ [18.8, 20.1] & [8.1, 9.5] \end{pmatrix}, \quad \boldsymbol{b}_{\{1,3\}} = \begin{pmatrix} [292.1, 292.7] \\ [28.4, 30.3] \end{pmatrix}.$$

$$\boldsymbol{A}_{\{2,3\}} = \begin{pmatrix} [-15.6, -15.2] & [14.8, 16.7] \\ [18.8, 20.1] & [8.1, 9.5] \end{pmatrix}, \quad \boldsymbol{b}_{\{2,3\}} = \begin{pmatrix} [-361.9, -361.1] \\ [28.4, 30.3] \end{pmatrix}.$$

Their solution sets and hulls are depicted in Figure 6.4. Notice that the intersection of hulls/enclosures of subsquares tends to the hull of the original system. When

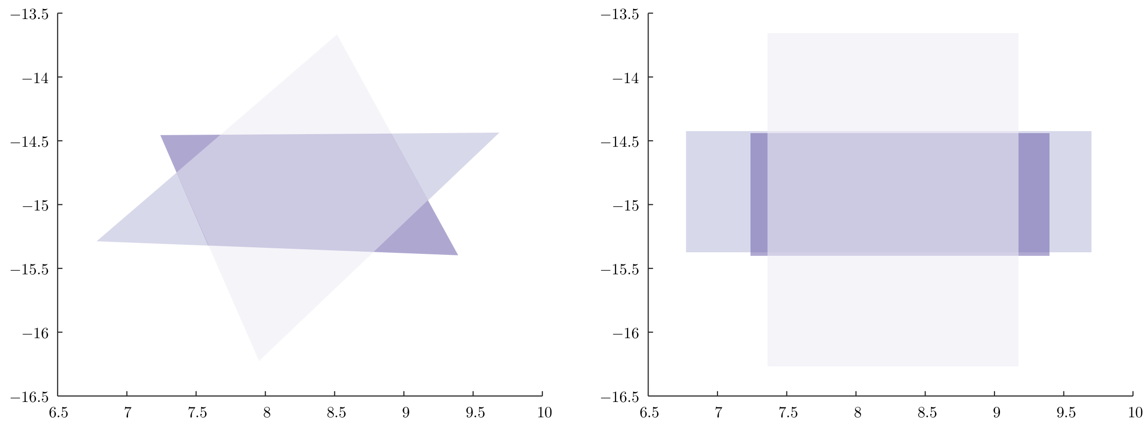


Figure 6.4: The subsquares from Example 6.11 – on the left there are the solution sets, on the right there are the hulls of $\boldsymbol{A}_{\{1,2\}}x = \boldsymbol{b}_{\{1,2\}}$ (the intermediate color), $\boldsymbol{A}_{\{1,3\}}x = \boldsymbol{b}_{\{1,3\}}$ (the darkest color) and $\boldsymbol{A}_{\{2,3\}}x = \boldsymbol{b}_{\{2,3\}}$ (the lightest color).

some subsquares of an overdetermined system are chosen, the intersection of their solution enclosures provides hopefully tighter enclosure on the original solution set. The enclosures of all subsquares computed using the HBR method are depicted in Figure 6.5. It can be seen that the intersection of enclosures is indeed close to the original hull.

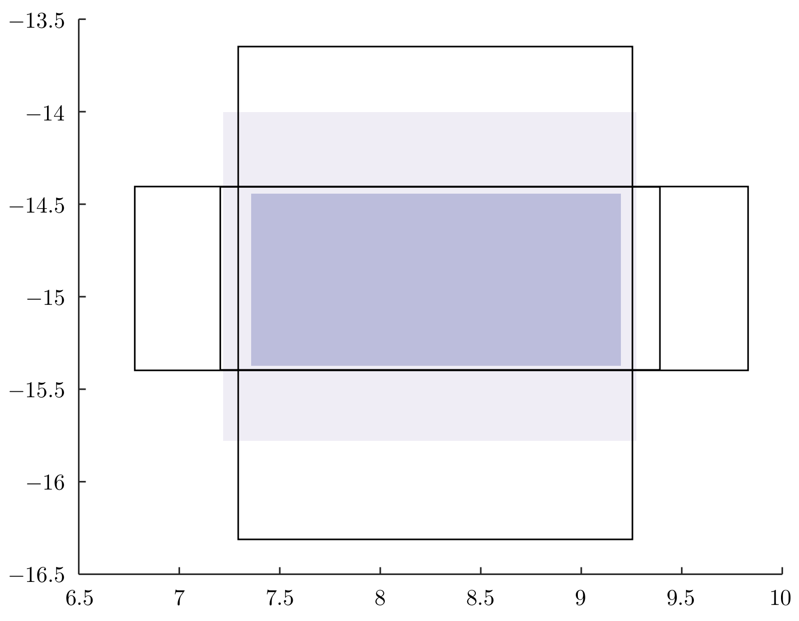


Figure 6.5: The enclosures of the subsquares from Example 6.11. Rectangles represent enclosures of subsquares computed by the HBR method. The darkest area represents the hull of the original overdetermined system, the lighter rectangle is an enclosure computed by Rohn's method.

6.8.1 Simple algorithm

If we compute enclosures of square subsystems separately and then intersect the resulting enclosures, we get the simple Algorithm 6.12.

Algorithm 6.12. (Simple subsquares) Input is an overdetermined system $\boldsymbol{A}x = \boldsymbol{b}$. The algorithm returns an enclosure $\boldsymbol{x}$ of its solution set.

1. Select k random subsquares $\boldsymbol{A}_i x = \boldsymbol{b}_i$ for $i \in \{1, \ldots, k\}$.
2. Compute enclosures of all subsquares $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$.
3. Intersect the enclosures, i.e., return the enclosure $\boldsymbol{x} := \bigcap_{i=1}^{k} \boldsymbol{x}_i$.
4. If $\boldsymbol{x}_i \cap \boldsymbol{x}_j = \emptyset$ for some two $i \neq j$ ($\boldsymbol{x}$ is empty), then the original system is not solvable.

Such an approach is a little naive, but it has its advantages. First, if we compute enclosures of all possible square subsystems, we may, as the Figure 6.4 suggests, expect getting close to the interval hull.

Example 6.13. The enclosure obtained by intersecting enclosures of all subsquares is compared to the interval hull of the original system. To compute the hull we used the procedure described in Section 5.2. For computation time reasons only 10 systems were generated for each size. The systems were again generated in the same way as in Example 6.7. To spare time we used only a nonverified linear programming (Octave `glpk` method). Hence, the results should taken with caution, however experience shows that the nonverified hull is for systems generated in such a way close to the verified hull. The Table 6.6 shows the results for random examples of systems.

If we have an $m \times n$ system, the number of all square subsystems is equal to $\binom{m}{n}$. However, we can see that for n small or for n close to m the number $\binom{m}{n}$ may not be so large. The low computational time emerges when a system is noodle-shaped or nearly-square. However for a nearly-square systems there are not enough equations to plausibly form subsquares that could shave the intersecting enclosure well.

The second advantage is that Algorithm 6.12 can be made faster by incorporating parallelism – solving of one subsquare system does not depend on the others. The third advantage is that Algorithm 6.12 can, in contrast to other methods, often decide whether a system is unsolvable – if an intersection of enclosures of some two subsquares is empty, then the whole overdetermined system is unsolvable. We test the number of subsquares needed to detect unsolvability in Chapter 7.

For most rectangular systems it is however not convenient to compute enclosures of all or many square subsystems. The selection of subsquares and the solving algorithm can be modified to be less time consuming.

6.8.2 Selecting less subsquares

We wish to have a method that returns sharp enclosures, can reveal unsolvability and is parallelizable. All can be done by the simple algorithm. However, there is a problem

Table 6.6: Simple subsquares method solving all subsquares compared to the non-verified hull – enclosures comparison (Example 6.13). The second and third column shows the average ratio of subsquares method to the unverified hull. The last column shows the average computation time of `subsq` method. Various system matrix sizes $m \times n$ and radii r were tested.

$m \times n$	$r = 0.01$	$r = 0.0001$	time `subsq`
5×3	1.0067	1.0001	0.3 s
9×5	1.0115	1.0001	4.2 s
13×7	1.0189	1.0002	1 m 2 s
15×9	1.0248	1.0003	3 m 17 s
25×21	1.0926	1.0011	12 m 56 s
30×29	1.4522	1.0022	2.4 s

– extremely long computation time for a general overdetermined system. For solving by subsquares method we definitely need to choose less subsquares. Here are some desirable properties of the set of selected subsquares:

1. We do not want to have too many subsquares.
2. We want each equation in the overdetermined system to be covered by at least one subsquare.
3. The overlap of subsquares (equations shared by any two subsquares) must not be too low, nor too high.
4. We select subsquares that narrow the resulting enclosure as much as possible.

We can select subsquares randomly, but then we do not have the control over this selection. This works fine, however, it is not clear how many subsquares should we choose according to the size of the overdetemined system. Moreover, experiments have shown that it is advantageous when subsquares overlap. That is why we propose a different strategy.

The first and second property can be settled by covering the system with subsquares step by step using some overlap parameter. About the third property, experiments show that taking the consecutive overlap $\approx n/3$ is a reasonable choice. Property four is a difficult task to handle. We think deciding which systems to choose (in a favourable time) is still an area to be explored. Yet random selection will serve us well.

Among many possibilities we tested, the following selection of subsystems worked well. During the selection algorithm we divide numbers of equations of an overdetermined system into two sets – `Covered`, which contains equations that are already contained in some subsquare, and `Waiting`, which contains equations that are not covered yet. We also use a parameter `overlap` to define the overlap of two subsequent subsquares.

The first subsystem is chosen randomly, other subsystems will be composed of `overlap` equations with indices from `Covered` and ($n-$`overlap`) equations with indices from `Waiting`. The last system is composed of all remaining uncovered equations and then some already covered equations are added to form a square system. The selection procedure is described in Algorithm 6.14. The algorithm implementation is not necessarily optimal, it should serve as an illustration. The procedure randsel(n, S) selects n random nonrepeating numbers from a set S. The total number of subsquares selected by this algorithm is

$$1 + \left\lceil \frac{m-n}{n - overlap} \right\rceil . \tag{6.5}$$

Algorithm 6.14 (Selecting subsquares)**.** Algorithm takes an overdetermined system $\boldsymbol{A}x = \boldsymbol{b}$ with m equations and n variables. Algorithm chooses a suitable set of subsquares stored in variable `Subsquares`.

1. Set `Subsquares` $:= \emptyset$, `Covered` $:= \emptyset$ and `Waiting` $:= \{1, 2, \ldots, m\}$.
2. While `Waiting` $\neq \emptyset$ repeat the following steps.
3. At the beginning (if `Covered` $= \emptyset$), set `Indices` $:=$ randsel(n, `Waiting`).
4. At the end (if $|$`Waiting`$| \leq (n -$ `overlap`$)$) set

   `Indices` $:=$ `Waiting` $\cup$ randsel($n - |$`Waiting`$|$, `Covered`).

5. Otherwise, set

   `Indices` $:=$ randsel(`overlap, Covered`) $\cup$ randsel($n -$ `overlap`, `Waiting`).

6. Add the subsquare $\boldsymbol{A}_{\texttt{Indices}} x = \boldsymbol{b}_{\texttt{Indices}}$ to `Subsquares`.
7. Update `Covered` $:=$ `Covered` $\cup$ `Indices`.
8. Update `Waiting` $:=$ `Waiting` $\setminus$ `Indices`.

6.8.3 Solving subsquares – the multi-Jacobi method

The only thing left is to solve the selected subsquares. The first obvious choice is to solve each subsquare separately and then intersect the enclosures as in the case of the simple algorithm 6.12.

In [84] we proposed a different strategy. We use the Jacobi method for solving each subsquare. Nevertheless, the subsquares are not solved completely but only one Jacobi iteration is applied to all subsquares. After the iteration is completed, the global enclosure is updated (by intersection). Then, the second iteration is applied to each subsquare and so on. Let us call this method the *multi-Jacobi* method.

The following example shows, that the multi-Jacobi method is more efficient than the simple subsquares approach.

Table 6.7: Random subsquares compared to the multi-Jacobi method – average ratios of enclosures (Example 6.15). Random subsquares are compared to the multi-Jacobi method, each column corresponds to a fixed radius r of intervals, the symbol '-' means the methods did not return finite enclosure for any of the systems, $m \times n$ is the size of a system matrix.

$m \times n$	$r = 0.0001$	$r = 0.001$	$r = 0.01$
5 × 3	1.85	1.41	1.60
15 × 13	1.57	1.41	1.53
35 × 23	1.66	1.89	2.66
50 × 35	1.75	1.85	1.83
100 × 87	2.26	1.60	-

Example 6.15. We compare the multi-Jacobi method with the simple subsquares method. The HBR method is used to solve subsquares of the second method. Initial enclosure of both methods is found as a HBR enclosure of some subsquare. The second method chooses the same number of random square subsystems (according to (6.5)). The random solvable systems are generated in the same way as in Example 6.7. The results are in Table 6.7. The multi-Jacobi method reaches better enclosures with some minor computational time added (Table 6.8). The table shows the average ratios of computation times $\frac{t(multi-Jacobi)}{t(subsquares)}$. The idea behind the success of the multi-Jacobi might be similar to simulated annealing process.

Next, we try to run the multi-Jacobi on the results of the best method from the comparison for overdetermined interval systems – Rohn's method.

Example 6.16. For comparison we choose Rohn's method with direct computation of d. The multi-Jacobi method uses $\varepsilon = 10^{-5}$ for stopping criterion. The results of the comparison are displayed in Table 6.9. We can see that in some cases it can slightly improve the enclosure returned by Rohn's method. As Rohn's method is the best, a large improvement of an enclosure cannot be expected. The computation times for the multi-Jacobi method include computation of Rohn's enclosure.

A larger computation time is not too much of an issue since the multi-Jacobi method can be parallelized. If an enclosure is obtained by some other method the multi-Jacobi method can be used as a second shaver. We need to remind that one advantage of the multi-Jacobi method to Rohn's method is that it can detect unsolvability.

6.9 Other methods

There exist other approaches to solving overdetermined interval systems. Popova inroduced an approach for underdetermined and overdetermined systems that can

Table 6.8: Random subsquares vs. multi-Jacobi method – ratio of computation times $\frac{t(multi-Jacobi)}{t(subsquares)}$ in seconds.

size	$r = 0.0001$	$r = 0.001$	$r = 0.01$
5×3	2.79	2.82	2.92
15×13	1.50	1.72	2.18
35×23	1.40	1.55	2.06
50×35	1.27	1.51	1.90
100×87	1.17	1.19	0.79

Table 6.9: The multi-Jacobi method run on results of Rohn's method – average enclosure ratios and average computation times. The computation times are in seconds, $m \times n$ is the size of a system matrix, fixed radius of intervals is denoted by r, overlap is the parameter for selection of subsquares, the last two columns show average computation times in seconds, computation time of the multi-Jacobi method includes the computation time of Rohn's.

$m \times n$	r	overlap	av. rat time	time Rohn's	time multi-Jacobi
11×7	0.1	2	0.991738	0.0112985	0.0679437
11×7	0.2	2	0.987414	0.011084	0.0610227
11×7	0.3	2	0.985185	0.011123	0.0520721
15×10	0.1	3	0.995979	0.011762	0.0818686
15×10	0.2	3	0.994436	0.0117518	0.0725302
15×10	0.3	3	0.994124	0.0114046	0.0807104
25×13	0.1	3	0.999436	0.0117957	0.344695
25×13	0.2	3	0.998644	0.0118171	0.272701
25×13	0.3	3	0.997601	0.0120146	0.0709837
37×20	0.05	7	0.999795	0.0118177	0.0963902
37×20	0.0001	7	0.99998	0.0117649	0.103442

deal with parametric dependencies between intervals in [154]. A generalization of the Hansen–Bliek–Rohn enclosure for overdetermined systems was done by Rohn in [182]. Underdetermined and overdetermined systems are also discussed in [191].

7 (Un)solvability of interval linear systems

- Methods for detecting unsolvability
- Full column rank
- Scaled maximum norm
- Equivalence of two sufficient conditions for unsolvability
- Methods for detecting solvability
- Comparison of methods

There exist many methods for computing interval enclosures of the solution set of an interval linear system. Nevertheless, many of them return nonempty enclosure even if the system has no solution. In some applications such as system validation or technical computing we do care whether systems are solvable or unsolvable. Moreover, solving a system may be a computationally demanding task, therefore in some cases we want to know ahead whether it is worth trying to solve it.

Unfortunately, checking solvability and unsolvability of am interval linear system are both hard problems; NP-complete and coNP-complete respectively [85, 178]. That is why it would be favorable to have at least some sufficient conditions or algorithms detecting for solvability and unsolvability that are computable in polynomial time. In this chapter, such algorithms and conditions are in the center of focus. Most of them are well-known, but used so far for a different purpose than checking unsolvability. We are going to show how they can be modified to detect unsolvability, what are the relations between them and how strong they are. The two strongest conditions are based on sufficient conditions for an interval matrix having full column rank. Related to the second condition our algorithm for computation of scaled maximum norm is presented. We prove that under a certain assumption these conditions are equivalent. The topic of solvability is also touched. We present two strategies for detecting solvability of an interval linear system. Strength of methods is tested and graphically displayed using heat maps. This chapter is a slightly modified and extended version of our paper [87].

7.1 Definition

Even though, we touched solvablity and unsolvability in Chapter 5, let us remind the definitions and state them more explicitly.

Definition 7.1 (Solvability and unsolvability)**.** If the solution set Σ of $\boldsymbol{A}x = \boldsymbol{b}$ is empty, we call the system *unsolvable*. Otherwise we call it *solvable*.

In another words, when an interval system is unsolvable, then no system $Ax = b$ in $\boldsymbol{A}x = \boldsymbol{b}$ has a solution. Such a solvability concept can be called *weak* solvability. There are other concepts of solvability. An interval system is called *strongly solvable* when each $Ax = b$ in $\boldsymbol{A}x = \boldsymbol{b}$ is solvable. There are other more generalized concepts of solvability [203]. For more details on solvability we refer to [178]. The problem of this chapter is:

Problem: Decide whether $\boldsymbol{A}x = \boldsymbol{b}$ is unsolvable or solvable.

7.2 Conditions and algorithms detecting unsolvability

Let us start with the well-known methods, that are not often used for detecting unsolvability or that can detect unsolvability as a byproduct.

7.2.1 Linear programming

In Section 5.2 we explained how to use verified linear programming in combination with the Oettli–Prager theorem to compute the hull of a solution set. As showed, signs of an initial enclosure of the solution set give a hint which orthants need to be inspected for existence of a solution. If a verified linear programming announces nonexistence of a solution in all suspected orthants, then the system is unsolvable. However, the verified linear programming might not always be able to decide about the existence of a solution in each orthant. Moreover, computation time of this method might be too long and the method requires an implementation of a verified linear programming. That is why we only mention this method for the sake of completeness, and we are not going to compare it against the other methods.

7.2.2 Interval Gaussian Elimination

In Chapter 6 we described the interval version of Gaussian elimination for overdetermined systems. The last $m - n + 1$ rows of the eliminated system are in the following shape:

$$\begin{aligned} \boldsymbol{u}_n x_n &= \boldsymbol{v}_n, \\ \boldsymbol{u}_{n+1} x_n &= \boldsymbol{v}_{n+1}, \\ &\vdots \\ \boldsymbol{u}_m x_n &= \boldsymbol{v}_m, \end{aligned}$$

for some intervals $\boldsymbol{u}_n, \ldots, \boldsymbol{u}_m, \boldsymbol{v}_n, \ldots, \boldsymbol{v}_m$ that occurred during the elimination. Now, the interval enclosure of the solution of the variable x_n is

$$\boldsymbol{x}_n = \bigcap_{i:\, 0 \notin \boldsymbol{u}_i} (\boldsymbol{v}_i / \boldsymbol{u}_i) \,.$$

If such an intersection is empty, then the original system is unsolvable.

Gaussian elimination is often used with preconditioning. When the preconditioning described in Chapter 6 is used, an unsolvable system usually becomes solvable. However, if we do not use preconditioning, interval operations may result in an overestimation anyway, hence we get a nonempty enclosure again, even if the original one was unsolvable. That is why detection of unsolvability by interval Gaussian elimination is suitable only for very small systems. We are going to address sizes of systems that are suitable for this method later.

7.2.3 Square subsystems

The subsquares method described for overdetermined interval systems in Chapter 6 is favorable for certain interval systems. As already mentioned, when the subsquares are selected randomly, enclosure of their solution set is computed using some method described in Chapter 5 and then intersected, occurrence of an empty interval proves empty solution set of the original system. Usually a low number of subsquares is needed to prove unsolvability.

Example 7.2. To generate random interval overdetermined systems we first generated A_c, b_c with coefficients randomly and uniformly from $[-10, 10]$. For sufficiently small radii such systems will be unsolvable. Then, the midpoints were inflated into intervals using defined fixed radius. The average number of subsquares needed to reveal unsolvability of 100 systems for each size and radius of intervals is shown in Table 7.1.

7.2.4 The least squares enclosure

The least squares approach can also used for detecting unsolvability. As usually, an interval system can be viewed as a set of point real systems. First, an enclosure of the all least squares solutions $A^T A x = A^T b$ for all $A \in \boldsymbol{A}, b \in \boldsymbol{b}$ is computed. As already mentioned, possibly the best way to do that is by solving the following interval system

$$\begin{pmatrix} I & \boldsymbol{A} \\ \boldsymbol{A}^T & 0 \end{pmatrix} \begin{pmatrix} y \\ x \end{pmatrix} = \begin{pmatrix} \boldsymbol{b} \\ 0 \end{pmatrix}. \tag{7.1}$$

Table 7.1: Number of random subsquares needed to reveal unsolvability (Example 7.2).

$m \times n$	$r = 10^{-3}$	$r = 10^{-4}$	$r = 10^{-5}$
5 × 3	2.00	2.20	2.05
15 × 13	2.05	2.00	2.05
35 × 23	2.00	2.00	2.00
50 × 35	2.15	2.00	2.00
100 × 87	2.60	2.00	2.00

The enclosure of the all least squares solutions $\boldsymbol{x}$ appears as the last n components of the obtained enclosure. If $0 \notin \boldsymbol{Ax} - \boldsymbol{b}$ we are sure that there is no x, A, b such that $Ax - b = 0$, $A \in \boldsymbol{A}, b \in \boldsymbol{b}$ and the original interval system is not solvable. Another possibility to prove unsolvability, is to check whether $0 \notin \boldsymbol{y}$, where $\boldsymbol{y}$ appears as the first m components of the obtained enclosure [35]. Note that we can also use other before mentioned methods for computing enclosure $\boldsymbol{x}$ of the solution set.

7.3 Full column rank

In this section we define sufficient conditions for detecting unsolvability of an interval linear system based on full column rank.

Definition 7.3 (Full column rank)**.** A matrix $A \in \mathbb{R}^{m\times n}$ has *full column rank* if its rank is equal to the number of its columns, i.e., $\text{rank}(A) = n$. An interval matrix $\boldsymbol{A}$ has full column rank if every $A \in \boldsymbol{A}$ has full column rank.

Let $\boldsymbol{A}x = \boldsymbol{b}$ be an interval linear system. If for every instance $Ax = b$, where $A \in \boldsymbol{A}, b \in \boldsymbol{b}$ the matrix $(\, A \mid b \,)$ has full column rank, then it means that the vector b does not belong to the column space of A and hence the system has no solution (according to the well-known Frobenius theorem). Therefore, the whole interval system $\boldsymbol{A}x = \boldsymbol{b}$ is unsolvable, if $(\, \boldsymbol{A} \mid \boldsymbol{b} \,)$ has full column rank.

Checking whether an interval matrix has full column rank is a coNP-complete problem [85]. Theorem 4.2 can be generalized for rectangular matrices because the proof in [164] can be used with only some minor changes.

Theorem 7.4. *A square interval matrix $\boldsymbol{A}$ has full column rank if for some real matrix R the following condition holds*

$$\varrho(|I - RA_c| + |R|A_\Delta) < 1.$$

Particularly, if A_c has full column rank, for $R = A_c^+$ the condition reads

$$\varrho(|A_c^+| \cdot A_\Delta) < 1. \tag{7.2}$$

Proof. Assume that $\boldsymbol{A} = [A_c - A_\Delta, A_c + A_\Delta]$ does not have full column rank. Then the system $\boldsymbol{A}x = 0$ must have some solution $x \neq 0$. According to the Oettli–Prager theorem

$$|A_c x| \leq A_\Delta |x|$$

holds. Using this we have

$$\begin{aligned} |x| = |(I - RA_c)x + RA_c x| &\leq |I - RA_c||x| + |R||A_c x| \leq \\ &\leq |I - RA_c||x| + |R|A_\Delta|x| \leq (|I - RA_c| + |R|A_\Delta)|x|. \end{aligned}$$

Hence, we get

$$1 \geq \varrho(|I - RA_c| + |R|A_\Delta)$$

by the Perron–Frobenius theorem [139], which is a contradiction. □

Since A_c consists of linearly independent columns we can write

$$A_c^+ = \left(A_c^T A_c\right)^{-1} A_c^T.$$

Next we show that taking $R = A_c^+$ is optimal from some point of view and under specific assumptions. The proof is an adaptation of the analogous proof for square matrices [163].

Theorem 7.5 (Horáček et al., [87])**.** *Assume that $R \in \mathbb{R}^{n\times m}$ is of the form $R = CA_c^+$, where $C \in \mathbb{R}^{n\times n}$ is nonsingular. If*

$$\varrho(|I - RA_c| + |R|A_\Delta) < 1, \tag{7.3}$$

then A_c has full column rank and

$$\varrho(|A_c^+|A_\Delta) \leq \varrho(|I - RA_c| + |R|A_\Delta).$$

Proof. We have

$$\varrho(I - RA_c) \leq \varrho(|I - RA_c|) \leq \varrho(|I - RA_c| + |R|A_\Delta) < 1. \tag{7.4}$$

Thus, RA_c is nonsingular and A_c has full column rank. Again, in this case $A_c^+ = ({A_c}^T A_c)^{-1} A_c^T$ and $A_c^+ A_c = I$. Now, define

$$G := |I - RA_c| + |R|A_\Delta + \varepsilon e e^T, \quad \alpha := \varrho(G) < 1,$$

where $\varepsilon > 0$ is small enough. Such ϵ exists due to continuity of the spectral radius [91, 126]. Since $G > 0$, by Perron–Frobenius Theorem there exists $0 < x \in \mathbb{R}^n$ such that $Gx = \alpha x$. Using the fact that $\alpha < 1$, we derive

$$\alpha|I - RA_c|x \leq |I - RA_c|x \leq |I - RA_c|x + |R|A_\Delta x < \alpha x,$$

and when we combine the last and then the first inequality we get

$$|R|A_\Delta x < \alpha(I - |I - RA_c|)x.$$

By (7.4) and the Neumann series theory, $I - |I - RA_c|$ has a nonnegative inverse, which yields

$$(I - |I - RA_c|)^{-1}|R|A_\Delta x < \alpha x.$$

Now, from

$$\begin{aligned} A_c^+ &= (CI)^{-1}CA_c^+ = (CA_c^+A_c)^{-1}CA_c^+ = (RA_c)^{-1}R \\ &= (I - (I - RA_c))^{-1}R = \sum_{i=1}^{\infty}(I - RA_c)^i R \end{aligned}$$

we derive

$$|A_c^+| \leq \sum_{i=1}^{\infty} |I - RA_c|^i |R| = (I - |I - RA_c|)^{-1}|R|.$$

Putting all together, we obtain

$$|A_c^+|A_\Delta x \leq (I - |I - RA_c|)^{-1}|R|A_\Delta x < \alpha x.$$

By Perron–Frobenius theory, $\varrho(|A_c^+|A_\Delta) < \alpha < 1$, from which the statement follows. □

Even though the assumption on R of the above theorem is quite restrictive, it covers a lot of natural choices: not only the pseudoinverse $R = A_c^+$, but also $R = A_c^T$ and their multiples, among others. The following example shows that A_c^+ is not the best preconditioner in general.

Example 7.6. Let

$$A_c = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix}, \quad A_\Delta = \frac{1}{4}\begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} -1.5385 & 0.0769 & 0.4615 \\ 1.2404 & 0.0192 & -0.2596 \end{pmatrix}.$$

Then $\varrho(|A_c^+|A_\Delta) \approx 1.04167 > 1$, so full column rank of $\boldsymbol{A}$ is not confirmed yet. However, using the sufficient condition for full column rank of $\boldsymbol{A}$ in Theorem 7.4, we get

$$\varrho(|I - RA_c| + |R|A_\Delta) \approx 0.89937 < 1,$$

confirming full column rank of $\boldsymbol{A}$.

In the further text, many of our results will be in terms of matrix norms. We will use only *consistent* matrix norms i.e, those that satisfy

$$\|A \cdot B\| \leq \|A\| \cdot \|B\|$$

for real matrices (or vectors) A, B of the corresponding size. The norms mentioned in Chapter 3 are all consistent. In [181] the following theorem for real matrices can be found. Here, we prove a stronger version.

Theorem 7.7. *Let $A \in \mathbb{R}^{m\times n}$. There exists a matrix $R \in \mathbb{R}^{n\times m}$ such that for an arbitrary consistent matrix norm $\|\cdot\|$ the inequality*

$$\|I - RA\| < 1$$

holds, if and only if A has full column rank.

Proof. ($\Leftarrow$) This implication is rather simple. If A has full column rank then A^TA is regular and therefore by setting $R = (A^TA)^{-1}A^T$ we obtain $RA = I$. Therefore, $\|I - RA\| = 0 < 1$.

($\Rightarrow$) Let there be a matrix $R \in \mathbb{R}^{n\times m}$ such that

$$\|I - RA\| < 1.$$

Using the well-known relation between the spectral radius and a consistent norm [126] we get

$$\varrho(I - RA) \leq \|I - RA\| < 1.$$

Hence, $I - RA$ has all its eigenvalues located somewhere within a circle with the center 0 and radius < 1. Adding I to the matrix $(-RA)$ shifts all its eigenvalues to the right by 1. The eigenvalues of $(-RA)$ are located within a circle with the center -1 and radius < 1. This circle does not intersect 0, therefore, no eigenvalue can be 0 and therefore $(-RA)$ and also (RA) are nonsingular. This implies that A must have full column rank otherwise RA would be singular. □

The remaining question is how to choose R. The matrix R can be set as an approximate pseudoinverse matrix of A (e.g., by using `pinv` function in Matlab/Octave).

The more important implication of Theorem 7.7 holds also for interval matrices. The proof easily follows from the definition of interval matrix norm.

Corollary 7.8. *Let $\boldsymbol{A} \in \mathbb{IR}^{m\times n}$ be an interval matrix. Suppose there exists a real matrix $R \in \mathbb{R}^{n\times m}$ such that for an arbitrary consistent matrix norm $\|\cdot\|$ the inequality*

$$\|I - R\boldsymbol{A}\| < 1 \tag{7.5}$$

holds, then $\boldsymbol{A}$ has full column rank.

The remaining task is to find the matrix R and to compute the norm of an interval matrix. Inspired by the real case, R can be set as an approximate pseudoinverse of the midpoint matrix of $\boldsymbol{A}$. Regarding the computation of matrix norms, there are easily computable consistent matrix norms $\|\boldsymbol{A}\|_1$, $\|\boldsymbol{A}\|_\infty$ (defined in Section 3.7). We do not use the norm $\|\cdot\|_2$ here since checking whether $\|\boldsymbol{A}\|_2 < 1$ for an interval matrix $\boldsymbol{A}$ is coNP-hard even for a very specialized case [136]. However, it can happen that $\|\boldsymbol{A}\|_1 \geq 1$, $\|\boldsymbol{A}\|_\infty \geq 1$, even though the spectral radius $\varrho(\boldsymbol{A}) < 1$ (for th definition see Section 11.9). For this sake we can still use the scaled maximum norm $\|\boldsymbol{A}\|_u$ for some $u > 0$ (defined also in Section 3.7). Let us demonstrate it for a real matrix.

Example 7.9. Let us have the matrix

$$A = \begin{pmatrix} 0.5 & 0.2 & 0.3 \\ 0.2 & 0.4 & 0.2 \\ 0.3 & 0.2 & 0.5 \end{pmatrix},$$

then $\varrho(A) \approx 0.94641$, $\|A\|_1 = 1$, $\|A\|_\infty = 1$. However, for $u = (0.62, 0.45, 0.62)^T$, $\|A\|_u = 0.95111 < 1$.

The previous example showed that the scaled maximum norm can help. The question is how to choose a proper vector u. We know that for each $u > 0$ the spectral radius $\varrho(\boldsymbol{A}) \le \|\boldsymbol{A}\|_u$. According to (3.6) and (3.7) we have

$$\|\boldsymbol{A}\|_u \le \alpha < 1 \iff \mathrm{mag}(\boldsymbol{A}) \cdot u \le \alpha \cdot u < u. \tag{7.6}$$

The matrix $C = \mathrm{mag}(\boldsymbol{A})$ is a nonnegative matrix and hence for a certain α and u we get

$$\alpha = \varrho(C) = \inf_{u>0} \|C\|_u$$

[139]. Hence, to compute such a vector u we can run a few steps of the well-known power method (see, e.g., [126]). It may converge to the eigenvector corresponding to the largest eigenvalue of C. When $\varrho(C) < 1$, the approximate eigenvector might be a suitable candidate for u satisfying (7.6).

Algorithm 7.10 (Computing u)**.** Input is an interval matrix $\boldsymbol{A}$. Output is a vector $u > 0$ satisfying $\|\boldsymbol{A}\|_u < 1$ when found.

1. Start with some $u_0 > 0$ (possibly $u_0 = (1, \dots, 1)^T$).
2. Compute $u_{k+1} = \mathrm{mag}(\boldsymbol{A}) u_k$ until $|u_{k+1} - u_k| < \epsilon$.
3. Set $u = u_{k+1}$ and check the property (7.6).
4. Return u if satisfied, otherwise return a message stating that such a u was not found.

Note that unlike the power method, it is not necessary to normalize the vectors u_k, since the algorithm might run only for a few steps.

Example 7.11. For the matrix from Example 7.9

$$A = \begin{pmatrix} 0.5 & 0.2 & 0.3 \\ 0.2 & 0.4 & 0.2 \\ 0.3 & 0.2 & 0.5 \end{pmatrix},$$

with $\varrho(A) = \varrho(|A|) \approx 0.94641$, let us take $u_0 = (1, 1, 1)^T$. Then

$$u_0 = (1, 1, 1)^T, \quad \|A\|_{u_0} = 1,$$
$$u_1 = (1, 0.8, 1)^T, \quad \|A\|_{u_1} = 0.96 < 1.$$

Example 7.12. The algorithm can also work for nonsymmetric matrices with varying signs of coefficients. Let

$$A = \begin{pmatrix} 0.40 & -0.27 & 0.27 & 0.20 \\ 0.27 & 0.19 & 0.31 & -0.18 \\ 0.27 & -0.31 & 0.06 & 0.13 \\ 0.20 & 0.18 & 0.13 & -0.22 \end{pmatrix},$$

$\varrho(A) \approx 0.306691$, $\varrho(|A|) \approx 0.927584$, let us take $u_0 = (1,1,1,1)^T$. Then

$$\begin{aligned} u_0 &= (1,1,1,1)^T, \quad \|A\|_{u_0} = 1.14, \\ u_1 &= (1.14, 0.95, 0.77, 0.73)^T, \quad \|A\|_{u_1} = 0.96545 < 1. \end{aligned}$$

Example 7.13. However, the algorithm will not always help, let us have

$$A = \frac{2}{5}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & 1 \\ 1 & -1 & -1 & 1 \end{pmatrix},$$

then $\varrho(A) = 0.8 < \varrho(|A|) = 1.6$, let us take $u_0 = (1,1,1,1)^T$. Then

$$\begin{aligned} u_0 &= (1,1,1,1)^T, \quad \|A\|_{u_0} = 1.6, \\ u_1 &= (1.6, 1.6, 1.6, 1.6)^T, \quad \|A\|_{u_1} = 1.6 > 1. \end{aligned}$$

7.3.1 Relationship between the two sufficient conditions

In the previous subsection the two sufficient conditions for a matrix having full column rank were introduced – (7.2) and (7.5). The question is what is the relation between these two conditions?

When $\boldsymbol{A}$ is a square interval matrix, both conditions are of the same strength.

Theorem 7.14. *When* $\boldsymbol{A}$ *is a square interval matrix, then*

$$(7.2) \iff (7.5).$$

Proof. ($\Leftarrow$) When $\boldsymbol{A}$ is a square interval matrix, then for every $(I - RA) \in (I - R\boldsymbol{A})$

$$\varrho(I - RA) \le \varrho(\mathrm{mag}(I - R\boldsymbol{A})) \le \|I - R\boldsymbol{A}\| < 1.$$

Using the properties (3.1)–(3.5) we get

$$\varrho(\mathrm{mag}(I - R\boldsymbol{A})) = \varrho(|I - RA_c| + |R|A_\Delta) < 1,$$

which is actually the sufficient condition for regularity from Theorem 4.2. Hence in the square case (7.5) implies Theorem 4.2 and also (7.2).

($\Rightarrow$) If A_c has full column rank then $A_c^+ = A_c^{-1}$ in the square case and the condition (7.2) means $\boldsymbol{A}$ is strongly regular (Theorem 4.33 statement *3.*). When setting $R = A_c^{-1}$ we get the statement *4.* of Theorem 4.33 which is equivalent to (7.5). □

What is the relation in the rectangular case? In the following theorem we claim that the second condition is stronger.

Theorem 7.15 (Horáček et al. [87]). *For a general matrix $\boldsymbol{A} \in \mathbb{IR}^{m\times n}$ the implication a) $\Rightarrow$ b) holds, where*

a) A_c has full column rank and $\varrho(|A_c^+|A_\Delta) < 1$,

b) $\exists u \in \mathbb{R}^n$, $u > 0$ and $\exists R \in \mathbb{R}^{n\times m}$ such that $\|I - R\boldsymbol{A}\|_u < 1$.

Proof. When A_c has full column rank then $A_c^+ = (A_c^T A_c)^{-1} A_c^T$, which causes the matrix $A_c^+\boldsymbol{A}$ to have the midpoint matrix equal to I. Hence $I - A_c^+\boldsymbol{A}$ is the matrix with the midpoint matrix 0. According to the property (3.4) it has the radius matrix equal to $|A_c^+|A_\Delta$. Therefore, for all $C \in I - A_c^+\boldsymbol{A}$ it holds that $|C| \leq |A_c^+|A_\Delta$. Hence, together with *a)*, it gives

$$\varrho(C) \leq \varrho\left(|A_c^+|A_\Delta\right) < 1.$$

By [139] (Lemma 3.2.1), there must exist some $u > 0$ such that $\|C\|_u < 1$ for each $C \in (I - A_c^+\boldsymbol{A})$. According to the definition of the scaled maximum norm there must exist $u > 0$ such that $\|I - A_c^+\boldsymbol{A}\|_u < 1$. Finally, to make *b)* hold, set $R = (A_c^T A_c)^{-1} A_c^T$. □

Using Theorem 7.5 we can formulate the other implication. However, we need to modify the second condition a little.

Theorem 7.16 (Horáček et al. [87]). *For a general matrix $\boldsymbol{A} \in \mathbb{IR}^{m\times n}$ the implication a) $\Leftarrow$ b*) holds, where*

a) A_c has full column rank and $\varrho(|(A^c)^+|A^\Delta) < 1$,

b) $\exists u \in \mathbb{R}^n, u > 0$, $\exists R = (CA_c^+) \in \mathbb{R}^{n\times m}$, for some nonsingular $C \in \mathbb{R}^{n\times n}$ such that $\|I - R\boldsymbol{A}\|_u < 1$.*

Proof. The statement *b*)* is equivalent to $\mathrm{mag}(I - R\boldsymbol{A})u < u$ for a suitable R. Using the properties (3.1)–(3.5) we get

$$\mathrm{mag}(I - R\boldsymbol{A}) = |I - RA_c| + |R|A_\Delta$$

and

$$(|I - RA_c| + |R|A_\Delta)u < u.$$

By [139] (Corollary 3.2.3), because the whole matrix on the left side is nonnegative, the formula is equivalent to $\varrho(|I - RA_c| + |R|A_\Delta) < 1$ and according to Theorem 7.5 the claim *a)* holds. □

7.4 Solvability

In the final comparison of the mentioned methods for detecting unsolvability a method for detecting the opposite – solvability of a system – might bring a new information to understanding the bigger picture. Hence, this small section is devoted to this topic. As was mentioned earlier, here, we are going to deal only with weak solvability concept. Unfortunately, generally, checking weak solvability is an NP-complete problem [178]. That is why we focus only on sufficient conditions here.

First option is to consider the midpoint system $A_c x = b_c$. This system is possibly unsolvable, that is why we set

$$x \approx A_c^+ b_c.$$

The vector x may not be a solution of the midpoint system, however, we assume that x is a solution of a system that is close enough to the midpoint system, and hence still contained in the original interval system. We can check this by applying the Oettli–Prager theorem to x. The checking must be done in a verified way using interval arithmetics. We refer to this procedure as *midpoint check*.

Secondly, the vector $\text{sign}(x)$ gives us a hint in which orthant the solution can be found. With such knowledge we can rewrite the Oettli–Prager formula for the given orthant and apply verified linear programming. We refer to this procedures as *orthant check*.

7.5 Comparison of methods

In this section we compare the previously discussed methods for detecting unsolvability; namely:

- `ge` – Gaussian elimination approach described in Section 7.2.2,
- `subsq` – the subsquares approach described in Section 7.2.3, with 5 random square subsystem selections,
- `lsq` – the least squares approach discussed in Section 7.2.4,
- `fcr` – the approach using the full column rank sufficient condition (7.5) with $\|\cdot\|_\infty$ norm described in Section 7.3,

- `fcrit` – the approach using the condition (7.5), with scaled maximum norm and iterative search for a vector u described in Section 7.3, maximum number of iterations is set to 5,
- `eig` – the approach using condition (7.2) with nonverified computation of spectral radius described in Section 7.3.

The method `eig` is shown for comparison purpose only, it is not a verified method since the spectral radius in the formula is not computed in a verified way.

The methods are tested on random systems with intervals having fixed radii. The radius range is selected to suit a particular group of methods – to catch the region of its applicability. The methods were applied to systems with various number of variables ($n = 5, 10, 15, \ldots, 100$), the number of equations m was always selected according to n as $m = \frac{3}{2}n$ to form a rectangular system. Random systems were generated as described in Example 7.2. For each combination of a radius and a system size, 100 random systems were generated and tested for unsolvability by various methods.

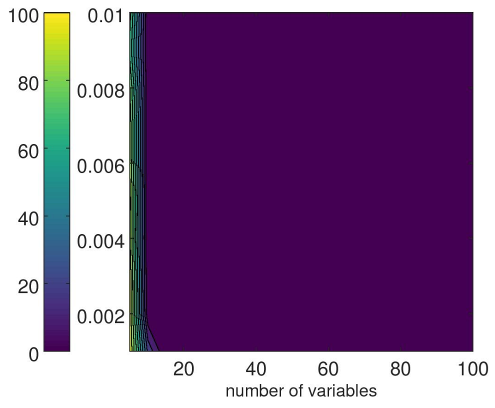
The results of testing are displayed as heat maps in Figure 7.1 and 7.2. A point on a heat map shows the percentage of systems that were detected to be unsolvable by a given method, a given number of system variables (x-axis) and a given radius (y-axis). Note that, even though, the sizes (x-axes) remain the same, the interval radii range (y-axes) might change from method to method. There are basically two types of methods. The first group works only for "smaller" radii relative to the coefficients of A_c, b_c ($r < 0.01$) and "smaller" system sizes ($n < 40$) – `ge`, `lsq`, `subsq` (Figure 7.1). The methods in the second group work even for "larger" radii ($r < 1$) – `fcr`, `fcrit`, `eig` (Figure 7.2).

The method `ge` works only for very small systems. Since for detection of unsolvability it must be used without preconditioning, the interval operations cause large overestimation that will occur for larger systems ($n > 10$) and Gaussian elimination will find a solution or it will not be able to proceed because all pivot intervals contain 0 at some step.

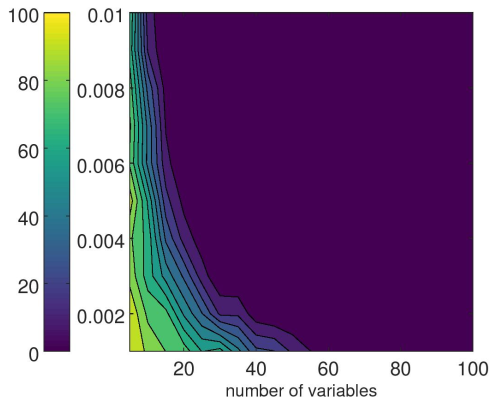
The methods `lsq` and `subsq` detect unsolvability with a similar success rate. The efficiency and the computation time of `subsq` depend on the number of random square subsystems inspected. Both methods depend on the efficiency of a method used for computing enclosures of square interval systems.

The best methods are `fcr` and `fcrit`. The `frc` is the fastest method (the largest average computation time that occurred during testing using the DESKTOP setting was 0.2415 seconds). In the tested cases, the iterative search for scaled maximum norm seemed to help. It adds only some minor computational time, the longest average computation took about 0.2426 seconds.

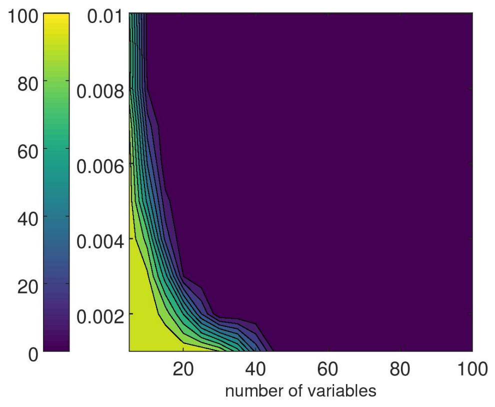
The method `eig` returned great results too, however because of nonverified computation it did not return verified results and therefore it was excluded from the competition. Nevertheless, the heat maps of `eig` and `fcrit` look very similar. The strength of `fcrit` stands or falls on finding a proper vector u. In this case, the heat maps show, that our heuristic iterative search for u does the job very well.



(a) `ge`

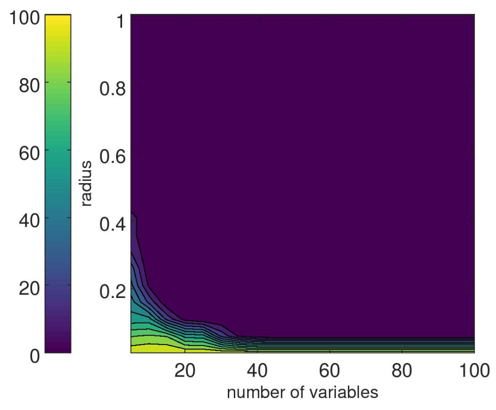


(b) `subsq`

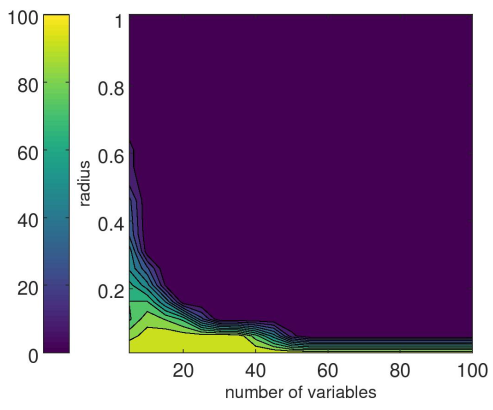


(c) `beeck`

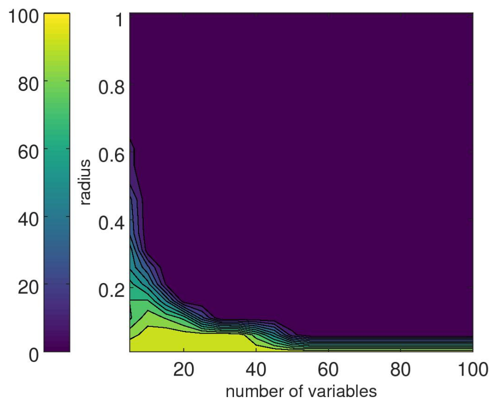
Figure 7.1: Strength of unsolvability tests `ge`, `subsq` and `beeck`. Color corresponds to percentage of unsolvable systems discovered (the lighter the area the higher the percentage).



(a) `fcr`

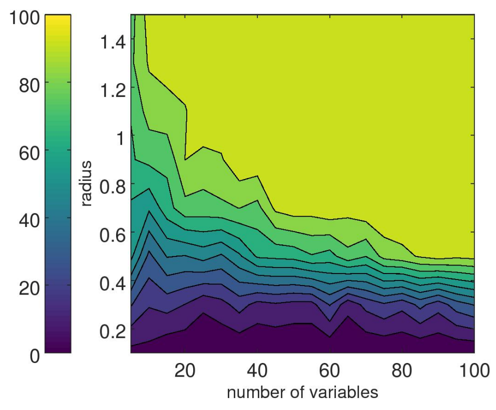


(b) `fcrit`

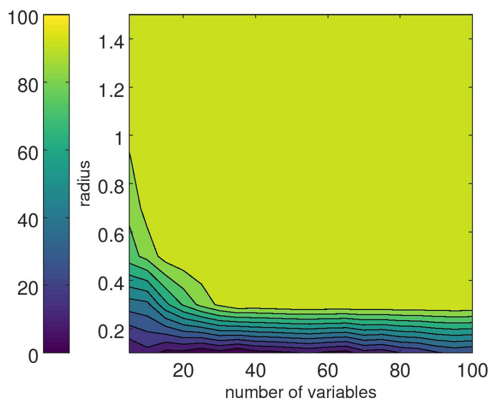


(c) `eig`

Figure 7.2: Strength of unsolvability tests `fcr`, `fcrit` and `eig`. Color corresponds to percentage of unsolvable systems discovered (the lighter the area the higher the percentage). Notice the different scale on y-axis in contrast to Figure 7.1.



(a) midpoint check



(b) orthant check

Figure 7.3: Strength of two solvability tests from Section 7.4. Color corresponds to percentage of solvable systems discovered (the lighter the area the higher percentage).

With growth of interval widths, generated systems become solvable. To check this we applied a similar test for the two solvability conditions. The results are depicted in 7.3. The orthant check is clearly better. The heat map (b) in Figure 7.2 and the heat map (b) in Figure 7.3 form a gap between the NP-complete and coNP-complete problem (unsolvability and solvability). For the tested systems the gap seems to be narrow.

8 Determinant of an interval matrix

- Known results
- Complexity of approximations
- Methods for computing determinant enclosures
- Determinant of symmetric matrices
- Classes of matrices with polynomially computable determinant bounds
- Comparison of methods

Applications of interval determinants were discussed in [208] for testing for Chebyshev systems or in [158] for computer graphics applications. Nevertheless, the area of interval determinants has not been much explored yet. In this chapter we address computational properties of determinants of general interval matrices. Next, we mention a known tool for computing interval determinants – interval Gaussian elimination. We then show how to modify existing tools from the classical linear algebra – Hadamard's inequality and the Gerschgorin circle theorem. After that, we design our new method based on Cramer's rule and solving interval linear systems. Regarding symmetric matrices, there are results about enclosing their eigenvalues that can also be used for computing interval determinants. All the methods work much better when combined with some kind of preconditioning. We briefly address this topic. Since computing a general interval determinant is intractable we point out classes of matrices with polynomially computable tasks connected to determinants. At the end we provide thorough testing of the mentioned methods on random general and symmetric interval matrices and discuss the use of these methods. The chapter is based on our work [86].

8.1 Definition

Definition 8.1 (Interval determinant 1)**.** Let $\boldsymbol{A}$ be a square interval matrix, then its determinant is defined as

$$\det(\boldsymbol{A}) = \{\det(A) \mid A \in \boldsymbol{A}\}.$$

Since the determinant of a real matrix is actually a polynomial, it is continuous. A closed interval is a compact set, so is the Cartesian product of them. Hence an interval matrix is a compact set. The image of the compact set under continuous mapping is again a compact set. That is why we can define the interval determinant in a more pleasant but equivalent way.

Definition 8.2 (Interval determinant 2)**.** Let $\boldsymbol{A}$ be a square interval matrix, then its determinant can be defined as the interval

$$\det(\boldsymbol{A}) = \Big[\min\{\det(A) \mid A \in \boldsymbol{A}\},\ \max\{\det(A) \mid A \in \boldsymbol{A}\}\Big].$$

Sometimes we refer to the exact determinant as the *hull.* In the following section we will state that computing the exact bounds on an interval determinant is an intractable problem. That is why, we are usually satisfied with an enclosure of the interval determinant. Of course, the tighter is the enclosure the better.

Definition 8.3 (Enclosure of interval determinant)**.** Let $\boldsymbol{A}$ be a square interval matrix, then an interval enclosure of its determinant is defined to be any $\boldsymbol{d} \in \mathbb{IR}$ such that

$$\det(\boldsymbol{A}) \subseteq \boldsymbol{d}.$$

Therefore, through this chapter we deal with the following problem:

Problem: Compute a tight enclosure of the determinant of $\boldsymbol{A}$.

8.2 Known results

To the best knowledge of ours, there are only a few theoretical results regarding interval determinants. Some of the results can be found in [112, 173]. In [173] we find a theorem stating that for an arbitrary matrix $A \in \boldsymbol{A}$ a matrix $A' \in \boldsymbol{A}$ can be found such that both A and A' have equal determinants and all coefficients of A', except one, come from some *edge matrix* of $\boldsymbol{A}$. (i.e., a real matrix where each coefficient A_{ij} is equal to the lower or upper bound of $\boldsymbol{A}_{ij}$).

Theorem 8.4 (Edge theorem)**.** *Let $\boldsymbol{A}$ be an interval matrix, then for each $A \in \boldsymbol{A}$, there exists a pair of indices (k, l) and $A' \in \boldsymbol{A}$ in the following form*

$$A'_{ij} \in \begin{cases} \{\underline{A}_{ij}, \overline{A}_{ij}\}, & (i,j) \neq (k,l), \\ \left[\underline{A}_{ij}, \overline{A}_{ij}\right], & (i,j) = (k,l), \end{cases}$$

such that $\det(A) = \det(A')$.

We prove the theorem with a more detailed demonstration than the one showed in [173].

Proof. Let $A \in \boldsymbol{A}$ be given. For a matrix $A' \in \boldsymbol{A}$ such that $\det(A) = \det(A')$, we remember the number of coefficients of A' such that $A'_{ij} \notin \{\underline{A}_{ij}, \overline{A}_{ij}\}$ (i.e., they do not lie on the edge of interval matrix). We wish to find A' that minimizes this number. We show that there exists $A' \in \boldsymbol{A}$ such that $\det(A') = \det(A)$ and this number is at most 1.

For the sake of contradiction let us assume that this number is 2 or greater. Thus there exist two pairs of indices $(p, q), (r, s)$ such that $A'_{pq} \in (\underline{A}_{pq}, \overline{A}_{pq})$ and $A'_{rs} \in (\underline{A}_{rs}, \overline{A}_{rs})$. Notice that here open intervals are used. The determinant of A' can be expressed as a function of these coefficients.

$$\det(A) = \det(A') = a \cdot A'_{pq} + b \cdot A'_{rs} + c \cdot A'_{pq}A'_{rs} + d, \tag{8.1}$$

for some $a, b, c, d \in \mathbb{R}$. When we fix the value of the determinant, we can express a variable (without loss of generality A'_{pq}) as

$$A'_{pq} = -\frac{b \cdot A'_{rs} + (d - \det(A))}{c \cdot A'_{rs} + a}, \tag{8.2}$$

which is a linear fractional function. Note that the denominator cannot be zero, otherwise it forces the function (8.1) to have only one variable which is a contradiction to our assumption that the number of variables is greater than or equal to 2.

The two cases, which are depicted in Figure 8.1, can occur. The dark box represents the Cartesian product of intervals $\boldsymbol{A}_{rs} \times \boldsymbol{A}_{pq}$. The first case represents a linear fractional function. In the second case the function degenerates to just a line. According to the definition of A' and (8.1) the point (A'_{pq}, A'_{rs}) lies in the interior of the box. Hence the function (8.2) intersects the box. We then move the point (A'_{pq}, A'_{rs}) along the graph of the function (8.2) to reach a new point (A''_{rs}, A''_{pq}) that lies on the border of the box. This way we actually obtained a new matrix A'' from A' that decreases the number of coefficients that do not belong to $\{\underline{A}_{ij}, \overline{A}_{ij}\}$ by one. If necessary, we can repeat the process and reduce the number of such coefficients to one.

□

The following claim is an immediate consequence and is also mentioned without an explicit proof in [173]. It claims that the exact bounds of the interval determinant can be computed as minimum and maximum determinant of all 2^{n^2} possible edge matrices of $\boldsymbol{A}$. Another reasoning for the corollary, not using the Edge theorem, is simply based just on linearity of determinant of a real matrix with respect to each coefficient.

Corollary 8.5. *For a given square interval matrix $\boldsymbol{A}$ the interval determinant can be obtained as*

$$\det(\boldsymbol{A}) = [\min(S), \max(S)], \ \textit{where } S = \{\det(A) \mid \forall i, j \ A_{ij} = \underline{A}_{ij} \textit{ or } A_{ij} = \overline{A}_{ij}\}.$$

Proof. For each $A \in \boldsymbol{A}$ a matrix A' can be constructed. This matrix has at most one coefficient $A'_{ij} \in (\underline{a}_{ij}, \overline{a}_{ij})$. A determinant of A' expressed in this coefficient is a linear

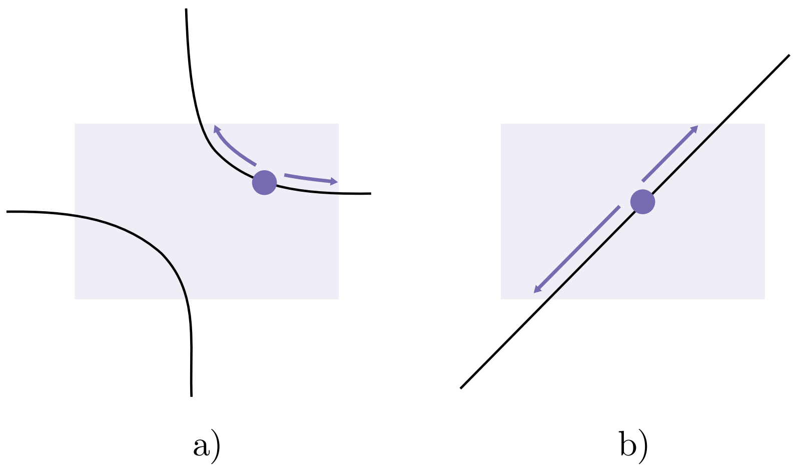


Figure 8.1: The two possible cases from the proof of Theorem 8.4. The dark box represents the Cartesian product of intervals $\boldsymbol{A}_{rs} \times \boldsymbol{A}_{pq}$. The curve represents the function (8.2).

function. Clearly the function value can be increased or decreased by setting

$$A'_{ij} = \underline{A}_{ij} \quad \text{or} \quad A_{ij} = \overline{A}'_{ij}.$$

That is why the matrix having minimum (or maximum) determinant must be some edge matrix of $\boldsymbol{A}$. □

A known result coming also from [173] is the following.

Theorem 8.6. *Let A_c be a rational nonnegative symmetric positive definite matrix. Then checking whether the interval matrix*

$$\boldsymbol{A} = [A_c - E, A_c + E]$$

is regular is a coNP*-complete problem.*

Proof. For a proof see, e.g., [173]. □

As a consequence of this theorem we can obtain the following important theorem [112, 173].

Theorem 8.7. *Let A_c be a rational nonnegative matrix. Computing the either of the exact bounds $\underline{\det(\boldsymbol{A})}$ or $\overline{\det(\boldsymbol{A})}$ of the matrix*

$$\boldsymbol{A} = [A_c - E, A_c + E],$$

is NP*-hard.*

Proof. The proof of this theorem is also described in [112, 173]. □

8.3 Complexity of approximations

At the end of the previous section we stated that the problem of computing the exact bounds of the determinant of an interval matrix is generally an NP-hard problem. We could hope for having at least some approximation algorithms. Unfortunately, in this section we prove that this is not the case, neither for relative nor for absolute approximation.

Theorem 8.8 (Relative approximation, Horáček et al. [86])**.** *Let A_c be a rational nonnegative symmetric positive definite matrix. Let $\boldsymbol{A} = [A_c - E, A_c + E]$ and ε be arbitrary such that $0 < \varepsilon < 1$. If there exists a polynomial time algorithm returning $[\underline{a}, \overline{a}]$ such that*

$$\det(\boldsymbol{A}) \subseteq [\underline{a}, \overline{a}] \subseteq [1 - \varepsilon, 1 + \varepsilon] \cdot \det(\boldsymbol{A}),$$

then $\mathsf{P} = \mathsf{NP}$.

Proof. We use the fact from Theorem 8.6 that for a rational nonnegative symmetric positive definite matrix A_c, checking whether the interval matrix $\boldsymbol{A} = [A_c - E, A_c + E]$ is regular is a coNP-complete problem. We show that if such an ε-approximation algorithm existed, it would decide regularity from the above mentioned problem; which implies $\mathsf{P} = \mathsf{NP}$.

For a regular interval matrix we must have $\underline{\det(\boldsymbol{A})} > 0$ or $\overline{\det(\boldsymbol{A})} < 0$. If $\underline{\det(\boldsymbol{A})} > 0$ then, from the second inclusion $\underline{a} \geq (1 - \varepsilon) \cdot \underline{\det(\boldsymbol{A})} > 0$. On the other hand, if $\underline{a} > 0$ then from the first inclusion $\underline{\det(\boldsymbol{A})} \geq \underline{a} > 0$. Therefore, we have $\underline{\det(\boldsymbol{A})} > 0$ if and only if $\underline{a} > 0$. The corresponding equivalence for $\overline{\det(\boldsymbol{A})} < 0$ can be derived in a similar way. Therefore, if we had such an ε-approximation algorithm, from the sign of the returned determinant enclosure the regularity can be decided. □

Theorem 8.9 (Absolute approximation, Horáček et al. [86])**.** *Let A_c be a rational nonnegative symmetric positive definite $n \times n$ matrix. Let $\boldsymbol{A} = [A_c - E, A_c + E]$ and let ε be arbitrary such that $0 < \varepsilon$. If there exists a polynomial time algorithm returning $[\underline{a}, \overline{a}]$ such that*

$$\det(\boldsymbol{A}) \subseteq [\underline{a}, \overline{a}] \subseteq \det(\boldsymbol{A}) + [-\varepsilon, \varepsilon],$$

then $\mathsf{P} = \mathsf{NP}$.

Proof. We again use the fact from Theorem 8.6 and show that if such an ε-approximation algorithm existed, then we can decide the coNP-complete problem. Which would imply $\mathsf{P} = \mathsf{NP}$.

Let the matrix A_c consist of rational numbers with nominator and denominator representable by k bits (we can take k as the maximum number of bits needed for any nominator or denominator). Then nominators and denominators of coefficients in $A_c - E$ and $A_c + E$ are also representable using $O(k)$ bits. Each row of the matrices is now multiplied with a product of all denominators from the corresponding row of both $A_c - E, A_c + E$. Each denominator still uses k bits and each nominator uses $O(nk)$ bits. We obtained a new matrix $\boldsymbol{A}'$. The whole matrix now uses $O(n^3k)$ bits which is polynomial in n and k.

We only multiplied by nonzero constants therefore the following property holds

$$0 \notin \det(\boldsymbol{A}) \Longleftrightarrow 0 \notin \det(\boldsymbol{A}').$$

After canceling fractions, the matrix $\boldsymbol{A}'$ has integer bounds. Its determinant must also have integer bounds. Therefore deciding whether $\boldsymbol{A}'$ is regular means deciding whether $|\det(\boldsymbol{A}')| \geq 1$. We can multiply one arbitrary row of $\boldsymbol{A}'$ by 2ε and get a new matrix $\boldsymbol{A}''$ having $\det(\boldsymbol{A}'') = 2\varepsilon \det(\boldsymbol{A}')$. Now, we can apply the approximation algorithm and compute an absolute approximation $[\underline{a}'', \overline{a}'']$ of the determinant of $\boldsymbol{A}''$. Let $\det(\boldsymbol{A}') \geq 1$. Then $\det(\boldsymbol{A}'') \geq 2\varepsilon$ and the lower bound of the absolute approximation is

$$\underline{a}'' \geq \underline{\det(\boldsymbol{A}'')} - \varepsilon \geq \varepsilon > 0,$$

On the other hand, if $\underline{a}'' > 0$ then

$$2\varepsilon\, \underline{\det(\boldsymbol{A}')} = \underline{\det(\boldsymbol{A}'')} \geq \underline{a}'' > 0.$$

Hence, even $\underline{\det(\boldsymbol{A}')} > 0$ and since it is an integer it must be greater or equal to 1. The case of $\overline{\det(\boldsymbol{A}')} \leq -1$ is handled similarly. Therefore, we proved

$$0 \notin \det(\boldsymbol{A}) \Longleftrightarrow 0 \notin \det(\boldsymbol{A}') \Longleftrightarrow 0 \notin [\underline{a}'', \overline{a}''].$$

That means we can decide regularity with our ε-approximation algorithm. $\square$

8.4 Enclosure of a determinant: general case

8.4.1 Gaussian elimination

To compute an enclosure of the determinant of an interval matrix Gaussian elimination introduced in Chapter 5 can be used – after transforming a matrix into row echelon form an enclosure of the determinant is computed as the product of the intervals on the main diagonal. We remind that, as in the real case, swapping of two rows changes the sign of the resulting enclosure.

It is usually favorable to use Gaussian elimination together with a preconditioning (more details will be explained in Subsection 8.4.6). We would recommend the midpoint inverse preconditioning as was done in [208].

Example 8.10. Because of properties of interval arithmetic (subdistributivity) interval Gaussian elimination leads to a certain overestimation. Let us have a matrix

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{a}_{11} & \boldsymbol{a}_{12} \\ \boldsymbol{a}_{21} & \boldsymbol{a}_{22} \end{pmatrix}.$$

In Section 4.2 we computed the hull of the determinant of such a matrix as $\det(\boldsymbol{A}) = \boldsymbol{a}_{11} \cdot \boldsymbol{a}_{22} - \boldsymbol{a}_{12} \cdot \boldsymbol{a}_{21}$ (the determinant of a 2×2 matrix is a formula with single occurrence of each matrix coefficient and we can apply Theorem 3.13).

After one elimination step we get the matrix

$$\begin{pmatrix} \boldsymbol{a}_{11} & \boldsymbol{a}_{12} \\ 0 & \boldsymbol{a}_{22} - \frac{\boldsymbol{a}_{21}}{\boldsymbol{a}_{11}} \cdot \boldsymbol{a}_{12} \end{pmatrix}.$$

The following holds according to subdistibutivity and nonexistence of inverse element in the interval arithmetics.

$$\boldsymbol{a}_{11} \cdot \left(\boldsymbol{a}_{22} - \frac{\boldsymbol{a}_{21}}{\boldsymbol{a}_{11}} \cdot \boldsymbol{a}_{12} \right) \supseteq \boldsymbol{a}_{11} \cdot \boldsymbol{a}_{22} - \boldsymbol{a}_{12} \cdot \boldsymbol{a}_{21} = \det(\boldsymbol{A}).$$

8.4.2 Gerschgorin discs

It is a well-known result that the determinant of a real matrix is a product of its eigenvalues. That is why an enclosure of an interval determinant can be computed as a product of enclosures of interval matrix eigenvalues, e.g., [69, 78, 108, 124]. The Gerschgorin circle theorem can be used as well.

This classical result claims that for a real square matrix A each its eigenvalue lies inside at least one *Gerschgorin disc* in complex plane with centers A_{ii} and radius $\sum_{j \neq i} |A_{ij}|$. When $\boldsymbol{A}$ is an interval matrix, to each real matrix $A \in \boldsymbol{A}$ there corresponds a set of Gerschgorin discs. Increasing or decreasing the coefficients of A within $\boldsymbol{A}$ shifts or scales the discs. However, all discs corresponding to ith diagonal element of $\boldsymbol{A}$ in all situations are contained inside a disc with the center $\mathrm{mid}(\boldsymbol{A}_{ii})$ and the radius $\mathrm{rad}(\boldsymbol{A}_{ii}) + \sum_{j \neq i} \mathrm{mag}(\boldsymbol{A}_{ij})$ as depicted in Figure 8.2. We can call such a disc an *interval Gerschgorin disc.*

As in the case of the real Gerschgorin discs, it is also well known that in the union of k intersecting discs there somewhere lie k eigenvalues. By intersecting discs we mean that their projection on the horizontal axis is a continuous line. That might complicate the situation a bit. When k interval Gerschgorin discs intersect each $A \in \boldsymbol{A}$ specifies a distribution of k eigenvalues in the bunch of the k interval discs.

That is why we can deal with each bunch of intersecting discs separately. We compute the verified interval enclosing all products of k eigenvalues regardless of their position inside this bunch. The computation of the verified enclosures will depend on the number of discs in the bunch (odd/even) and on whether the bunch contains the point 0. In Figures 8.3 and 8.4 all the possible cases and resulting verified enclosures are depicted. The resulting determinant will be a product of intervals corresponding to all bunches of intersecting discs.

The formulas for enclosures of a bunch of discs are based on the following simple fact depicted in Figure 8.5: an eigenvalue lying inside an intersection of two discs can be real or complex $(c + bi)$. In the second case the conjugate complex number $c - bi$ is also an eigenvalue. Their product is $b^2 + c^2$, which can be enclosed from above by a^2, where a is defined in Figure 8.5. The whole reasoning is based on Pythagorean theorem and geometric properties of hypotenuse.

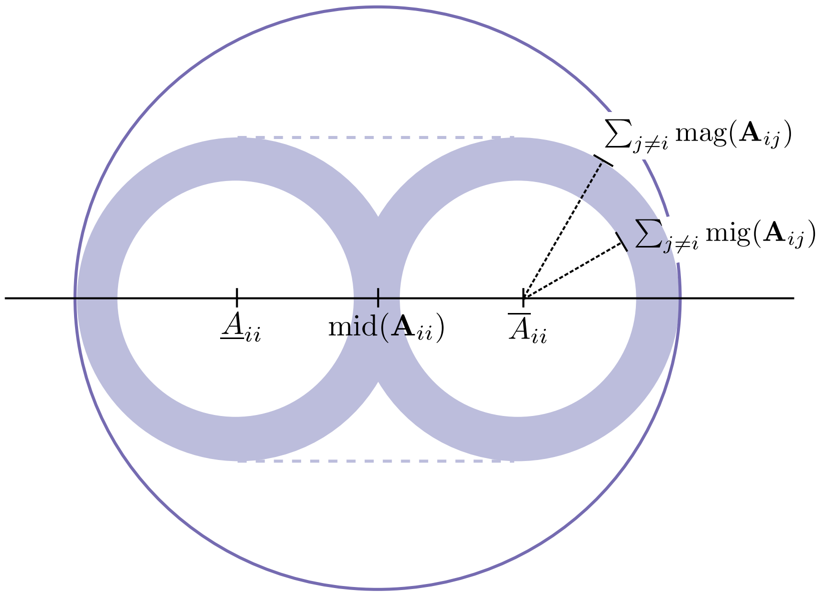

$\sum_{j\neq i} \text{mag}(\mathbf{A}_{ij})$
$\sum_{j\neq i} \text{mig}(\mathbf{A}_{ij})$
$\underline{A}_{ii}$
$\text{mid}(\mathbf{A}_{ii})$
$\overline{A}_{ii}$

Figure 8.2: One interval Gerschgorin disc (the large circle). The grey area mirrors the scaling and shifting of a real Gerschgorin disc when shifting coefficients of A within intervals of $\boldsymbol{A}$

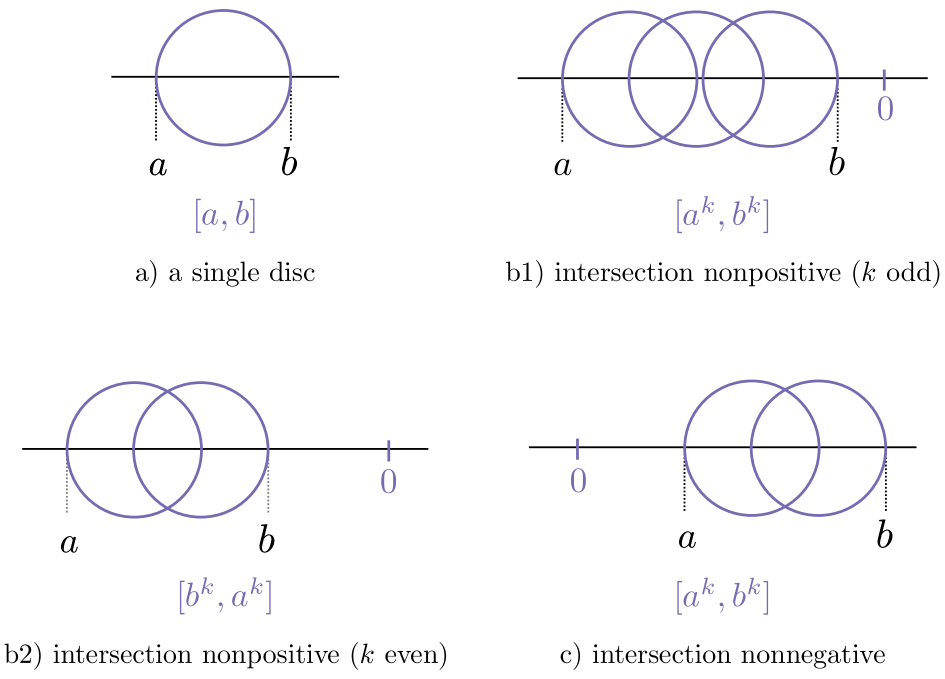

a
b
$[a, b]$
a) a single disc
a
b
0
$[a^k, b^k]$
b1) intersection nonpositive (k odd)
a
b
0
$[b^k, a^k]$
b2) intersection nonpositive (k even)
0
a
b
$[a^k, b^k]$
c) intersection nonnegative

Figure 8.3: Verified enclosures of any product of real eigenvalues inside a bunch of intersecting interval Gerschgorin discs not containing 0.

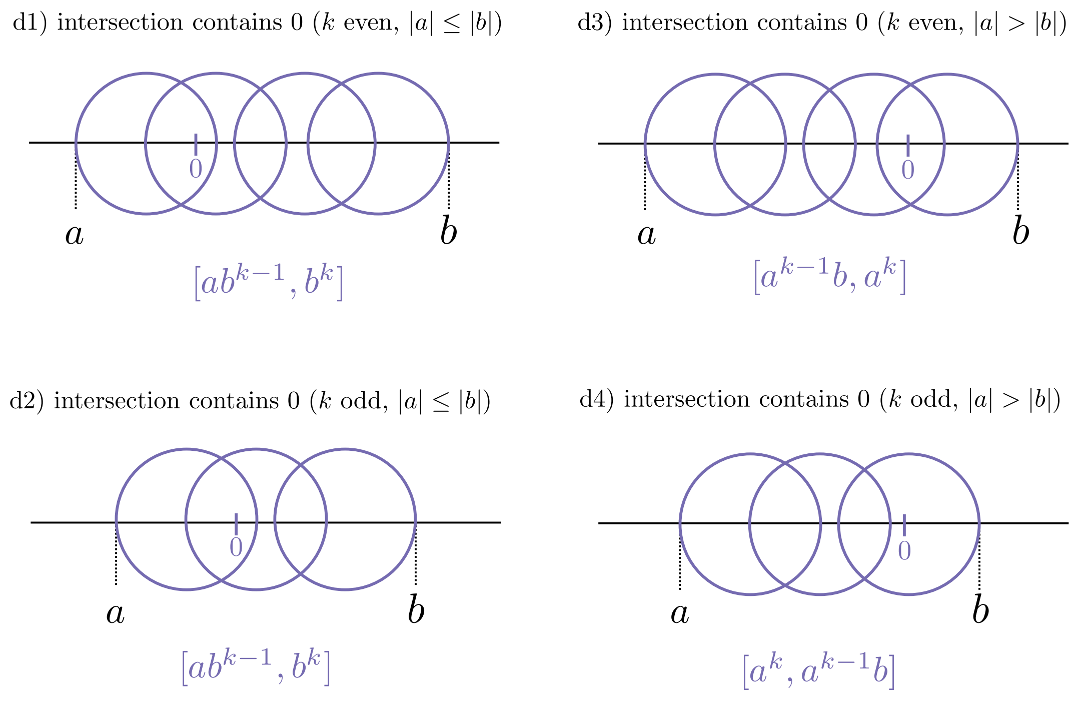

d1) intersection contains 0 (k even, $|a| \leq |b|$)
0
a
b
$[ab^{k-1}, b^k]$
d3) intersection contains 0 (k even, $|a| > |b|$)
0
a
b
$[a^{k-1}b, a^k]$
d2) intersection contains 0 (k odd, $|a| \leq |b|$)
0
a
b
$[ab^{k-1}, b^k]$
d4) intersection contains 0 (k odd, $|a| > |b|$)
0
a
b
$[a^k, a^{k-1}b]$

Figure 8.4: Verified enclosures of any product of real eigenvalues inside a bunch of intersecting interval Gerschgorin discs containing 0.

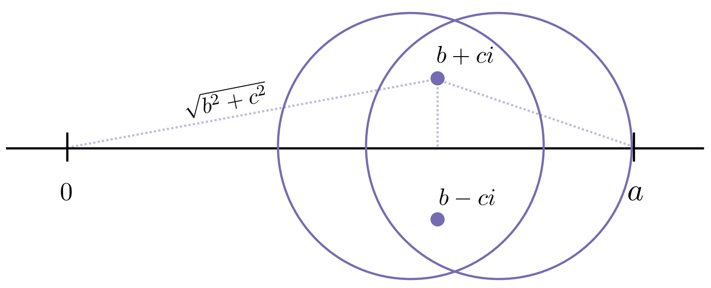


Figure 8.5: Enclosing a product of two complex eigenvalues.

The generalized interval Gerschgorin discs approach may produce large overestimation. However, it might be useful in case of tight intervals or a matrix close to a diagonal one.

8.4.3 Hadamard's inequality

A simple but rather crude enclosure of interval determinant can be obtained by the well known Hadamard's inequality. For an $n \times n$ real matrix A we have

$$|\det(A)| \leq \prod_{i=1}^{n} \|A_{*i}\|_2 = \prod_{i=1}^{n} \left(\sum_{j=1}^{n} |A_{ji}|^2 \right)^{\frac{1}{2}},$$

where $\|A_{*i}\|_2$ is the Euclidean norm of the ith column of A. This inequality is simply transferable to the interval case. Since the inequality holds for every $A \in \boldsymbol{A}$ we have

$$\det(\boldsymbol{A}) \subseteq [-d, +d]\,, \text{ where } d = \prod_{i=1}^{n} \left(\sum_{j=1}^{n} \operatorname{mag}(\boldsymbol{A}_{ji})^2 \right)^{\frac{1}{2}}.$$

Since $\det(A) = \det(A^T)$, the same formula can be computed also for rows instead of columns and intersection of the two determinant enclosures can be taken. It is a fast and simple method. A drawback is that the obtained enclosure is often wide. A second problem is that it is impossible to detect the sign of the determinant.

8.4.4 Cramer's rule

In this section we introduce our method that is based on Cramer's rule [86]. In Chapter 5 we introduced various methods for computing an enclosure of the solution set of a square interval linear system and we can again make use of them. According to Cramer's rule for a real system of equations $Ax = b$ we get

$$x_1 = \frac{\det(A_{1 \leftarrow b})}{\det(A)},$$

where x_1 is the first coefficient of the solution vector x and $A_{1\leftarrow b}$ is the matrix that emerges when we substitute the first column of A with b. We can rewrite the equation as

$$\det(A) = \frac{\det(A_{1\leftarrow b})}{x_1}.$$

Let $b = e_1$ and let us assume that we know x_1 from solving a system $Ax = b$ then $\det(A_{1\leftarrow b})$ is equal to $\det(A_{2:n})$ which emerges by omitting the first row and column from A. Now, we have reduced our problem to computing determinant of a matrix of lower order and we can repeat the same procedure iteratively until the determinant is easily computable. Such a procedure will not pay off in the case of real matrices. However, it will help in the interval case. We actually get

$$\det(\boldsymbol{A}) \subseteq \det(\boldsymbol{A}_{2:n})/\boldsymbol{x}_1, \tag{8.3}$$

where $\boldsymbol{x}_1$ is the interval enclosure of the first coefficient of the solution of $\boldsymbol{A}x = e_1$, computed by some of the cited methods. Notice that we can use arbitrary e_i instead of e_1. The method works when all enclosures of $\boldsymbol{x}_1$ in the recursive calls (8.3) do not contain 0.

8.4.5 Monotonicity checking

According to [149], the partial derivatives of $\det(A)$ of a real nonsingular matrix $A \in \mathbb{R}^{n\times n}$ are

$$\frac{\partial \det(A)}{\partial A} = \det(A)A^{-T}.$$

Let $\boldsymbol{B}$ be an an interval enclosure for the set $\{A^{-T} \mid A \in \boldsymbol{A}\}$. Since $\boldsymbol{A}$ is regular, every $A \in \boldsymbol{A}$ has the same sign of determinant. Hence, e.g., $\det(A_c)\boldsymbol{B}_{ij}$ gives information about monotonicity of the determinant.

When long as 0 is not in the interior of $\boldsymbol{B}_{ij}$, then we can do the following reasoning: if $\det(A_c)\boldsymbol{B}_{ij}$ is a nonnegative interval, then $\det(A)$ is nondecreasing in A_{ij}, and hence its minimal value is attained at $A_{ij} = \underline{A}_{ij}$. Similarly for $\det(A_c)\boldsymbol{B}_{ij}$ nonpositive.

In this way, we split the problem of computing $\det(\boldsymbol{A})$ into two subproblems of computing the lower and upper bounds separately. For each subproblem, we can fix those interval entries of $\boldsymbol{A}$ at the corresponding lower or upper bounds depending on the signs of $\boldsymbol{B}_{ij}$. This makes the set $\boldsymbol{A}$ smaller in general. We can repeat this process or call another method for the reduced interval matrix.

Notice that there are classes of interval matrices with monotone determinant. They are called inverse stable [169]. Formally, $\boldsymbol{A}$ is inverse stable if $|A^{-1}| > 0$ for each $A \in \boldsymbol{A}$. This class also includes interval M-matrices [12], inverse nonnegative [117] or totally positive matrices [45] as particular subclasses that are efficiently recognizable; cf. [75].

8.4.6 Preconditioning

In an interval case by preconditioning we mean transforming an interval matrix into a form that is more suitable for further processing. It is generally done by multiplying an interval matrix $\boldsymbol{A}$ with some real matrices B, C from the left and right respectively.

$$\boldsymbol{A} \quad \mapsto \quad B\boldsymbol{A}C.$$

Regarding interval determinant, we have the following result.

Proposition 8.11. *Let $\boldsymbol{A}$ be a square interval matrix and let B, C be real square matrices of the corresponding size. Then*

$$\det(B)\cdot\det(\boldsymbol{A})\cdot\det(C) \subseteq \det(B\boldsymbol{A}C).$$

Proof. For any $A \in \boldsymbol{A}$ we have $\det(B)\cdot\det(A)\cdot\det(C) = \det(ABC) \in \det(B\boldsymbol{A}C)$. □

We will further use the consequence

$$\det(\boldsymbol{A}) \subseteq \frac{1}{\det(B)\cdot\det(C)} \cdot \det(B\boldsymbol{A}C).$$

There are many possibilities how to choose the matrices B, C for a square interval matrix. First, we can use the approach from [208] – take the midpoint matrix A_c and compute its LU decomposition $PA_c = LU$, where L is a lower triangular matrix having ones on the main diagonal, U is upper triangular and P is a permutation matrix. Obviously, $\det(L) = \det(L^{-1}) = 1$. Determinant of P is 1 or -1. We take $B \approx L^{-1}$ (the main diagonal of B is set to ones) and $C = I$. Then according to Proposition 8.11 we have that

$$\det(\boldsymbol{A}) \subseteq \frac{1}{\det(P)} \cdot \det(L^{-1}P\boldsymbol{A}).$$

The resulting preconditioned interval matrix should be "close" to the upper triangular matrix U. We assume that such a preconditioning might be favorable for Gaussian elimination, since the preconditioned matrix is already close to row echelon form.

For a symmetric matrix an LDL^T decomposition can be used. A symmetric matrix A can be decomposed as $A = LDL^T$, where L is upper triangular with ones on the main diagonal and D is a diagonal matrix. Similarly, as in the previous case, we set $B \approx L^{-1}, C \approx L^{-T}$ and obtain

$$\det(\boldsymbol{A}) \subseteq \det(L^{-1}\boldsymbol{A}L^{-T}).$$

The resulting preconditioned interval matrix should be "close" to the diagonal matrix D.

For solving interval linear systems, there are various preconditioners used [74, 103]. The most common choice is taking $B = A_c^{-1}$ when A_c is nonsingular and $C = I$. Such a choice of B, C is also optimal in a certain sense [137, 139]. Of course, we are computing in a finite precision arithmetic, therefore we take only some approximation $B \approx A_c^{-1}$. According to Theorem 8.11 we get

$$\det(\boldsymbol{A}) \subseteq \det(A_c^{-1}\boldsymbol{A}) / \det(A_c^{-1}).$$

Notice that the matrix A_c^{-1} does not generally have its determinant equal to 1. That is why we need to compute a verified determinant of a real matrix. We present an example of such an algorithm in the next section.

8.5 Verified determinant of a real matrix

In [145] a variety of algorithms for computation of verified determinant of real matrices is presented. We are going to use the simplest one by Rump [195]. For a real square matrix X we compute its LU decomposition using the floating point arithmetics such that

$$PX \approx LU,$$

where L is lower triangular, U is upper triangular and P is a permutation matrix following partial pivoting (therefore $\det(P) = \pm 1$). Let X_L, X_U be approximate inverses of L, U respectively. We force X_L to be lower triangular with unit main diagonal (therefore $\det(X_L) = 1$). We denote $Y := X_L P X X_U$. We enclose the coefficients of X with verified intervals and obtain an interval matrix $\boldsymbol{X}$. Therefore, the resulting matrix $\boldsymbol{Y} = X_L P \boldsymbol{X} X_U$ will be close to the identity matrix and its determinant is close to 1. To compute its determinant, we can apply, e.g., the interval version of the Gerschgorin circle theorem (Section 8.4.2). From

$$\det(Y) = \det(P)\det(X)\det(X_U).$$

we get

$$\det(X) = \frac{1}{\det(P)} \cdot \frac{\det(Y)}{\det(X_U)}.$$

We can also enclose the diagonal elements of X_U with tight intervals and compute its determinant simply as a product of these intervals. If $0 \notin \det(\boldsymbol{X}_U)$ we get

$$\det(X) \in \det(\boldsymbol{X}) \subseteq \frac{1}{\det(P)} \cdot \frac{\det(\boldsymbol{Y})}{\det(\boldsymbol{X}_U)}.$$

8.6 Enclosure of a determinant: special cases

Even though we are not going to compare all of the mentioned methods in this section, for the sake of completeness, we will mention some cases of matrices, that enable the use of another tools. For some classes of interval matrices tasks connected to determinants are computable efficiently.

8.6.1 Symmetric matrices

Many problems in practical use are described by symmetric matrices. In connection with determinant a new approach can be used. We specify what we mean by an interval symmetric matrix in the following definition.

Definition 8.12 (Symmetric interval matrix)**.** For a square interval matrix $\boldsymbol{A}$ we define the symmetric matrix $\boldsymbol{A}^S$ as

$$\boldsymbol{A}^S = \{A \in \boldsymbol{A} \mid A = A^T\}.$$

Its eigenvalues are defined as follows.

Definition 8.13. For a real symmetric matrix A let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$ be its eigenvalues. For $\boldsymbol{A}^S$ we define its ith set of eigenvalues as $\boldsymbol{\lambda}_i(\boldsymbol{A}^S) = \{\lambda_i(A) \mid A \in \boldsymbol{A}^S\}$.

For symmetric interval matrices there exist various methods to enclose each ith set of eigenvalues. A proposition by Rohn [175] gives a simple enclosure.

Proposition 8.14. $\boldsymbol{\lambda}_i(\boldsymbol{A}^S) \subseteq [\lambda_i(A_c) - \varrho(A_\Delta), \lambda_i(A_c) + \varrho(A_\Delta)]$.

The previous proposition requires computation of verified enclosures of eigenvalues of real matrices; for more details on such an issue see, e.g., [128, 129, 221].

There exist various other approaches for computing enclosures of the eigenvalues (e.g., [107, 119]), there are several iterative improvement methods (e.g., [15, 79]). For the exact minimum and maximum extremal eigenvalues, there is a closed-form expression [64], which is however exponential.

8.6.2 Symmetric positive definite matrices

Let $\boldsymbol{A}^S$ be a symmetric (strongly) positive definite matrix, that is, every $A \in \boldsymbol{A}^S$ is positive definite. For more details about positive definite matrices see Section 11.10.

The matrix with maximum determinant can be found by solving the optimization problem

$$\max\ \log\det(A) \text{ subject to } A \in \boldsymbol{A}^S.$$

The condition $A \in \boldsymbol{A}^S$ can be rewritten as linear conditions

$$\forall i,j \quad \underline{a}_{ij} \le a_{ij} \le \overline{a}_{ij}, \quad \forall i \neq j \quad a_{ij} = a_{ji},$$

and the function $\log\det(A)$ is a so-called *self-concordant* function for which such an optimization problem is solvable in polynomial time with respect to dimension of a problem and $1/\varepsilon$ (where ε is a desired accuracy) using interior point methods; see Boyd and Vandenberghe [22]. Therefore, we have:

Proposition 8.15. *The maximum determinant of a symmetric positive definite matrix is computable in polynomial time.*

8.6.3 Matrices with $A_c = I$

Preconditioning $\boldsymbol{A}$ by A_c^{-1} results in an interval matrix with I as the midpoint matrix. We saw that such matrices imply favorable properties (polynomial hull computation – Subsection 5.6.2, nicer sufficient conditions for regularity – Section 4.1).

Proposition 8.16. *Suppose that $\varrho(A_\Delta) < 1$. Then the minimum determinant of $\boldsymbol{A}$ is attained for $\underline{A}$.*

Proof. According to Corollary 4.3 the fact $\varrho(A_\Delta) < 1$ implies regularity of $\boldsymbol{A}$; and also of $\underline{A}$.

We will proceed by mathematical induction. For $n = 1$ the proof is trivial. For a general case, we express the determinant of $A \in \boldsymbol{A}$ as in (8.3)

$$\det(A) = \det(A_{2:n})/x_1. \tag{8.4}$$

Notice that $\boldsymbol{A}$ and $\boldsymbol{A}_{2:n}$ have identity matrices as midpoints, whose determinant is equal to 1. Regularity of every $A \in \boldsymbol{A}$, and hence of $A_{2:n} \in \boldsymbol{A}_{2:n}$, then implies

$$\det(A) > 0, \quad \det(A_{2:n}) > 0.$$

Therefore, we know that also $x_1 > 0$. To obtain lower bound on $\det(\boldsymbol{A})$ we need to minimize the numerator and maximize the denominator of (8.4). By induction hypothesis, the smallest value of $\det(A_{2:n})$ is attained for $A_{2:n} = \underline{A}_{2:n}$. The solution x of $Ax = e_1$ is the first column of A^{-1}. From Theorem 11.21 it follows that the upper bound on $\boldsymbol{A}^{-1}_{*1}$ is obtained by setting $A = (I - A_\Delta) = \underline{A}$. Therefore $A = \underline{A}$ simultaneously minimizes the numerator and maximizes the denominator of (8.4). □

Example 8.17. If the condition $\varrho(A_\Delta) < 1$ does not hold, then the claim is generally wrong. Let us have the matrix $\boldsymbol{A} = [A_c - A_\Delta, A_c + A_\Delta]$ where

$$A_c = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad A_\Delta = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

we have $\varrho(A_\Delta) = 3$ and $\det(\underline{A}) = -2$, however, the $\det(\boldsymbol{A}) = [-6, 14]$. The minimum bound is attained, e.g., for the matrix

$$\begin{pmatrix} 0 & -1 & 1 \\ -1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}.$$

The reasoning from the proof of Theorem 8.16 cannot be applied for computing the upper bound of $\det(\boldsymbol{A})$.

Example 8.18. For the matrix $\boldsymbol{A} = [A_c - A_\Delta, A_c + A_\Delta]$ where

$$A_c = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad A_\Delta = \frac{1}{4} \cdot \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix},$$

we have $\varrho(A_\Delta) = 0.75 < 1$ and $\det(\boldsymbol{A}) = [0.25, 2.1875]$. However, $\det(\overline{A}) = 1.75$.

Computing the maximum determinant of $\boldsymbol{A}$ is a more challenging problem. It is an open question whether is can be done in polynomial time. Obviously, the maximum determinant of $\boldsymbol{A}$ is attained for a matrix $A \in \boldsymbol{A}$ such that $A_{ii} = \overline{A}_{ii}$ for each i. Specifying the off-diagonal entries is, however, not so easy.

8.6.4 Tridiagonal H-matrices

Consider an interval tridiagonal matrix

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{a}_1 & \boldsymbol{b}_2 & 0 & \dots & 0 \\ \boldsymbol{c}_2 & \boldsymbol{a}_2 & \boldsymbol{b}_3 & \ddots & \vdots \\ 0 & \boldsymbol{c}_3 & \boldsymbol{a}_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \boldsymbol{b}_n \\ 0 & \dots & 0 & \boldsymbol{c}_n & \boldsymbol{a}_n \end{pmatrix}.$$

Suppose that it is an interval H-matrix, which means that each matrix $A \in \boldsymbol{A}$ is an H-matrix (for a definition see Section 4.4). Without loss of generality let us assume that the main diagonal is positive, that is, $\underline{a}_i > 0$ for all $i = 1, \ldots, n$. Otherwise, we can multiply the corresponding rows by -1.

Recall that the determinant D_n of such a real tridiagonal matrix of order n can be computed by the recursive formula

$$D_n = a_n D_{n-1} - b_n c_n D_{n-2}.$$

Since $\boldsymbol{A}$ is an H-matrix with positive diagonal, the values of $D_1, \ldots, D_n$ are positive for each $A \in \boldsymbol{A}$ (see, e.g., [19]). Hence the largest value of $\det(A)$ is attained at $a_i := \overline{a}_i$ and b_i, c_i such that $b_i c_i = \underline{\boldsymbol{b}_i \boldsymbol{c}_i}$. Analogously for the minimal value of $\det(A)$. Hence we constructively proved the following proposition.

Proposition 8.19. *Determinants of interval tridiagonal H-matrices are computable in polynomial time.*

Complexity of determinant computation for general tridiagonal matrices remains an open problem, similarly as solving an interval system with tridiagonal matrix [112]. Nevertheless, not all problems regarding tridiagonal matrices are open or hard, e.g., deciding whether a tridiagonal matrix is regular can be done in polynomial time [11].

8.7 Comparison of methods

In this section some of the previously described methods are compared. First, we start with general square matrices. Then we test on symmetric matrices. All the tests were computed using the DESKTOP setup (see Section 3.11).

8.7.1 General case

For general matrices the following methods are compared:

- `ge` - interval Gaussian elimination,
- `cram` - our method based on Cramer's rule with HBR method for solving square interval systems,
- `had` - interval Hadamard's inequality,
- `gersch` - interval Gerschgorin circles.

The suffix `+inv` is added when the preconditioning with midpoint inverse was applied and the suffix `+lu` is added when the preconditioning based on LU decomposition was used. We use the label `hull` to denote the exact interval determinant.

Table 8.1: Enclosures of determinants from Example 8.20. Bounds of the enclosures are rounded off to 3 decimal digits. Fixed radii of intervals are denoted by r.

method	$r = 0.2$	$r = 0.1$	$r = 0.01$
`hull`	[-0.6, 21.72]	[4.06, 14.88]	[8.465, 9.545]
`ge`	[-29.25, 87.75]	[3.000, 21.857]	[8.275, 9.789]
`ge+inv`	$[-\infty, \infty]$	[3.6, 18]	[8.46, 9.56]
`ge+lu`	[-99.45, 134.55]	[1.44, 22.482]	[8.244, 9.791]
`cram`	$[-\infty, \infty]$	[3.01, 23.143]	[8.326, 9.722]
`cram+inv`	$[-\infty, \infty]$	[3.6, 17.067]	[8.46, 9.56]
`cram+lu`	$[-\infty, \infty]$	[1.44, 21.434]	[8.244, 9.79]
`had`	[-564.788, 564.788]	[-526.712, 526.712]	[-493.855, 493.855]
`had+inv`	[-30.048, 30.048]	[-16.801, 16.801]	[-9.563, 9.563]
`had+lu`	[-46.178, 46.178]	[-35.052, 35.052]	[-27.019, 27.019]
`gersch`	[-3371.016, 11543.176]	[-3132.927, 11089.567]	[-2926.485, 10691.619]
`gersch+inv`	[-81, 243]	[0, 72]	[6.561, 11.979]
`gersch+lu`	[-11543.176, 6435.576]	[-11089.567, 6116.667]	[-10691.619, 5838.41]

Example 8.20. To obtain a general idea how the methods work, we can use the following example. Let us take the midpoint matrix

$$A_c = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 6 & 7 \\ 5 & 9 & 8 \end{pmatrix},$$

and inflate it into an interval matrix using three fixed radii of intervals $0.2, 0.1$ and 0.01 respectively and test all the mentioned methods. The resulting enclosures of the determinants are shown in Table 8.1.

The previous example shows a case where the `lu` preconditioning gives better results for `ge` than the `inv` preconditioning. However when testing for larger matrices the determinant enclosure using the `lu` preconditioning tends to be infinite too. From the above example we see that for a general matrix preconditioning is favorable. That is why we later test only `ge+inv`, `cram+inv`, `had+inv` and `gersch+inv` methods.

We can perceive the method `ge+inv` used in [208] as the “state-of-the-art” method. Therefore, every other method will be compared to it.

All methods are tested on randomly generated matrices of sizes $n = 10, \dots, 60$. To generate an interval matrix a real midpoint matrix is randomly generated with coefficients selected independently and uniformly from $[-1, 1]$. Then, such a matrix is inflated into an interval matrix by wrapping the coefficients with intervals of a given fixed radius. Here we choose the radii ($r = 10^{-3}$ and $r = 10^{-5}$). For each size and radius we test on 100 matrices.

For each radius, size and method an average ratio of computed enclosures and average computation time are computed. We compute the ratios according to the

Table 8.2: Number of infinite enclosures returned by various method (out of 100) for fixed radii $r = 10^{-5}$ and $r = 10^{-3}$ respectively. The sizes of matrices are denoted by n.

n	`cram+inv`	`ge+inv`	`ge+inv`	`cram+inv`
10	0	0	4	2
20	0	0	15	15
30	0	0	10	10
40	0	0	34	31
50	0	0	38	38
60	2	2	54	51
r	10^{-5}		10^{-3}	

formula (3.8). If the average ratio is < 1 it means a methods returned narrower results than `ge+inv`. It can happen that an enclosure returned by a method is infinite. Such case is omitted from the computation of the average. The occurrence of such a phenomenon is captured in Table 8.2. We can see that for smaller radii it happens only rarely. The methods `had+inv` and `gersch+inv` never returned an infinite enclosure.

Average ratios of widths are presented in Table 8.3. When the ratio is a number less then 1000, it is displayed rounded off to 2 decimal digits. When it is greater, only the approximation 10^x is displayed. To accentuate the similarity of the results returned by `ge+inv` and `cram+inv`, their ratio of enclosures is rounded off to 6 decimal digits. With increasing size of a system (and also with increasing overestimation of `ge+inv` and `cram+inv`) the ratio difference of `had+inv` becomes less apparent.

Average computation times for $r = 10^{-5}$ are displayed in Table 8.4. Since the methods are basically direct (except for verified inverse computation in HBR method), the computation times for $r = 10^{-3}$ are very similar. The method `cram+inv` is significantly faster than `ge+inv`. To more clearly see the difference in computation times between the two most efficient methods `ge+inv` and `cram+inv` see Figure 8.6.

8.7.2 Symmetric matrices

We repeat the same test procedure for symmetric interval matrices. Symmetric matrices are generated in a similar way as before, only they are shaped to be symmetric (the lower triangle of a matrix is mirrored to the upper triangle). We again compare the `ge+inv`, `gersch+inv`, `had+inv` and `cram+inv`. We add one new method `eig` that is based on computation of enclosures of eigenvalues using the Rohn's simple enclosure (Proposition 8.14). The method `ge+inv` stays the reference method, i.e, we compare all methods with respect to this method.

The ratios of enclosures widths for symmetric matrices are displayed in Table 8.5 and Table 8.6. We can see that as in the general case the results of `cramer+inv` are very similar to `ge+inv`. When $r = 10^{-3}$ the overestimation by `had+inv` becomes smaller than on `eig` at a certain point ($n = 40$).

Table 8.3: Average ratios of widths of enclosures returned by various methods for interval matrices with fixed radii 10^{-5} and 10^{-3} respectively. The methods are compared to `ge+inv`, the sizes of matrices are denoted by n.

n	`gersch+inv`	`had+inv`	`cram+inv`	`gersch+inv`	`had+inv`	`cram+inv`
10	35.34	10^3	1.000100	10^3	14.58	0.999974
20	50.16	10^3	1.000000	10^9	6.18	1.000911
30	10^9	257.45	1.000010	10^{18}	3.78	1.006991
40	10^4	178.02	0.999999	10^{24}	2.52	0.998934
50	10^{25}	117.85	1.000049	10^{29}	2.06	1.001731
60	10^{24}	101.19	0.999980	10^{40}	1.46	1.002089
r		10^{-5}			10^{-3}	

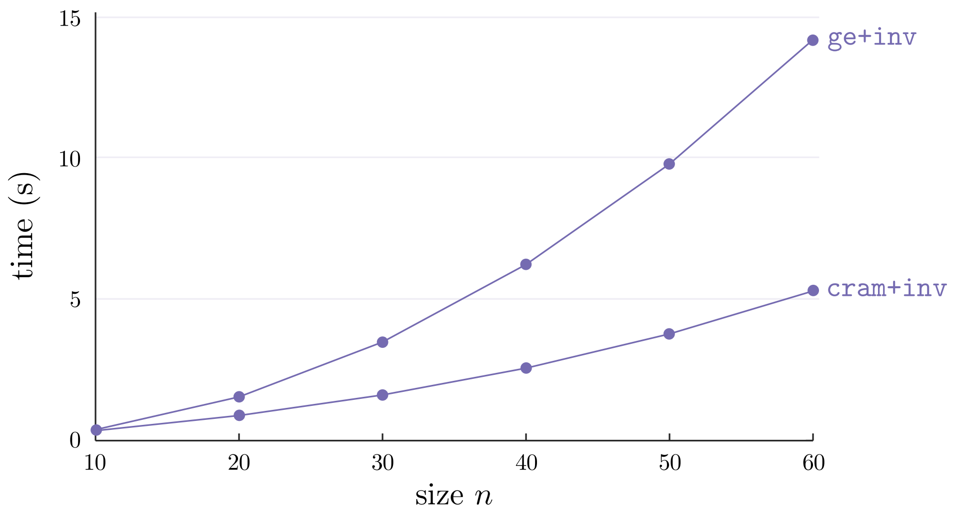


Figure 8.6: Visual comparison of average computation times (in seconds) of `ge+inv` and `cram+inv` for various matrix sizes n.

Table 8.4: Average computation times (in seconds) of various methods for fixed radii 10^{-5} and various sizes of matrices n.

n	`gersch+inv`	`had+inv`	`ge+inv`	`cram+inv`
10	0.04	0.01	0.39	0.36
20	0.06	0.01	1.58	0.90
30	0.09	0.02	3.56	1.64
40	0.12	0.02	6.34	2.59
50	0.15	0.04	9.95	3.80
60	0.19	0.05	14.37	5.32

Table 8.5: Average ratios of widths of enclosures returned by various methods for symmetric matrices with fixed radii $r = 10^{-5}$. The reference method is `ge+inv`, the sizes of matrices are denoted by n.

n	`gersch+inv`	`had+inv`	`cram+inv`	`eig`
10	18.50	10^3	1.000000	2.62
20	50.51	10^3	0.999999	3.07
30	108.66	10^3	1.000000	3.23
40	126.79	250.03	1.000000	3.59
50	10^9	166.60	1.000001	3.63
60	10^{11}	117.62	0.999999	3.63

Table 8.6: Average ratios of widths of enclosures returned by various methods for symmetric matrices with fixed radii $r = 10^{-3}$. The reference method is `ge+inv`, the sizes of matrices are denoted by n.

n	`gersch+inv`	`had+inv`	`cram+inv`	`eig`
10	242.93	19.48	1.000637	2.49
20	10^9	7.69	0.999870	2.88
30	10^{15}	4.16	0.998364	2.96
40	10^{22}	3.31	0.995946	3.56
50	10^{26}	2.56	1.000100	4.18
60	10^{33}	2.04	1.002171	4.99

Table 8.7: Average computation times (in seconds) of various methods for symmetric matrices with fixed radii 10^{-5}. The sizes of matrices are denoted by n.

n	`gersch+inv`	`had+inv`	`ge+inv`	`cram+inv`	`eig`
10	0.04	0.01	0.39	0.35	0.02
20	0.06	0.01	1.56	0.89	0.03
30	0.09	0.02	3.51	1.62	0.04
40	0.12	0.03	6.26	2.56	0.07
50	0.15	0.04	9.82	3.77	0.10
60	0.19	0.05	14.20	5.29	0.15

The average computation times are displayed in Table 8.7. We can see that `eig` shows slightly higher computational demands than `had`. In case of $r = 10^{-3}$ and $n \geq 40$ it pays off to use rather `had+inv` than `eig`. However, for $r = 10^{-5}$ "reasonable" overestimation in a fraction of `cram+inv` computation time is obtained. The method `eig` was based on a simple enclosure. That explains the low computational time. Of course, it is possible to use, e.g., filtering methods to obtain even tighter enclosures of eigenvalues. However, they work well in specific cases [79] and the filtering is much more time consuming.

8.7.3 Summary

It is always the question of the payoff between computation speed and quality of enclosure. Based on the tests from the previous subsections, we recommend to use `cram+inv` method, since it produces equivalent results to `ge+inv` in much less computational time.

9 Application of intervals to medical data

- Multiple breath washout test
- Finding breath ends
- Regression on interval data
- Interval regression with integer data matrix
- Application to medical data
- Hypotheses

This chapter is based on results from a joint research project of Department of Applied mathematics, Faculty of Mathematics and Physics and Department of Paediatrics, 2nd Faculty of Medicine at Charles University, Prague, especially, on collaboration with Václav Koucký. The author of this work was the head researcher of this project. First, we introduce the medical background of the project. Then, we discuss interval regressions and how to improve them for the sake of our problem. The conclusions from this project are still in a form of hypotheses that need to be further verified or rejected. However, this work might contribute to the ongoing discussions related to these topics. Some of our initial (rather too optimistic) ideas were published in [88]. More detailed results are contained in our unpublished work [90]. The algorithm for finding breath ends is published in [89].

9.1 Multiple Breath Washout test

First works concerning multiple breath washout test (MBW) date back to 40s or 50s [28]. In those days the method faced crucial limits. The precision of sensors was not satisfactory and also the computational power of digital computers was insufficient to handle problems described with too many parameters (much of mathematical work was still done manually). With increasing power of sensors and computers MBW received its rebirth in 90s.

MBW is a very promising method since it does not require any specific breathing maneuvers. The only necessity is the ability to breathe normally with regular pattern, which makes it applicable to the variety of age scale. Small infants usually undergo this procedure in artificial sleep.

In contrast to classical methods (e.g., spirometry, bodypletysmography) MBW is

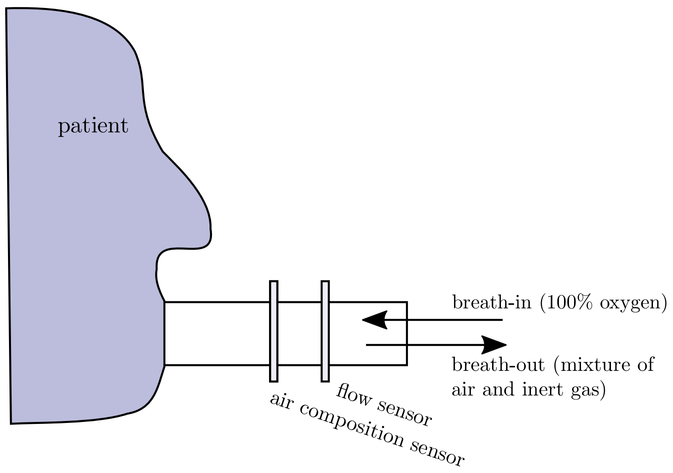


Figure 9.1: Schematic depiction of the washout phase.

able to evaluate even the most peripheral airway. The high sensitivity to the most peripheral airway changes has been shown in most of chronic lung diseases (e.g., bronchial asthma, cystic fibrosis, primary cilliary dyskinesia, etc.) [33, 56, 121].

The test consists of two phases – the washin and washout. During the first phase, lung is filled with an inert gas (sulphur hexafluoride SF_6, helium He or nitrogen N_2), during the second phase, the inert gas is washed out by air or by 100% oxygen (depending on the inert gas used). Concentration of the respective inert gas, volume of exhaled gas and flow are measured online. The measurement is stopped after reaching a certain concentration of innert gas within lung (usually 2.5%). The pattern of inert gas concentration decrease gives information about the homogeneity of ventilation and thus about the patency of airways. The washout phase is depicted in Figure 9.1.

In our work we focus on use of nitrogen (N_2) as inert gas. Although, the SF_6 has been historically used for much longer in practice, use of nitrogen has many advantageous properties:

- SF_6 can potentially have narcotic effects,
- SF_6 is not used in medicine, so it must be specially prefabricated, N_2 is naturally present in the surrounding air,
- N_2 is naturally present in the surrounding air and also in lung, that is why there is no need for washin phase,
- N_2 is present also in poorly ventilated areas of lung.

A small draw back is that there are questions whether N_2 is really an inert gas because of its absorbance and ocurence in tissues. During a measurement, N_2 returned back from tissues can influence the real measured concentrations of N_2, especially for infants. That is why the method is recommended for patients older than 6 months.

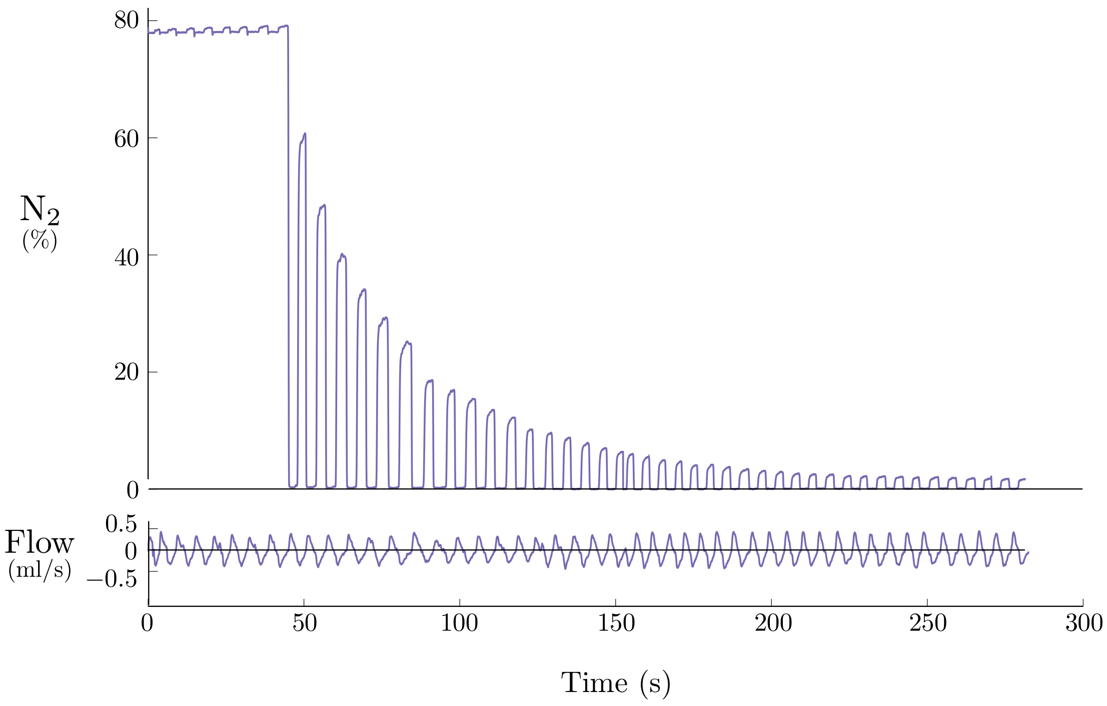


Figure 9.2: Nitrogen concentration (the top curve) and air flow (the bottom curve) in time measured during the nitrogen washout process.

The main output of the measurement is depicted in Figure 9.5. There are two main graphs – actual flow (the bottom curve) and decreasing nitrogen concentration (the top curve) measured in each *time-slice* (here it is 5ms). These data are further used for computing clinically significant indices (FRC, LCI, Scond, Sacin, etc.). Some of them will be mentioned later. The sensibility of MBW can detect a pathology in its early stages, which enables to start the cure early and with greater effect.

9.2 LCI and FRC

Currently, the most important indices calculated from MBW data are FRC and LCI. If we omit the deadspace correction, the *functional residual capacity* (FRC) is calculated as

$$\mathrm{FRC} = \frac{\mathrm{N}_{out}}{\mathrm{N}_{start} - \mathrm{N}_{end}},$$

where N_{out} is the total volume of expired N_2; N_{start} and N_{end} are concentrations of nitrogen at initial and terminal peak respectively. The FRC relates to the size of lung. The *lung clearance index* (LCI) is calculated simply as

$$\mathrm{LCI} = \frac{\mathrm{V}_{out}}{\mathrm{FRC}},$$

where V_{out} is the total volume of expired air. It is necessary to specify how we decide the terminal breath. Usually, a measurement is stopped when the concentration of nitrogen in peaks decreases below 2.5% of the initial nitrogen concentration. This level is chosen historically. The terminal peak is defined to be the first of three consequent peaks with concentration below 2.5% of the initial nitrogen concentration. The corresponding LCI index is then marked LCI2.5. It states how many air volumes (equal to FCR) exchanges are necessary to clean the lung from the inert gas (more specifically to reach the level of 2.5% of initial inert gas concentration). LCI index seems to be very useful to evaluate the homogeneity of lung ventilation (the most peripheral airways included).

Completing the washout process up to 2.5% might be too time consuming, which makes it difficult for uncooperative patients to finish the MBW test properly. That is why there are being discussions about use of the level 5%.

9.3 Our data

We collected the data according to proper conditions for valid measurement defined in [101]. The three necessary conditions are:

- a patient has sufficiently regular breathing pattern during measurement,
- there is no leakage during the measurement,
- washout phase is finished (nitrogen is washed out at least to a given level).

The measurements were approved by the ethical committee of Motol University Hospital, Prague, Czech republic. Patients (or their parents in case of infants) were informed about the measurement before the tests.

In all data files the peaks of the nitrogen concentrations have been identified using our own algorithm described in the next section.

9.4 Finding breath ends

Breath detection (i.e., finding the spot where an expiration ends and an inspiration starts) is a crucial step in pulmonary function testing (PFT). It is a starting point for computing various clinically significant indices, performing regression analyses or making predictions. With the increasing importance of PFT as a diagnostic tool, new methods of PFT and approaches to data analysis are required especially in infants and toddlers (i.e., uncooperative children). In this age category, precise raw data analysis is of utmost importance, as the PFT is very prone to technical errors. Based on our clinical experience, the current PFT algorithms suffer from severe imprecisions, which may lead to difficult and time-consuming interpretation of results or even raw data rejection.

Although breath detection is a relatively easy task for a physician as a human being, automated detection by computer remains a challenge, especially in cases of severely distorted data (e.g., as a result of patients' insufficient cooperation, severe volume drift, etc.). An approach to the breath detection analysis is primarily determined by the signals being measured. Usually, a time-flow signal is captured. In this situation, two basic algorithms for breath detection have been proposed – threshold and smoothing approach, each with numerous modifications and extensions increasing their reliability and accuracy [13]. The threshold approach outputs any breath having parameters above a pre-set threshold. On the other hand, the smoothing approach smooths the signal to eliminate spurious breath endings. Despite the significant progress done in this field, clinicians are still facing situations in which the measured signal is too distorted to be automatically analyzed.

In comparison with the "conventional" methods that are based solely on flow, volume and pressure measurements and estimated primarily airway resistance (e.g., bodypletysmography, tidal breath analysis, etc.), MBW brings a new dimension to raw data – the gas concentration signal (O_2, CO_2 , inert gas). A current commercial software (Spiroware, Ecomedics, Duernten, Switzerland) uses concentrations only for constructing washout curves. However, this information may be also used for breath detection. The aim of our study [89] was to design and justify a new and robust algorithm for breath detection using not only time-flow data but also gas concentration signal. Such a breath detection algorithm can significantly outperform the current threshold-based algorithms. Moreover, its key ideas have the potential to contribute to the general design of the medical algorithms.

9.4.1 Our algorithm

Our algorithm (`Alg-OUR`) was programmed in the free software GNU Octave, version 4.0.0. and works in several steps, which are outlined below. A depiction of each step can be found in Figure 9.3.

Algorithm 9.1 (Breath end detection (`Alg-OUR`))**.** The input is raw flow and CO_2 concentration data in time. The algorithm outputs integer intervals containing numbers of zero-crossings corresponding to one breath end.

1. Load raw data.

2. Zero-crossings detection – a zero-crossing is defined as a time spot, where the air flow changes its direction from minus to plus (see a comment on general physiology of respiratory tract in Subsection 9.4.4). All the zero-crossings in flow raw data are detected and numbered from 1 to N, where N is the total number of zero-crossings. They form a set of potential breath ends.

3. For each $-/+$ zero-crossing at time T, the nearest peak of CO_2 curve (i.e., local maximum) is found and attributed to the time T.

4. The volume of each inhalation and exhalation (V_{in}, V_{out}) corresponding to the time T is calculated by integration of the flow curve (using simple trapezoidal rule, similarly as in Example 2.4).

5. The zero-crossings with corresponding CO_2 peaks of insufficient concentration (i.e., less than 2% – see a comment in Subsection 9.4.4) are discarded; the numbering of zero-crossings is preserved. Next, our goal is to discard zero-crossings

that do not form a breath end or capture intervals of zero-crossings that belong to the same breath. Initially, we view each zero-crossing to be a singleton interval ($[b, b]$). Next, the algorithm is going to discard or merge some of these intervals (steps 6 to 8).

6. Two intervals of zero-crossings $[a, b]$ and $[c, d]$ are merged if the CO_2 concentration between b and c does not drop below 0.5%. Consequently, a new interval $[a, d]$ instead of the previous two is created. This process is repeated until there exists no such a pair of intervals. Note that in an interval $[a, b]$, a can be equal to b.

7. The two consecutive intervals of zero-crossings $[a, b]$ and $[c, d]$ where $c = b + 1$ are merged if the ratio of volumes V_{in}/V_{out} for zero-crossing b is greater than 5 (see the comment in section Discussion). This process is repeated until there exists no such a pair of intervals.

8. The upper bounds of the remaining intervals (even tight ones - $[a, a]$) are marked as the breath ends (i.e., from $[a, b]$, it is b, from $[a, a]$, it is a).

(!) Note that the order of the steps 5, 6 and 7 cannot be changed; otherwise the algorithm produces incorrect results.

For the sake of comparison, the most commonly used flow threshold algorithm (originally described in [216]) were implemented in our software. Two different thresholds (`Alg3-0.01` and `Alg3-0.25`) according to the age of the patient and an additional plausibility check were used as specified in [13], [198] and [216].

9.4.2 Test data characteristics

To test the clinical usefulness and accuracy of our newly developed algorithm, we compared it with representatives of the currently used algorithms on real patient data. We intentionally selected severely distorted measurements, which are, in our experience, very difficult to be automatically analyzed by the current software. In total, 47 anonymized traces (A-files) coming from 19 patients were enrolled. Such an approach was in general approved by the local ethics committee. Patients' characteristics are stated in Table 9.1. The rationale for intentional selection of severely distorted data was the fact, that only severely distorted data offer the possibility to test the performance of breath detection algorithms properly. Analysis of regular breathing is no challenge for current breath detection approaches anymore.

9.4.3 Comparison of algorithms

The raw data were analysed in four different ways:

1. analysis performed by our algorithm described above (`Alg-OUR`),

2. analysis performed by the previously described algorithms (`Alg3-0,01` and `Alg3-0,25`) that are implemented in our software,

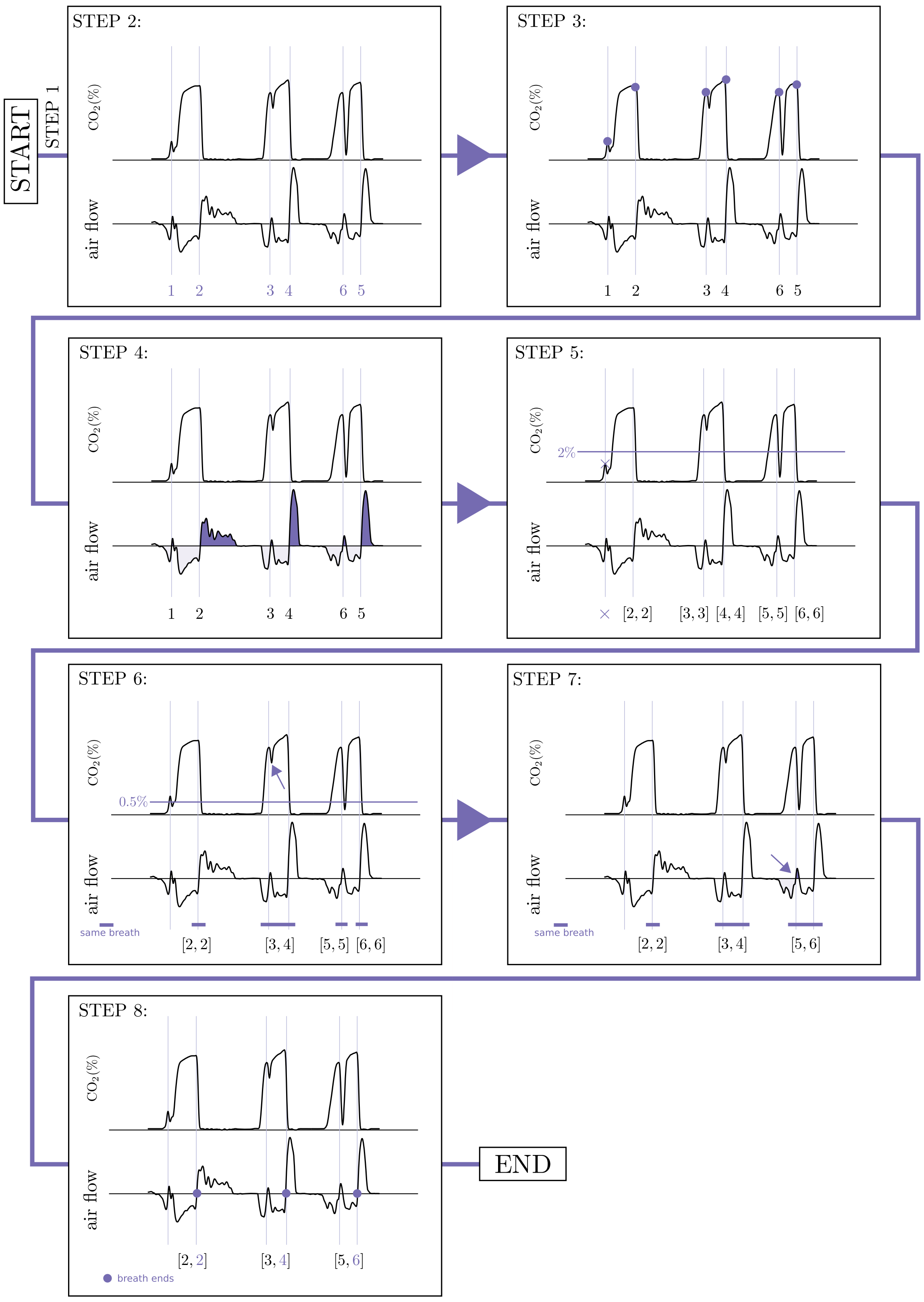


Figure 9.3: Flow diagram and depiction of each step of our breath detection algorithm (`Alg-OUR`).

Table 9.1: Characterization of the patients. Their A-files were used for the sake of the comparison of breath detection algorithms.

General information	Number of patients (male)	19(9)
	Number of A-files	47
	Age (mean ± SD) [years]	6.6 ± 5.6
	Weight (mean ± SD) [kg]	26.6 ± 19.0
	Weight z-score (mean ± SD)	0.02 ± 1.2
	Height (mean ± SD) [cm]	114.6 ± 33.2
	Height z-score (mean ± SD)	-0.4 ± 1.3
Diagnoses	nonrespiratory problematic	4
	cystic fibrosis	7
	primary ciliary dyskinesia	2
	repeated obstructive bronchitis	4
	miscellaneous	2

3. analysis performed by the commercial package Spiroware 3.2.0 (`Alg-Spi`),
4. manual analysis performed by two specialists experienced in PFT.

After loading an A-file into our software, the number of breaths detected by `Alg-OUR`, `Alg3-0,01` and `Alg3-0,25` were calculated. The A-file was also loaded into Spiroware and the number of breaths was estimated using the functionality of this commercial software. Afterwards, two PFT specialists inspected the data from each A-file independently. The inspection was done in the interface of our software, created for this purpose. It enables visualization of flow, volume and CO_2 concentration, while at the same time visualisation of breath ends found by the respective algorithms. Such visualization enables both the estimation of the number of true breaths (reference number of breaths – RNB) and simultaneously the localization of falsely positive/negative breaths as analyzed by different algorithms.

All the A-files included in our testing could be successfully analysed by all the implemented algorithms. The analysis time was longer for `Alg-OUR` than for the threshold algorithms (1.35 ± 0.23s vs. 0.12 ± 0.01s, $p < 0.001$). The manual analysis took much longer; the average analysis time was roughly estimated to be between 100 and 180s.

The two specialists in PFT working independently detected the same number of breaths in 35 out of 47 A-files (74%). In the remaining cases, differences were not larger than two breaths. These cases were reanalyzed by the two specialists jointly in order to reach consensus and the "reference number of breaths" (RNB) was assigned to each A-file. Finally, 2861 true breaths in 47 A files were included.

The agreement between the algorithm `Alg-OUR` and RNB was in 70.2% files (33 out of 47 A-files), the maximal difference between the result of `Alg-OUR` and RNB was 7 breaths. The falsely positive breaths (i.e., zero-crossings misinterpreted as breath

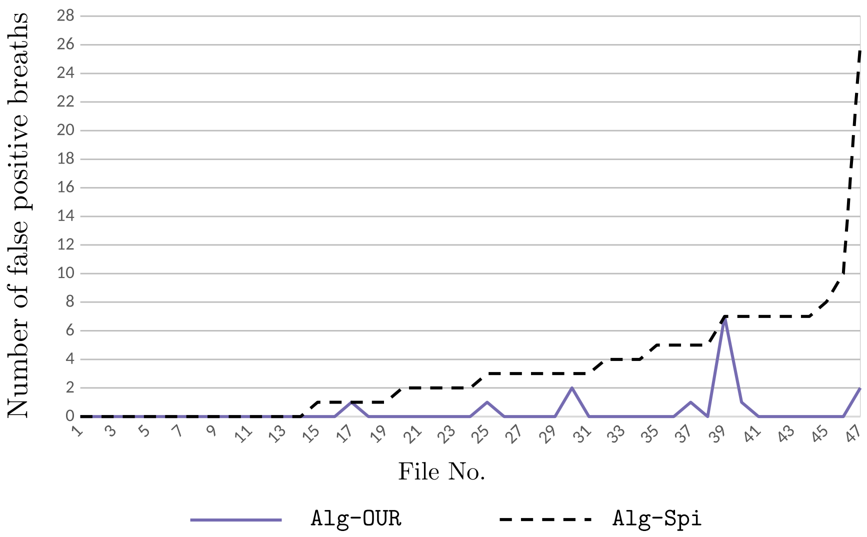


Figure 9.4: Number of false positive breaths detected in each A-file by **`Alg-OUR`** and **`Alg-Spi`**. The dashed line displays number of false positive breaths for **`Alg-Spi`**, the solid line displays the false positive breaths for **`Alg-OUR`**. The A-files are ordered (numbered) according to the increasing amount of false positive breaths detected by **`Alg-Spi`**.

ends) were the most prominent issue of this automated breath end detection. On the other hand, its sensitivity was high enough not to miss any true breath (number of falsely negative breath ends was equal to zero in the whole analysis). All other algorithms were clearly less effective. The agreement between `Alg-Spi` and RNB was only 17.0% files (8 A-files); the maximum difference in breaths detected was 26 breaths, no falsely negative breath was detected. `Alg3-0,01` suffered severely from the false positive breath detection, even in the youngest age category (toddlers under 3 years). Agreement between `Alg3-0,01` and RNB was reached only in 4 cases (8.5% files), no false negative breath was detected. On the other hand, `Alg3-0,25` showed tendency to miss true breaths (so called false negative breaths), especially in the youngest age category. In adolescents older than 15 years, the agreement with RNB was much higher (55.6% files).

In total, there was 2861 reference breaths. Our algorithm successfully detected all of them (100%). It detected no false negative breaths (0%). Our algorithm returned 2876 breath ends, hence it returned 15 false positive breaths (0.52 %). Later, we are going to use these numbers to compare our algorithm with other published methods for finding breath ends, since it is a commonly used measure in most of the cited papers.

Related to `Alg-Spi`, the higher effectiveness of `Alg-OUR` in comparison with `Alg-Spi` is clearly demonstrated in Figure 9.4. Note that there was no A-file for which the `Alg-OUR` was less effective than `Alg-Spi`. Additionally, the performance of the two algorithms (`Alg-OUR` and `Alg-Spi`) was compared with RNB using the Wilcoxon paired test. `Alg-OUR` did not detect significantly different numbers of breaths in comparison with RNB ($p = 0.789$), while `Alg-Spi` did ($p < 0.001$).

9.4.4 Final thoughts on our algorithm

We proposed an innovative algorithm for breath detection that has similar accuracy to that of human experts. In comparison with the existing threshold-based algorithms and commercial software algorithm, it exhibits significantly higher success rate in recognition of true breaths, especially in severely distorted data. The algorithm addresses both the problems of false negative and positive breath detection. We are convinced that the higher performance is caused by a simultaneous use of more types of information obtained from the measurement and by respecting the basic facts of respiratory tract physiology. The main characteristics taken into account when designing our algorithm were:

1. The breath end corresponds to the time spot, when the inspiration starts and expiration ends or vice versa. Consequently, the direction of flow must change. This is the crucial presumption that we implemented it in step 2 of Algorithm 9.1.

2. During the expiration, carbon dioxide, which is being produced by body metabolism, is eliminated from the alveoli. Consequently, CO_2 concentration in exhaled air increases up to 6%. Its concentration during the expiration needs to be at least 2%, otherwise CO_2 will cumulate in the body, which will lead to respiratory failure. This characteristic is reflected in step 5 of Algorithm 9.1. It allows for the elimination of false breaths like breath (A) in Figure 9.5.

3. Carbon dioxide concentration in atmospheric (inhaled) air is approximately 0.04%. Consequently its concentration between two subsequent zero-crossings that both correspond to the true breath ends must drop close to this level. The level of 0.5% was chosen to safely allow for minor technical issues such as time shift of signals. This characteristic is reflected in step 6 Algorithm 9.1. It discards the zero-crossing (C) or earlier discarded zero-crossing (A) in Figure 9.5.

4. Volume of inhaled air must be approximately the same as the volume of exhaled air. In case when these volumes differ by more than 5 times, severe hyperinflation or detrimental changes to residual volume would occur. This is not attributable to physiologic tidal breathing. This characteristic is reflected in step 7 of Algorithm 9.1) and would discard the zero-crossing (E) in Figure 9.5.

These are the only theoretical assumptions used in our algorithm. No standalone assumption is universal, i.e., it is not sufficient to eliminate all false breath ends. However, appropriate combination and sequence of these conditions does the job very well.

In comparison with the previously published algorithms [13], [216] and [198] and with the threshold one implemented in Exhalyzer D, `Alg-OUR` introduced several unique features:

- **No preset thresholds** – as the algorithm is based only on generally valid assumptions from respiratory tract physiology, it does not require any pre-set threshold or other patient specific limitations. Our algorithm is applicable in all

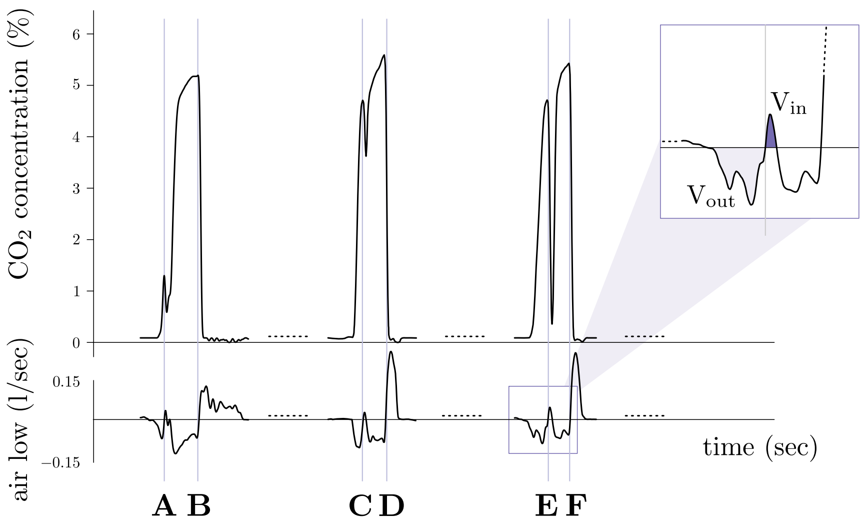


Figure 9.5: Possible cases of the shape of flow and CO$_2$ curves in real data. Vertical bars correspond to zero-crossings. The zero-crossings **B**, **D** and **F** correspond to true breath ends. The zero-crossings **A**, **C** and **E** form false breath ends and need to be filtered out. In the top right corner, there is a zoom-in of part of the curve below. It shows the ratios of volumes of exhaled and inhaled air.

types of respiratory diseases including restrictive, obstructive and mixed ventilatory disorders. We were able to use it successfully in patients with variety of obstructions (cystic fibrosis, primary ciliary dyskinesia, obstructive bronchitis, etc.).

- **Robustness** – the algorithm is capable to detect breath ends even in severely distorted data, there is no need of strict adherence to the tidal breathing.
- **No false negatives** – the algorithm was able to detect all breaths that human experts detected.
- **Simplicity** – the algorithm is easy to describe and implement in software.
- **Generalizability** – the principles of our algorithm can be translated to other types of gases (oxygen, nitrogen, sulphur hexafluoride, . . .).
- **Grouping of zero crossings** – the algorithm groups together the zero-crossings corresponding to one breath.

To the best of our knowledge, there exist only two previously published algorithms using CO$_2$ concentration signal to detect breath ends. An algorithm presented by Brunner et al [23] was developed in the 1980s and was intended for patients from intensive care units. In contrast to our algorithm, it does not include calculation and comparison of tidal volume of the consecutive breaths to filter out false positive breaths. Moreover, their algorithm does not use grouping of zero-crossings. Their validation was performed only on healthy subjects with intentionally introduced artefacts. The validation in children and on severely distorted data was missing. They reported there was no apparent algorithm failure during its clinical use on 100 patients, however precise specification of the testing conditions are not transparent. They provide test results from only one patient (150 breaths in total). Govindajaran and Prakash [55] proposed an algorithm for breath detection during different modes of artificial ventilation (volume and pressure controlled, patient triggered modes). They used mainly the flow and airway pressure signals; CO_2 data were only an additional input to confirm a computed delineation of detected breaths. They did not report the accuracy of the algorithm and no validation was performed. Because the algorithm is designed for artificial ventilation, it is of limited applicability in lung function testing.

Besides the algorithms based on flow and gas concentration signal analysis, another approaches to breath detection were proposed. Recently, Nguyen C. D. et al developed a breath detection algorithm based on finding inflexion points in flow or epiglottic pressure signal [142]. The validation was performed in healthy individuals and in patients with sleep obstructive apnoea syndrome using continuous positive airway pressure therapy (CPAP). Their algorithm correctly identified 97.6% of reference breaths. They do not mention false positives. If we assume there are no false negatives it makes 2.4% false negative detections (for the sake of comparison, our algorithm returned no false negatives and only 0.5% false positives). Moreover, their approach needs pressure measurement during CPAP therapy and relies on tight face mask.

There is also an approach using neural networks [197] on respiratory volume data. They tested it on three young healthy volunteers and six healthy infants. Their algorithm shows similar or better results than other existing algorithms using volume information [27] and [220]. The accuracy of the algorithm was 98% of the reference number of breaths with 2% false negatives and 5% false positives.

Another approaches use body image processing techniques analyzing body position and movements [10], [213] and photopletysmographic approach [120]. Nevertheless, such algorithms are more suitable for monitoring of vital functions rather than for further clinical processing.

We acknowledge several limitations of our algorithm. Although it outperforms the currently existing algorithms in their accuracy, it still suffers from false breath detection on severely distorted data. This only proves the difficulty of the task of automated processing. Even two independent human experts might not agree on what is the proper breath identification for a given dataset. That explains the small chance of having this problem fully solved by a computer. Moreover, in our study we did not include a comparison of `Alg-OUR` to the breath detection algorithms based on neural networks or sound analysis. However, we primarily focused on lung function testing, which relies on flow and gas concentration signal. The other algorithms have their application in other fields of medicine (e.g., sleep medicine). There also exist various possibilities to extend our algorithm, which we did not investigate in greater detail. One of the next steps might be creating a database of documented patterns of breathing curve behavior and its combination with breath end detection.

9.5 Nitrogen concentration at peaks

After localization of the breath ends the imprecision of machine sensors must be incorporated. We used machine Exhalyzer D, that does not measure nitrogen concentration directly. It computes the nitrogen concentration (in %) according to the formula [101]

$$100 = \mathrm{N_2}\% + \mathrm{O_2}\% + \mathrm{CO_2}\% + \mathrm{Ar}\%,$$

where $\mathrm{Ar}\% = \mathrm{N_2}\% \times 0.0093/0.7881$ and where the concentrations of nitrogen, oxygen, carbon dioxide and argon in inspired and expired air are supposed to sum up to 100 %. The argon concentration is fixed. Together it gives

$$\mathrm{N_2}\% = \frac{1}{1.0118}(100 - \mathrm{O_2}\% - \mathrm{CO_2}\%),$$

where all parameters are in percents.

According to the manufacturer, the O_2 sensor has 0.3% accuracy and the CO_2 sensor has 5% accuracy. From that we can derive a interval bounds for the nitrogen concentration in each time slice $\boldsymbol{n_i}$

$$\begin{aligned}\underline{n}_i &= \frac{1}{1.0118}(100 - 1.003 * \mathrm{O_2}\% - 1.05 * \mathrm{CO_2}\%),\\ \overline{n}_i &= \frac{1}{1.0118}(100 - 0.997 * \mathrm{O_2}\% - 0.95 * \mathrm{CO_2}\%).\end{aligned}$$

We subtracted the minimal possible value from 100 to obtain upper bound and the maximal possible value from 100 to obtain lower bound. In the MBW procedure there are many sources of errors:

- Imprecision of sensors
- Changing viscosity and humidity of air
- Time shift of signals
- Interaction with deadspace air
- Physiological noise (heart pulse, hick-ups, leaks)
- Irregular breathing pattern, apnea
- Computer and machine rounding errors
- etc.

Unknown distributions and interplay of the mentioned uncertain variables will result in intervals with unknown distribution. Hence it is necessary to work with only lower and upper bounds. Here the interval analysis can be viewed as a tool for dealing with such uncertainties algebraically (using the means of interval linear algebra). We further view the data as interval data as depicted in Figure 9.6.

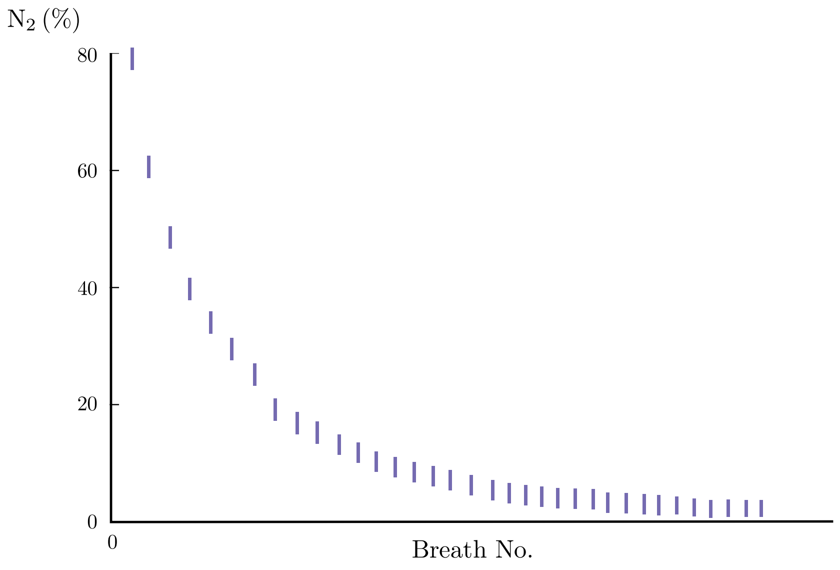


Figure 9.6: Illustration of decreasing concentration of nitrogen in peaks bounded with intervals.

9.6 Questions we asked

After long discussions we stated a few questions that are interesting from both clinical and mathematical point of view. The important and still discussed question is the behavior of the nitrogen washout in time. There is an observable difference between a healthy and diseased person, however the objective description is still missing. The long duration of washout (especially in severely affected patients) limits the feasibility of the test especially in small children (toddlers and pre-schoolers). Currently, the premature cessation of the washout (before reaching 2.5% of the starting nitrogen concentration) prevents us from analyzing the data. The possibility to derive substitute indices computable from an incomplete washout curve would be of great benefit.

9.7 Regression on interval data

Various authors approached the topic of regression on interval data, e.g, [24, 34, 77, 211]. Behind an interval regression or interval estimation the following general definition can be seen.

Definition 9.2. A result of a multi-linear interval regression on (interval) data tuples

$$(\boldsymbol{x}_1^i, \boldsymbol{x}_2^i, \ldots, \boldsymbol{x}_n^i, \boldsymbol{y^i}),$$

is generally

$$\boldsymbol{r}(x_1, x_2, \ldots, x_n) = \boldsymbol{p}_1 x_1 + \boldsymbol{p_2} x_2 + \cdots + \boldsymbol{p_n} x_n,$$

where $\boldsymbol{p} = (\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n)^T$ are interval parameters.

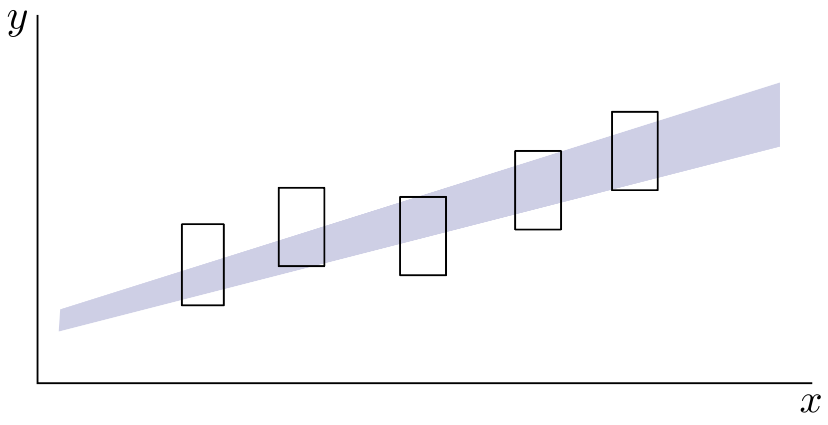


Figure 9.7: An example of $\boldsymbol{r} = \boldsymbol{p}_1 x + \boldsymbol{p}_2$. The band actually forms an interval line, which passes through each interval box.

The resulting $\boldsymbol{r}$ can be viewed as a multi-dimensional band. A two-dimensional example can be found in Figure 9.7.

As it was explained, there are various types of interval regression. They vary in computation of interval parameters $\boldsymbol{p}$. For example, $\boldsymbol{p}$ could be computed in such a way to force the band $\boldsymbol{r}$ to contain all the data tuples, or at least to cross all the interval data. For our purpose the interval least squares approach is the most meaningful.

Definition 9.3. For a given data: an $m \times n$ interval matrix $\boldsymbol{X}$, where its ith row is the tuple

$$(\boldsymbol{x}^i_1, \boldsymbol{x}^i_2, \ldots, \boldsymbol{x}^i_n),$$

and an m-dimensional column vector $\boldsymbol{y}$, where its coefficients are $\boldsymbol{y}^i$, the interval parameters $\boldsymbol{p}$ of the interval least squares estimation are defined in the following way,

$$\boldsymbol{p} = \square\{p : X^T X p = X^T y \text{ for some } X \in \boldsymbol{X}, y \in \boldsymbol{y}\}.$$

In Section 7.2.4 we addressed how to solve such a problem. We basically solved the following system

$$\begin{pmatrix} I & \boldsymbol{X} \\ \boldsymbol{X}^\top & 0 \end{pmatrix} \begin{pmatrix} p' \\ p \end{pmatrix} = \begin{pmatrix} \boldsymbol{y} \\ 0 \end{pmatrix} \tag{9.1}$$

using the means of some method for solving square interval systems from Chapter 5. The last n coefficients of the resulting enclosure give an enclosure on $\boldsymbol{p}$.

When we take a look at $(\boldsymbol{X}, \boldsymbol{y})$ data obtained by MBW procedure in Figure 9.6 we realize that

- $X = \boldsymbol{X}$ is thin, it consist of integers only (numbers of breaths) – we use such a form to avoid using intervals on the x-axis,
- intervals are only at the right-hand side $\boldsymbol{y}$,

- We want to use regression with nonlinear models that are linearizable, therefore $X^\top X$ is going to be small $n \times n$, $(n = 2, 3, 4)$, depending on the number of parameters of the model used (see the table 9.4 in advance),
- $X, \boldsymbol{y} > 0$ (component-wise).

Using these favorable properties, we hoped to design a method returning tighter enclosures than (9.1). Unfortunately, we were not able to find such a method. We believe that it is a really hard task since the mentioned properties are also in favor of (9.1). However, we were able to rewrite the formulas to obtain algorithms that are much faster.

9.7.1 Case 2×2

When the matrix X is of size $m \times 2$ (the left column is consists of ones, and the right one of numbers $1, \dots, m$), then X^TX is of size 2×2. We can apply the state of the art supersquare approach, however, in this case the "not recommended" approach of solving the interval normal equation $X^TXp = X^T\boldsymbol{y}$ will pay off. This actually means computing an enclosure of p as

$$\boldsymbol{p} = \left((X^TX)^{-1}X^T\right)\boldsymbol{b}. \tag{9.2}$$

When computing an inverse matrix, fractions can occur and therefore so can machine nonrepresentable numbers. That is why, we need to compute in a verified way with intervals. Nevertheless, it is advantageous to postpone the interval computation as far as possible, because the classical arithmetic is usually faster (e.g., in Octave or Matlab). In this case we use the simple shape of the 2×2 matrix inverse

$$(X^TX)^{-1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc}\begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

It is possible to compute X^TX in floating point arithmetics since X contains only integers; similarly for $ad - bc$.

When computing the expression $(X^TX)^{-1}X^T\boldsymbol{y}$, $\boldsymbol{y}$ is multiplied by an interval matrix, this unfortunately causes large growth of interval radii. And then it is multiplied again with the matrix $(X^TX)^{-1}$ which causes another growth. More suitable way is to rearrange the expression to multiply the integer parts (matrices) first and then multiplying with the interval elements. Thus, the enclosure of $\boldsymbol{p}$ can be computed as

$$(MX^T)(\boldsymbol{q}\boldsymbol{y}),$$

where

$$M = \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}, \ \boldsymbol{q} = \square\left(\frac{1}{ad - bc}\right).$$

We tested the difference between (9.1), which was solved by HBR method (`supsq`) and (9.2) solved directly by computing the verified inverse (`normal`), and the same procedure but with postponing the interval operations (`postponed`). The differences between approaches are clearly seen in the following example.

Table 9.2: Average computation times (in seconds) for 2×2 systems (Example 9.4) for the supersquare approach (`supsq`) and solving interval normal equations without (`normal`) and with postponing the interval operations (`postponed`).

m	`supsq`	`normal`	`postponed`
50	0.224	0.067	0.013
100	0.407	0.069	0.014
150	0.609	0.070	0.014
200	0.857	0.071	0.014
250	1.156	0.070	0.014

Example 9.4. The difference was tested on random systems for sizes up to $m = 250$, which represents the ceiling for the maximum number of breaths generally occurring during MBW testing. To generate a random right-hand side we first generated random intervals with centers from $[-10, 10]$ and fixed radii equal to 1 and then the intervals were placed along a random line and then shifted by a random number in $[-5, 5]$. The testing was done using LAPTOP setting. For each size we tested on 100 random systems. Both methods in all cases computed identical enclosures for $\boldsymbol{p}$. However, average computation times were different, they are displayed in the following Table 9.2.

9.7.2 Case 3×3 and larger

It would be more complicated to find similar inverse formula for a general square matrix. This time we refrain from postponing interval computations.

Example 9.5. We again compare with the supersquare approach. The test data were generated in a similar way to Example 9.4. The only difference is that the right-hand side intervals were placed along a parabola. The obtained enclosures of $\boldsymbol{p}$ are again identical and the average computation times are displayed in Table 9.3. The `normal` method is still faster than supersquare approach.

9.8 In search for a model

Inspired by Figure 9.6, the main goal is to derive the following function

$$f(n), \text{ for } n = 1, 2, \ldots$$

where n is the number of a peak (or breath), where the initial peak has number 1. The function f returns a nitrogen concentration at each peak n (it can be an interval concentration) and should plausibly model the nitrogen concentration at each peak. We call such a function f a *nitrogen washout curve model.* This goal was addressed earlier in [187] using a simplified model of lungs. They were not able to compute with

Table 9.3: Average computation times (in seconds) for 3×3 systems (Example 9.5) for the supersquare approach (`supsq`) and solving interval normal equations with (`normal`).

m	`supsq`	`normal`
50	0.26	0.07
100	0.42	0.07
150	0.67	0.07
200	0.92	0.07
250	1.19	0.07

models having more parameters due to the limited computational power (they handled many calculations manually). Their approach could be described as "bottom-up". A similar approach but for a different goal can be seen, e.g., in [212].

Our approach is slightly different, we could call it "top-down". Using a computer we explore the most frequent mathematical models of decay and test their ability to fit the measured medical data. Such a fitting might help to obtain more information about the real behavior of the nitrogen washout process and such knowledge will help to better predict the behavior of an incomplete measurement.

9.8.1 Center data

In the previous sections we showed how to derive interval data from a measured real patient data. To have at least rough idea about the behavior of the nitrogen washout process, classical least squares data fitting was applied on center data (for a while we consider only midpoints from all intervals).[1]

We are interested in fitting curves for which the process of good fitting can be transformed to solving a linear system of equations. The quality of fit was measured by rMSE which is the square root of MSE (mean squared error). We fit the data in least squares manner. If we evaluate the measurements visually, we could detect "exponential"-like decay in all data. An example could be seen in Figure 9.6. Many papers and books (also possibly the medical software shipped with the machine Exhalyzer D) describe this decay as an exponential function [32]. This is one of the classical fitting models. When talking about classical fitting models we tried to find the most suitable one among them. From the large collection of models [205] we selected the following model candidates fulfilling the visual criteria first. They are summarized in Table 9.4. For each model in the left column there is the abbreviation by which we address the model, in the second column there is the mathematical description of the model and in the third column there are the parameters that need to be computed to fit a given dataset with this model. As already mentioned, all of these models can be linearized. For a detailed description of this process for each model see [205].

[1]For this purpose we used a different data set to the one from Subsection 9.4.2. We selected cleaner data to make them more suitable for regression.

Table 9.4: Table of the fitting models used.

model	function $f(x)$	parameters
`exp`	$ae^{(bx)}$	a, b
`explin`	$a + bx + ce^x$	a, b, c
`pow`	ax^b	a, b
`exppow`	$ax^b c^x$	a, b, c
`log`	$a + b\log(x)$	a, b
`loglin`	$a + bx + c\log(x)$	a, b, c
`explin`	$a + bx + ce^x$	a, b, c
`explog`	$a + b\log(x) + ce^x$	a, b, c
`exploglin`	$a + bx + c\log(x) + de^x$	a, b, c, d

For each dataset (one measurement) each model was fitted and rMSE computed. As stated earlier, the 2.5% and 5% concentration level is significant for medical specialists. When we follow the nitrogen curve in time beyond the 2.5% level of concentration, it can be seen that the concentration peaks can be interpolated with a nearly horizontal line. It is difficult for all models to fit properly such slowly decreasing end. That is why we also measured the quality of fit to a level where something is “still happening” (the curve does not decrease so slowly) – up to 5%. The rMSE results can be seen in Tables 9.5, 9.6, 9.7 and 9.8 at the end of this chapter.

From the perspective of rMSE measure the model `loglin` is the winner. The rMSE penalizes heavily the large misfits. If we take a look at the `loglin` curve it can fit the initial part of the washout curve pretty well. All other models are penalized, except for the model `exploglin`. It sometimes seems to be better, however, the coefficient in exponential member of the formula (d) is usually an extremely tiny number ($\sim 10^{-10}$). That is why this model is usually the same as `loglin`. From the perspective of Occam’s principle we further consider only the `loglin` model. However, the curve with the best rMSE fit does not have to be necessary the best for the sake of prediction of the washout curve behaviour. Notice that the model `exp`, which is often used in describing the nitrogen washout curve in medical literature, is not so accurate.

When data sets were shortened up to the point where the nitrogen concentration decreases below 5% of its initial concentration, the model `exppow` works much better on this initial phase; and its fitting error improved. Nevertheless, the best fitting model is still `loglin`. We therefore have some candidates for interval fitting models. We omit the model `exploglin`, since it is too complicated. We exclude the model `log` since it is contained in `loglin` and does not have better results than `loglin`. We also cast out models `explin` and `explog` due to a large error rate. We have four remaining candidates – `exp`, `pow`, `exppow`, `loglin` – that we further use. None of the checked model curves was able to accurately fit the data from the 5% to 2.5%. The level of 5% therefore seems to be a meaningful level that still enables possible plausible fitting with one of the classical models. This could also be an important fact for current discussions about advantages of LCI5 over LCI2.5. However, we must be careful not to reach the conclusions too quickly, because the part of the washout curve between 5% and 2.5% can possibly contain some important information about the quality of patient airways. Tossing a terminal part of the data away might mean tossing away an important information for further medical analysis.

9.8.2 Interval models

We took the four candidates on fitting curves – `exp`, `pow`, `exppow`, `loglin` – and provided the interval fitting of each model in the least squares manner. Each fitting of a nonlinear model can be transformed to solving an interval linear system of equations (the process is thoroughly described in [24]) and then solved by the means described in Section 9.7. Unfortunately, the results were not encouraging – the resulting interval washout curve models are too wide to yield any insight on the process of nitrogen washout. Another reason for such an overestimation might be the fact that solving an interval linear system exactly is difficult and we produced only an overestimated enclosure. Also induced dependencies in the supersquare system may play an important role (see 7.2.4). Shapes typical for each interval washout model are depicted in Figure 9.8. The `exp` function misses the initial and terminal part of the washout data. The `pow` model misses the initial part. The `exppow` model is usually too wide, however, it contains the data inside the interval curve. The `loglin` model usually tends to widen in time; ruining any possibility of prediction. As shown in the next subsection, we blame the accuracy of the sensors. Hence our result, although negative, might be a serious contribution to the ongoing discussions on quality of sensors.

9.8.3 Hypothetical sensors

We showed that problem of quality of fitted interval models lies within precision of current sensors (0.3% for O_2 sensor and 5% for CO_2 sensor of Exhalyzer D machine) and also within the methods for solving interval systems of equations. One might claim that the main flaw lies in the methods for solving interval systems and their overestimation. To shed more light on this, let us assume we have sensors with a better accuracy by one order, i.e, 0.03% for O_2 sensor and 0.5% for CO_2 sensor.

Let us repeat the same procedure as in Figure 9.8, this time for the hypothetical sensors. The surprising results are displayed in Figure 9.9. We checked all the four mentioned models manually by visual evaluation. We omitted the model `pow`, because it gave poor fitting results in the initial parts. We also omitted the model `exp`. Although, it gave very narrow curves it resulted in a poor fit. We checked the two remaining models – `exppow` and `loglin`. The problems with `loglin` still persist. Even for narrow intervals the curve tends to rise at its end. This gives us the winning description model – `exppow`. If we take a look at Figure 9.9, we see that the behaviour of `exppow` model does not fit the data well under the horizontal line (5% concentration level). However, it seems to work well before it crosses the level . We further check its properties in the next section.

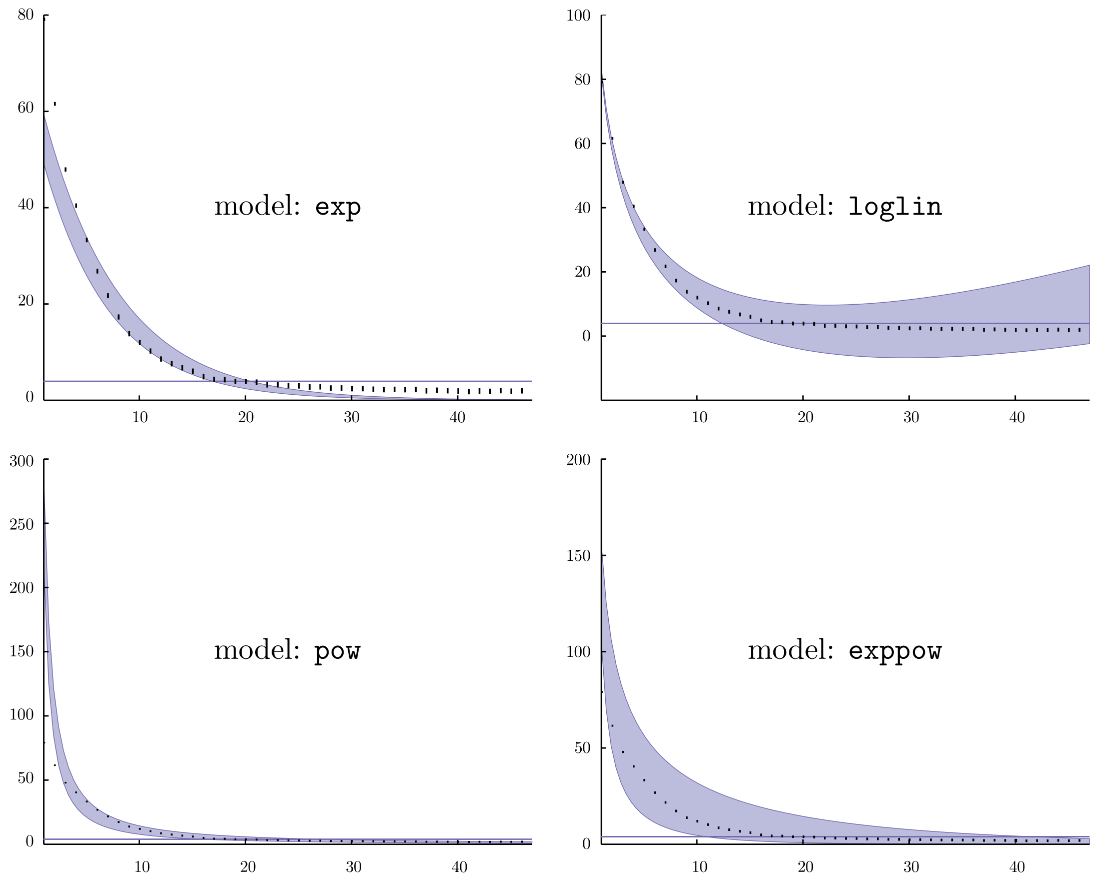


Figure 9.8: Interval curves fitting a real data with real measurement errors – typical behavior. The tiny rectangles represent the interval data. The horizontal line represents the level of 5% of the initial nitrogen concentration. Notice that the y-scale of each graph is different. The darker area corresponds to the interval least squares fitting curves (interval washout models).

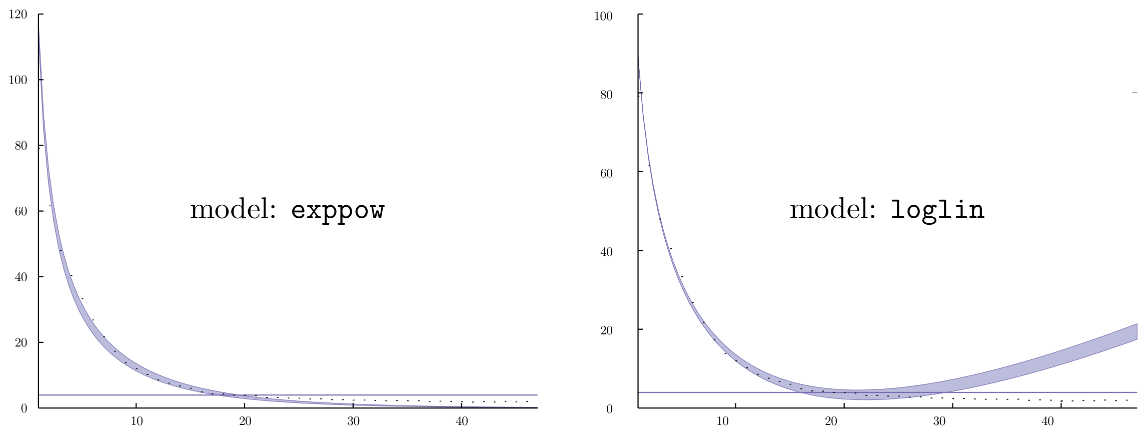


Figure 9.9: Interval curves fitting a real data with hypothetical measurement errors – typical behavior. The tiny rectangles represent the interval data. The horizontal line represents the level of 5% of the initial nitrogen concentration. Notice that the y-scale of each graph is different. The darker area corresponds to the interval least squares fitting curves (interval washout models).

9.8.4 Prediction

As it was said the level of nitrogen concentration where we stop the measurement is 2.5% or 5%. This boundary was set historically. For young uncooperative patients it might be difficult to prevent leaks and maintain calm and regular breathing for a longer period of time. Sometimes the measurement must be aborted. In order to not waste the so far good measurement we can try to predict the successive behavior of the washout curve. Using the previously developed interval washout models we focus on determination of the terminal breath of a measurement. To remind the definition, for a given level of nitrogen concentration (20%, 10%, 5% or 2.5%), the *terminal breath* for this concentration is defined to be the first one of the three consecutive breaths with concentration below the respective level.

We limited our prediction to the part of the washout curve between 10% and 5%. The goal was to predict an interval containing the terminal breath at 5% level and compare it with the real terminal breath at the corresponding level. For the prediction we used the hypothetical sensors only, the results are in Table 9.9 at the end of the chapter.

In the case of hypothetical sensors, the prediction is not generally bad. However, in some cases the prediction is completely wrong. We conclude that none of the tested models is completely suitable for absolutely correct prediction. Nevertheless, the quality of prediction brings us to the very important question we tackle more in the following subsection.

9.8.5 An alternative clinical index?

The prediction of the washout curve in current software (Spiroware) is of poor quality. We could see that the prediction using verified interval regression is also not too

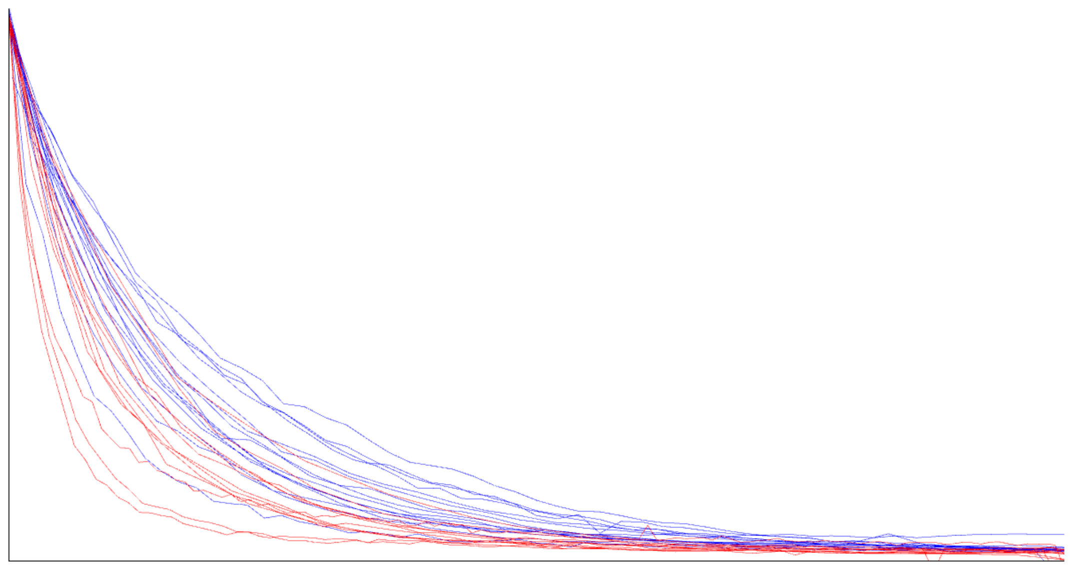


Figure 9.10: Washout curves (real data) of all patients normalized to the same length. The blue curves correspond to healthy persons, the red curves correspond to patients with cystic fibrosis.

trustworthy. The problem lies in an unsatisfactory model of the nitrogen washout process. We discussed many washout curve models, however none of them was plausible enough (for the purpose of prediction). Before starting to seek for better models, it needs to be specified, why exactly do we need predictions and models of washout process. One reason has been documented previously on an example of an interrupted measurement because of patient's weak cooperation. Indeed, the possibility to predict washout process would be of a great clinical value. Unfortunately, our results indicate, that predictions are not possible within the currently used approach to washout data analysis.

Let us say we want to predict LCI from an incomplete measurement. To derive the LCI, the FRC is also needed. For FRC derivation we need to compute V_{out} (as an integration of flow), therefore we need to know the missing flow data whose prediction is nearly impossible (too jagged shape of the flow curve). In conclusion, even if we had a good prediction of nitrogen washout behavior, there is no way to compute a meaningful LCI with this prediction.

With that a new question arises – can LCI be replaced by another index describing ventilation inhomogeneity and being more suitable to for prediction (and also robust enough to overcome some inaccuracy of prediction)? Our initial hypothesis was to use the information of curvature of the washout curve. However when the curves were normalized to stretch over the same time window, we obtained Figure 9.10. It shows that the patients cannot be simply separated as healthy or as having cystic fibrosis according to the curvature of the washout curve. Hence finding a new clinical index enabling prediction still remains a challenge.

9.9 Results relevant for medicine

We summarize the results that might be relevant to the ongoing medical discussions in the form of the following list:

- We demonstrated that the models that are usually used in literature for description of the behavior of the nitrogen washout process are not plausible.
- We showed that if we consider the classical fitting models, the best model (but still not ideal) for the washout curve description is `exppow`.
- Fitting the data with classical models up to 5% is much more achievable than the attempts to fit the data up to 2.5%.
- We gave an argument using interval analysis that current accuracy of Exhalyzer D sensors seems to be insufficient for interval data estimation and making reasonable predictions.
- If we had sensors with better accuracy just by one order the verified fitting would work.
- It is impossible to predict the future value of LCI based on an interrupted measurement due to properties of LCI.
- New clinical indices should be developed to suit prediction.
- Healthy persons and patients with cystic fibrosis cannot be simply distinguished by curvature of the washout curve.

In our work numerous ways of future research emerged – finding better models of the washout process, combination of the top-down and bottom-up approach in washout modeling, search for new clinical indices that will enable better prediction. It would be also interesting to combine the algebraic approach to uncertainty with the statistical one.

Table 9.5: Healthy persons – rMSE for fitting up to 2.5% of the initial nitrogen concentration.

No.	`exp`	`explin`	`pow`	`exppow`	`log`	`loglin`	`explin`	`explog`	`exploglin`	H/CF
1	7.15	23.39	28.44	6.33	4.29	**1.81**	23.39	22.54	3.50	H
2	7.98	22.22	24.45	6.59	4.64	**1.44**	22.22	21.56	2 4.59	H
3	10.67	11.84	11.30	8.11	6.26	**1.16**	11.84	5.70	6.98	H
4	10.07	11.67	11.11	5.44	6.32	**1.24**	11.67	5.75	0.96	H
5	10.80	11.82	8.80	5.94	6.63	**1.58**	11.82	6.12	1.32	H
6	8.86	10.85	16.38	5.79	4.82	**1.24**	10.85	4.68	0.77	H
7	10.71	19.15	5.91	9.44	5.71	**1.09**	19.15	19.15	19.15	H
8	7.00	10.40	21.37	4.63	4.25	**1.18**	10.40	3.83	1.18	H
9	7.10	10.60	20.71	3.81	4.43	**1.27**	10.60	4.05	1.26	H
10	3.44	23.75	31.45	2.35	2.67	**2.18**	23.75	23.75	23.75	H
11	4.27	25.36	35.95	3.04	3.03	**2.16**	25.36	25.36	25.36	H
12	3.61	24.58	33.17	2.14	2.58	**1.84**	24.58	24.58	24.58	H
13	2.74	26.09	36.08	**1.13**	2.29	1.87	26.09	26.09	26.09	H
14	5.30	10.11	24.22	2.44	3.82	**1.46**	10.11	3.40	1.45	H
15	10.30	16.14	6.34	7.50	5.96	**1.97**	16.14	16.14	16.14	H

Table 9.6: Patients with cystic fibrosis –rMSE for fitting up to 2.5% of the initial nitrogen concentration.

No.	`exp`	`explin`	`pow`	`exppow`	`log`	`loglin`	`explin`	`explog`	`exploglin`	H/CF
1	8.01	11.17	13.76	2.58	5.98	**1.34**	11.17	5.16	1.16	CF
2	9.54	12.69	**2.97**	4.05	6.23	3.26	12.69	12.45	12.45	CF
3	10.32	33.33	**2.71**	7.47	6.46	2.91	33.33	13.55	33.33	CF
4	9.08	11.69	11.43	3.85	7.00	**1.32**	11.69	5.78	0.81	CF
5	10.92	18.09	8.15	7.76	6.17	**2.20**	18.09	18.09	18.09	CF
6	8.31	14.40	5.36	3.38	4.69	**1.90**	14.40	14.40	14.40	CF
7	9.21	11.61	10.54	3.90	6.74	**1.27**	11.61	5.67	0.79	CF
8	7.55	11.04	22.42	5.86	4.89	**1.66**	11.04	4.35	1.66	CF
9	9.85	11.69	16.16	7.68	5.64	**1.00**	11.69	17.42	10.94	CF
10	7.38	10.78	23.83	5.41	4.53	**1.03**	10.78	4.07	1.03	CF
11	5.30	10.13	22.82	3.24	4.14	**1.35**	10.13	3.49	1.35	CF
12	7.40	10.88	19.37	4.57	5.04	**1.14**	10.88	4.43	1.11	CF

Table 9.7: Healthy persons – rMSE for fitting up to 5% of the initial nitrogen concentration.

No.	`exp`	`explin`	`pow`	`exppow`	`log`	`loglin`	`explin`	`explog`	`exploglin`	H/CF
1	1.99	7.80	20.35	**0.92**	2.01	1.85	7.80	1.72	1.69	H
2	3.46	8.45	18.63	1.84	2.42	**1.70**	8.45	1.96	1.59	H
3	4.51	8.83	14.25	2.22	3.57	**1.04**	8.83	2.51	0.98	H
4	5.67	9.27	11.05	1.83	4.21	**0.48**	9.27	3.18	0.46	H
5	6.75	9.88	10.43	2.42	4.74	**0.60**	9.88	3.74	0.52	H
6	4.95	8.95	14.95	1.96	3.02	**0.93**	8.95	2.44	0.90	H
7	10.71	19.15	5.91	9.44	5.71	**1.09**	19.15	19.15	19.15	H
8	4.06	8.66	17.30	1.88	2.66	**1.35**	8.66	2.14	1.29	H
9	4.96	9.14	15.59	2.18	3.12	**1.52**	9.14	2.65	1.50	H
10	2.32	25.03	22.83	**1.53**	2.18	2.16	25.03	26.78	25.01	H
11	1.96	7.67	22.55	**0.78**	2.02	2.02	7.67	1.93	1.91	H
12	2.17	7.20	20.63	**1.07**	1.63	1.63	7.20	1.53	1.51	H
13	1.84	7.69	21.54	**0.63**	1.61	1.58	7.69	22.16	10.80	H
14	4.03	8.71	16.24	1.77	2.75	**1.68**	8.71	2.36	1.67	H
15	7.31	9.88	11.26	2.70	4.09	**0.86**	9.88	3.65	0.83	H

Table 9.8: Patients with cystic fibrosis – rMSE for fitting up to 5% of the initial nitrogen concentration.

no.	`exp`	`explin`	`pow`	`exppow`	`log`	`loglin`	`explin`	`explog`	`exploglin`	H/CF
1	5.59	9.65	11.00	**0.94**	4.78	1.15	9.65	3.83	0.98	CF
2	8.52	10.41	8.64	3.58	4.82	**0.75**	10.41	4.32	0.64	CF
3	7.50	10.06	10.10	2.88	4.39	**0.68**	10.06	3.85	0.59	CF
4	4.18	8.15	9.97	1.22	4.75	**0.57**	8.15	2.93	0.52	2CF
5	7.39	10.14	12.08	3.67	4.29	**0.88**	10.14	3.69	0.88	CF
6	8.29	19.53	6.22	2.67	4.22	**1.17**	19.53	19.53	19.53	CF
7	6.01	9.44	10.36	2.25	5.31	**0.67**	9.44	3.71	0.59	CF
8	3.66	8.66	18.49	2.69	3.05	**1.93**	8.66	2.32	1.79	CF
9	4.84	9.07	14.72	1.57	3.15	**0.93**	9.07	2.54	0.92	CF
10	3.08	8.26	16.64	**0.74**	2.22	1.19	8.26	1.76	1.14	CF
11	2.72	8.17	17.33	**1.20**	2.66	1.48	8.17	1.97	1.38	CF
12	3.73	8.45	15.46	1.89	3.10	**1.32**	8.45	2.22	1.24	CF

Table 9.9: Prediction from 10% to 5% – hypothetical sensors; the intervals are predictions of the terminal breath number by various interval models, len – number of total breaths in file, real – number of real breath end at 5% level, H – healthy person, CF – patient with cystic fibrosis. Prediction intervals $[\underline{a}, \overline{a}]$ containing the true value of breath end having $|\overline{a} - \underline{a}| \leq 2$ are depicted in boldface.

No.	len	real	`exp`	`pow`	`exppow`	`loglin`	H/CF
1	49	23	[22, 22]	[49, 49]	**[22, 23]**	[18, 19]	H
2	39	23	[20, 20]	[39, 39]	[20, 21]	[17, 18]	H
3	25	14	[12, 12]	[22, 22]	[12, 13]	[11, 12]	H
4	23	13	[11, 11]	[20, 20]	[12, 12]	[11, 23]	H
5	25	14	[11, 11]	[19, 20]	[12, 12]	[12, 25]	H
6	35	21	[17, 18]	[35, 35]	[18, 19]	[16, 17]	H
7	51	51	[21, 21]	**[49, 51]**	[22, 23]	[19, 22]	H
8	32	22	[19, 19]	[32, 32]	[19, 20]	[16, 17]	H
9	32	22	[19, 19]	[32, 32]	[20, 21]	[17, 19]	H
10	51	40	[35, 35]	[51, 51]	[35, 36]	[27, 29]	H
11	51	35	[33, 34]	[51, 51]	**[34, 35]**	[27, 28]	H
12	50	34	[32, 32]	[50, 50]	[32, 33]	[26, 28]	H
13	51	37	**[36, 37]**	[51, 51]	**[37, 38]**	[30, 31]	H
14	36	21	[19, 19]	[36, 36]	**[20, 21]**	[17, 18]	H
15	63	26	[19, 19]	[37, 38]	[21, 22]	[22, 63]	H
1	28	12	[11, 11]	[17, 17]	**[11, 12]**	[28, 28]	CF
2	98	24	[16, 16]	[31, 32]	[17, 18]	[98, 98]	CF
3	80	21	[16, 16]	[30, 30]	[17, 18]	[80, 80]	CF
4	20	8	**[8, 8]**	[12, 12]	**[8, 8]**	[20, 20]	CF
5	48	22	[16, 16]	[31, 32]	[17, 18]	[15, 18]	CF
6	115	61	[37, 37]	[85, 89]	[45, 49]	[115, 115]	CF
7	23	10	[8, 8]	[12, 13]	[9, 9]	[23, 23]	CF
8	32	18	[15, 15]	[32, 32]	[15, 16]	[13, 13]	CF
9	40	19	[16, 17]	[34, 35]	[17, 18]	[15, 17]	CF
10	44	19	[18, 18]	[38, 39]	**[19, 20]**	[16, 18]	CF
11	26	16	[15, 15]	[26, 26]	[15, 15]	[13, 14]	CF
12	31	15	[13, 13]	[24, 25]	[13, 14]	[12, 13]	CF

10 A linear approach to CSP

- ▶ Linear relaxation
- ▶ Linear programming approach
- ▶ Vertex selection for relaxation
- ▶ Inner point selection for relaxation
- ▶ Properties of the obtained relaxation

In this chapter we introduce one particular approach to solving constraint satisfaction problems over interval boxes. We extend and generalize the work [8] by Araya, Trombettoni and Neveu. We introduce their concept of linear relaxation of a constraint satisfaction problem over a box, which results in a system of real inequalities. The box is then contracted with use of linear programming. To perform the linearization they need to select a vertex point (or a couple of them) of the box. We show that it is possible to select not only vertex points but also any point contained in the contracted box. We show some difficult examples for contractors and consistency techniques, that can be further improved by using the inner point choice. We prove that the proposed linearization is always at least as tight as Jaulin's linearization using two parallel affine functions [97, 99]. The whole chapter is a slightly reworked version of our paper [80]. The aim of this chapter is not to discuss the topic of nonlinear systems at large detail. There are many interesting books and works devoted to this topic, we will mention some of them at the end of this chapter.

10.1 The aim

In this chapter we deal with the *constraint satisfaction problem* (CSP). More specifically, we have a set of equality and inequality constraints

$$f_i(x) = 0, \quad i = 1, \ldots, k, \tag{10.1}$$
$$g_j(x) \le 0, \quad j = 1, \ldots, l, \tag{10.2}$$

where $f_i, g_j : \mathbb{R}^n \mapsto \mathbb{R}$ are real-valued functions. In compact form, it can be rewritten as

$$f(x) = 0,$$
$$g(x) \le 0,$$

where $f(x) = (f_1(x), \dots, f_k(x))$ and $g(x) = (g_1(x), \dots, g_l(x))$. In global optimization we additionally have a function $\varphi(x)$ and search for the global minimum of the function $\varphi(x)$ subject to these constraints. Such a problem can be transformed to the constraint satisfaction problem (see the next section).

We start with some initial intervals bounding the values of variables $x_1, \dots, x_n$. The bounding intervals $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_n)$ actually form an n-dimensional initial box $\boldsymbol{x}_1 \times \dots \times \boldsymbol{x}_n$, where we begin the search for solution (or minimum/maximum of $\varphi(x)$).

A common approach is to linearize nonlinear equalities and inequalities first. Such a procedure is called *linear relaxation.* Linear relaxations were also studied in, e.g., [6, 8, 26, 118, 217].

After linear relaxation a system of interval linear inequalities is obtained and linear programming can be used. The result is a box containing the solution that will be hopefully tighter than the initial one. If the box gets tighter, we can iterate this procedure. If the box cannot be tightened, we combine this technique with a branch and bound approach – the current box is split into halves and the procedure is repeated for both parts separately. We can recursively go on with splitting until the size of the box is small enough.

10.2 Global optimization as CSP

The problem of global optimization can be transformed to a constraint satisfaction problem and hence the previously mentioned techniques can be used. Let us have a global optimization problem

$$\min \ \varphi(x),$$

$$f_1(x) = 0, \dots, f_k(x) = 0, \tag{10.3}$$

$$g_1(x) \le 0, \dots, g_l(x) \le 0, \tag{10.4}$$

and an initial box $\boldsymbol{x}$. We would like to get a rigorous bounds for $\min \varphi(x)$ for $x \in \boldsymbol{x}$. First by solving the CSP problem defined by (10.3) and (10.4) we get some box $\boldsymbol{x}^*$ where the solution is located. Then we evaluate the $\varphi(x)$ on this box and take the minimum $\underline{\varphi}(\boldsymbol{x}^*)$, this provides a safe lower bound for the global minimum. To obtain an upper bound on the global minimum, we can take any feasible solution x' from $\boldsymbol{x}^*$ and its value $\varphi(x')$. A feasible solution can be found, for example, by local search techniques. As in the previous section, this approach can be combined with a branch and bound approach. That is why in the rest of the chapter we are going to deal with the constraint satisfaction problem only.

10.3 Interval linear programming approach

Our approach is based on linearization of constraints (10.1) and (10.2) by means of interval linear equations and inequalities. Then by using interval linear programming techniques [68] we construct a polyhedral enclosure to the solution set of (10.1) and (10.2) and contract the initial box $\boldsymbol{x}$. The process can be iterated, resulting in a nested sequence of boxes enclosing the solution set.

Let us have a function $h : \mathbb{R}^n \mapsto \mathbb{R}$ and some interval vector $\boldsymbol{x} \in \mathbb{IR}^n$. Then

$$h(\boldsymbol{x}) = \{h(x) \mid x \in \boldsymbol{x}\}.$$

However, for some more complex functions this is can hardly be computed. We usually compute some enclosure of $h(\boldsymbol{x})$.

First let us choose some point $x^0 \in \boldsymbol{x}$ which will be called a *center of linearization.* Suppose that a vector function $h : \mathbb{R}^n \mapsto \mathbb{R}$ can be enclosed by a linear enclosure

$$h(\boldsymbol{x}) \subseteq S_h(\boldsymbol{x}, x^0)(x - x^0) + h(x^0), \quad \text{for } \forall x \in \boldsymbol{x}, \tag{10.5}$$

for suitable interval-valued function $S_h : \mathbb{IR}^n \times \mathbb{R}^n \mapsto \mathbb{IR}^n$ This is usually calculated by the mean value form as explained in Chapter 3 or [139].

For more efficiency, successive mean value approach ([8]) or slopes ([59, 139]) can be employed. Alternatively, in some situations, a relaxation can be established by analyzing the structure of $h(x)$ – for example, quadratic terms can be relaxed as shown in [118]. After applying such a linearization to all functions $f_1, \ldots, f_k$ and $g_1, \ldots, g_l$ we obtain an interval linear system of equations and inequalities:

$$\begin{aligned} S_f(\boldsymbol{x}, x^0)(x - x^0) + f(x^0) &= 0, &(10.6)\\ S_g(\boldsymbol{x}, x^0)(x - x^0) + g(x^0) &\leq 0. &(10.7)\end{aligned}$$

We can briefly denote it as

$$\begin{aligned} \boldsymbol{A}(x - x^0) &= -f(x^0), &(10.8)\\ \boldsymbol{B}(x - x^0) &\leq -g(x^0). &(10.9)\end{aligned}$$

Theoretically, we do not need to choose the same x^0 for f's and g's. However, we choose the same x^0 for both of them. As the linearization depends on $x^0 \in \boldsymbol{x}$, the question is how to choose x^0.

10.4 Selecting vertices

First, let us take a look at the system (10.8). Using the Oettli–Prager theorem (Theorem 5.4) we can rewrite

$$\boldsymbol{A}(x - x^0) = -f(x^0),$$

as

$$|A^c(x - x^0) + f(x^0)| \leq A^\Delta |x - x^0|.$$

Note that $f(x^0)$ is actually a real number. Now we proceed as in Section 5.2. We can get rid of the first absolute value by rewriting it into the two cases:

$$\begin{aligned} A^c(x - x^0) + f(x^0) &\leq A^\Delta |x - x^0|,\\ -A^c(x - x^0) - f(x^0) &\leq A^\Delta |x - x^0|.\end{aligned}$$

We can get rid of the second absolute value by using knowledge of the sign of each coefficient of the vector in absolute value

$$A^c(x - x^0) + f(x^0) \le A^\Delta D_{\mathrm{sign}(x-x^0)}(x - x^0),$$
$$-A^c(x - x^0) - f(x^0) \le A^\Delta D_{\mathrm{sign}(x-x^0)}(x - x^0).$$

Now selection of x^0 should imply knowledge of $\mathrm{sign}(x - x^0)$. The very first idea that can come to our mind is to take x^0 as some corner of the initial box $\boldsymbol{x}$ [8]. If we take, for example, $x^0 = \underline{x}$, we immediately know that $(x - x^0)$ is nonnegative and get the linearization

$$\underline{A}x \le \underline{A}\underline{x} - f(\underline{x}), \quad \overline{A}x \ge \overline{A}\underline{x} - f(\underline{x}).$$

A similar technique can be applied to the system of inequalities (10.9). We can use the following Gerlach's characterization of all solutions to $\boldsymbol{A}x \le \boldsymbol{b}$ [53] (cf. [40, 70]).

Theorem 10.1 (Gerlach)**.** *A vector x is a solution of $\boldsymbol{A}x \le \boldsymbol{b}$ if and only if it satisfies*

$$A^c x - A^\Delta |x| \le \overline{b}.$$

By applying the theorem to (10.9) we obtain

$$B^c(x - x^0) \le B^\Delta |x - x^0| - g(x^0).$$

And using the same trick as before we rewrite the absolute value as

$$B^c(x - x^0) \le B^\Delta D_{\mathrm{sign}(x-x^0)}(x - x^0) - g(x^0).$$

Again, if we set, for example, $x^0 = \underline{x}$, we get the linearization

$$\underline{B}x \le \underline{B}\underline{x} - g(\underline{x}).$$

The question is, which corner to choose? In [8] it was proved that choosing the corner that gives the tightest linearization is an NP-hard problem. Even if the best corner for linearization was known, it would not guarantee significant contraction gain. However, this gives an insight, how difficult the problem is. Therefore, some heuristics need to be used. According to numerical tests in [8], they propose choosing two opposite corners of $\boldsymbol{x}$ and gathering linear inequalities from both linearizations as the input to a linear program. Which pair of opposite corners is the best choice is an open problem, a random selection seems to do well.

10.5 New possibility: selecting an inner point

Now we are able to linearize according to any corner of the initial box $\boldsymbol{x}$. What about the other points $x^0 \in \boldsymbol{x}$? In the following part we show that also an inner point can be used. Thus we provide an extension of [8].

Once again, for any x^0 the solution set to (10.8) and (10.9) is described by

$$\begin{aligned} |A_c(x - x^0) + f(x^0)| &\leq A_\Delta |x - x^0|, \\ B_c(x - x^0) &\leq B_\Delta |x - x^0| - g(x^0), \end{aligned}$$

which is a nonlinear system due to the absolute values. Fortunately, we can bound them using a theorem by Beaumont [14].

Theorem 10.2 (Beaumont). *Let $\boldsymbol{y} \in \mathbb{IR}$, then for every $y \in \boldsymbol{y}$*

$$|y| \leq \alpha y + \beta,$$

where

$$\alpha = \frac{|\overline{y}| - |\underline{y}|}{\overline{y} - \underline{y}}, \quad \beta = \frac{\overline{y}|\underline{y}| - \underline{y}|\overline{y}|}{\overline{y} - \underline{y}}.$$

Moreover, if $\underline{y} \geq 0$ or $\overline{y} \leq 0$ then the equality holds.

Now, the following proposition can be proved.

Proposition 10.3 (Hladík, Horáček [80]). *The linearization of (10.8) and (10.9) for an arbitrary $x^0 \in \boldsymbol{x}$ is*

$$(A^c - A^\Delta D_\alpha)x \leq A^c x^0 + A^\Delta v - f(x^0), \tag{10.10}$$

$$(-A^c - A^\Delta D_\alpha)x \leq -A^c x^0 + A^\Delta v + f(x^0), \tag{10.11}$$

$$(B^c - B^\Delta D_\alpha)x \leq B^c x^0 + B^\Delta v - g(x^0), \tag{10.12}$$

where α and v are vectors with coefficients

$$\begin{aligned} \alpha_i =& \frac{1}{x_i^\Delta}(x_i^c - x_i^0), \\ v_i =& \frac{1}{x_i^\Delta}(x_i^c x_i^0 - \overline{x}_i \underline{x}_i). \end{aligned}$$

Proof. First we show the relaxation for (10.9). Using Theorem 10.2

$$B_c(x - x^0) \leq B_\Delta |x - x^0| - g(x^0) \leq B_\Delta(D_\alpha(x - x^0) + \beta) - g(x^0),$$

where

$$\begin{aligned} \alpha_i &= \frac{1}{2x_i^\Delta}\left(|\overline{x}_i - x_i^0| - |\underline{x}_i - x_i^0|\right) = \frac{1}{2x_i^\Delta}\left(\overline{x}_i - x_i^0 - (x_i^0 - \underline{x}_i)\right) = \\ &= \frac{1}{x_i^\Delta}\left(\frac{(\overline{x}_i - \underline{x}_i)}{2} - x_i^0\right) = \frac{1}{x_i^\Delta}\left(x_i^c - x_i^0\right), \end{aligned}$$

$$\begin{aligned}\beta_i &= \frac{1}{2x_i^\Delta}\Big((\overline{x}_i - x_i^0)|\underline{x}_i - x_i^0| - (\underline{x}_i - x_i^0)|\overline{x}_i - x_i^0|\Big) = \\ &= \frac{1}{2x_i^\Delta}\Big((\overline{x}_i - x_i^0)(x_i^0 - \underline{x}_i) - (\underline{x}_i - x_i^0)(\overline{x}_i - x_i^0)\Big) = \\ &= \frac{1}{x_i^\Delta}(\overline{x}_i - x_i^0)(x_i^0 - \underline{x}_i).\end{aligned}$$

The inequality then takes the form

$$(B^c - B^\Delta D_\alpha)x \le B^c x^0 + B^\Delta(-D_\alpha x^0 + \beta) - g(x^0).$$

Herein,

$$\begin{aligned}(-D_\alpha x^0 + \beta)_i = -\alpha_i x_i^0 + \beta_i &= \frac{1}{x_i^\Delta}\Big(-(x_i^c - x_i^0)x_i^0 + (\overline{x}_i - x_i^0)(x_i^0 - \underline{x}_i)\Big) = \\ &= \frac{1}{x_i^\Delta}(-x_i^c x_i^0 + \overline{x}_i x_i^0 - \overline{x}_i\underline{x}_i + x_i^0\underline{x}_i) = \\ &= \frac{1}{x_i^\Delta}(-x_i^c x_i^0 - \overline{x}_i\underline{x}_i + \overline{x}_i x_i^0 + \underline{x}_i x_i^0) = \\ &= \frac{1}{x_i^\Delta}(-x_i^c x_i^0 - \overline{x}_i\underline{x}_i + 2x_i^c x_i^0) = \\ &= \frac{1}{x_i^\Delta}(x_i^c x_i^0 - \overline{x}_i\underline{x}_i) = v_i.\end{aligned}$$

Regarding (10.8) it is relaxed by Theorem 10.2 as

$$|A^c(x - x^0) + f(x^0)| \le A^\Delta|x - x^0| \le A^\Delta(D_\alpha(x - x^0) + \beta)),$$

which is just rewritten as the two cases

$$\begin{aligned}(A^c - A^\Delta D_\alpha)x &\le A^c x^0 + A^\Delta(-D_\alpha x^0 + \beta) - f(x^0),\\ (-A^c - A^\Delta D_\alpha)x &\le -A^c x^0 + A^\Delta(-D_\alpha x^0 + \beta) + f(x^0).\end{aligned}$$

$\square$

The Proposition 10.3 enables us to linearize according to any point from the initial box.

10.6 Two parallel affine functions

In [97, 99] Jaulin proposed a linearization using two parallel affine functions as a simple but efficient technique for enclosing nonlinear functions. In what follows, we show that for the purpose of polyhedral enclosure of a solution set of nonlinear systems, our approach is never worse than Jaulin's linearization estimate.

In accordance with (10.5) let us assume that a vector function $h : \mathbb{R}^n \mapsto \mathbb{R}^s$ has the following interval linear enclosure:

$$h(x) \subseteq \boldsymbol{A}(x - x^0) + b, \quad \forall x \in \boldsymbol{x}, \qquad (10.13)$$

for a suitable matrix $\boldsymbol{A} \in \mathbb{IR}^{n\times s}$ and $x^0 \in \boldsymbol{x}$, where $b = h(x^0)$. Using subdistributivity, for $A \in \boldsymbol{A}$ (see Section 3.7) we get

$$\boldsymbol{A}(x - x^0) + b \subseteq A(x - x^0) + b + (\boldsymbol{A} - A)(x - x^0).$$

Bounding the formula on the right-hand side from above and from below by two parallel affine functions gives

$$h(x) \le A(x - x^0) + b + \overline{(\boldsymbol{A} - A)(\boldsymbol{x} - x^0)},$$
$$h(x) \ge A(x - x^0) + b + \underline{(\boldsymbol{A} - A)(\boldsymbol{x} - x^0)}.$$

For $A = A_c, x^0 = x^c$ we particularly get

$$h(x) \le A(x - x^c) + b + A^\Delta x^\Delta,$$
$$h(x) \ge A(x - x^c) + b - A^\Delta x^\Delta.$$

Theorem 10.4 (Hladík, Horáček [80]). *For any selection of $x^0 \in \boldsymbol{x}$ and $A \in \boldsymbol{A}$, the linearization using the Beaumont theorem yields at least as tight enclosures as Jaulin's linearization using two parallel affine functions.*

Proof. We are going to prove the theorem for the estimation from above, the proof for the estimation from below can be done similarly. Using properties (3.1)–(3.5) the function $h(x)$ from (10.13) can be for $x \in \boldsymbol{x}$ bounded from above by

$$h(x) \le A^c(x - x^0) + A^\Delta|x - x^0| + b.$$

(This includes the vertex selection of x^0, too.) Then, the absolute value $|x - x^0|$ is linearized by means of Beaumont's theorem to

$$|x - x^0| \le D_\alpha(x - x^0) + \beta,$$

for some $\alpha, \beta \in \mathbb{R}^n$. The goal is to show that the interval linear programming upper bound

$$h(x) \le A^c(x - x^0) + A^\Delta(D_\alpha(x - x^0) + \beta) + b$$

is included in estimation using two parallel affine functions, that is

$$A^c(x - x^0) + A^\Delta(D_\alpha(x - x^0) + \beta) + b \in A(x - x^0) + (\boldsymbol{A} - A)(\boldsymbol{x} - x^0) + b,$$

or equivalently,

$$(A^c - A)(x - x^0) + A^\Delta(D_\alpha(x - x^0) + \beta) \in (\boldsymbol{A} - A)(\boldsymbol{x} - x^0),$$

The ith row of this inclusion reads

$$\sum_{j=1}^{n}(a^c_{ij} - a_{ij})(x_j - x^0_j) + \sum_{j=1}^{n} a^\Delta_{ij}(\alpha_j(x_j - x^0_j) + \beta_j) \in \sum_{j=1}^{n}(\boldsymbol{a}_{ij} - a_{ij})(\boldsymbol{x}_j - x^0_j).$$

We prove a stronger statement claiming that for any i, j it holds that

$$(a^c_{ij} - a_{ij})(x_j - x^0_j) + a^\Delta_{ij}(\alpha_j(x_j - x^0_j) + \beta_j) \in (\boldsymbol{a}_{ij} - a_{ij})(\boldsymbol{x}_j - x^0_j).$$

By substituting for α_j and β_j the left-hand side draws

$$(a^c_{ij} - a_{ij})(x_j - x^0_j) + a^\Delta_{ij}\Bigg(\frac{|\overline{x}_j - x^0_j| - |\underline{x}_j - x^0_j|}{2x^\Delta_j}(x_j - x^0_j) + \\ + \frac{(\overline{x}_j - x^0_j)|\underline{x}_j - x^0_j| - (\underline{x}_j - x^0_j)|\overline{x}_j - x^0_j|}{2x^\Delta_j}\Bigg). \tag{10.14}$$

This is a linear function in x_j, hence it is sufficient to show inclusion only for both endpoints of $\boldsymbol{x}_j$. By putting $x_j = \overline{x}_j$ the formula (10.14) simplifies to

$$(a^c_{ij} - a_{ij})(\overline{x}_j - x^0_j) + a^\Delta_{ij}|\overline{x}_j - x^0_j| \\ \in (\boldsymbol{a}_{ij} - a_{ij})(\overline{x}_j - x^0_j) \subseteq (\boldsymbol{a}_{ij} - a_{ij})(\boldsymbol{x}_j - x^0_j).$$

For $x_j = \underline{x}_j$ the proof is done analogously. □

10.7 Combination of centers of linearization

To obtain as tight polyhedral enclosure as possible it is convenient to simultaneously consider several centers for linearization. If we have no extra information, we recommend to relax according to two opposite corners of $\boldsymbol{x}$ (in agreement with [8]) and according to the midpoint $x^0 = x^c$. Putting all resulting inequalities together, we obtain a system of $3(2k + l)$ inequalities with respect to n variables. This system represents a convex polyhedron $\mathcal{P}$ and its intersection with $\boldsymbol{x}$ gives a new, hopefully tighter, enclosure of the solution set. Illustration of potential advantages of this process can be found in Figure 10.1.

When we calculate minima and maxima in each coordinate by calling linear programming, we get a new box $\boldsymbol{x}' \subseteq \boldsymbol{x}$. Rigorous bounds on the optimal values in linear programming problems were discussed in [95, 141]. The optimal values of the linear programs are attained in at most $2n$ vertices of $\mathcal{P}$, which lie on the boundary of $\boldsymbol{x}'$. It is tempting to use some of these points as a center x^0 for the linearization process in the next iteration. Some numerical experiments have to be carried out to show how effective this idea is. Another possibility is to linearize according to these points in the current iteration and append the resulting inequalities to the description of $\mathcal{P}$. By re-optimizing the linear programs we hopefully get a tighter enclosing box $\boldsymbol{x}'$. Notice that the re-optimizing can be implemented in a cheap way. If we employ the dual simplex method to solve the linear programs and use the previous optimal solutions as starting points, then the appending of new constraints is done easily and the new optimum is found in a few steps. We append only the constraints corresponding to the current optimal solution. Thus, for each of that $2n$ linear programs, we append after its termination a system of $(2k + l)$ inequalities and re-optimize.

In global optimization, a lower bound of $\phi(x)$ on $\mathcal{P}$ is computed, which updates the lower bound on the optimal value if lying in $\boldsymbol{x}$. Let x^* be a point of $\mathcal{P}$ in which

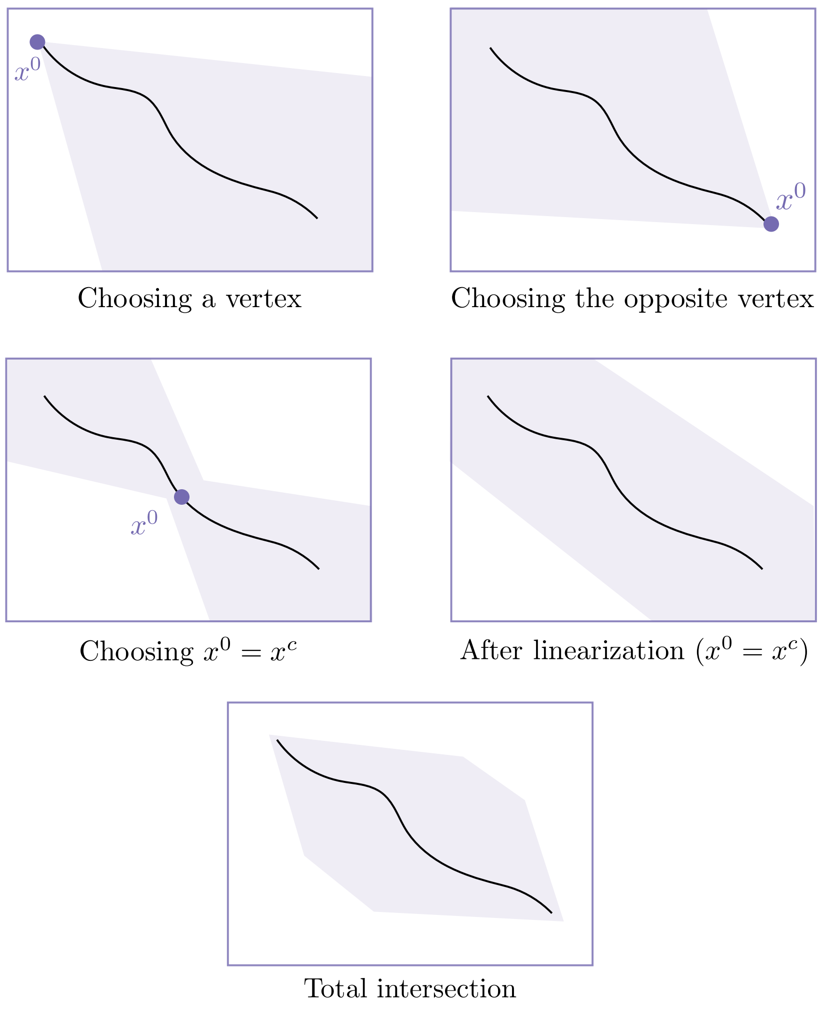


Figure 10.1: Illustration of relaxations obtained by selecting different centers of linearizations. The darker area is a linearized enclosure. The curve represents a set described by the constraints (10.1), (10.2)

.

the lower bound of $\phi(x)$ on $\mathcal{P}$ is attained. Then it is promising to use x^* as a center for linearization in the next iteration. Depending on a specific method for bounding of $\phi(x)$ from below, it may be desirable to append to $\mathcal{P}$ the inequalities (10.10)–(10.12) arising from $x^0 = x^*$, and to re-compute the lower bound of $\phi(x)$ on the updated polyhedron.

10.8 Convex case

If the constraint functions are of certain shape, then there is no need to use relaxation according to inner point, it is enough to linearize according to certain vertices (at most $n+1$) of $\boldsymbol{x}$. In the proposition below, an inequality is called a consequence of a set of inequalities if it can be expressed as a nonnegative linear combination of these inequalities. In other words, it is a redundant constraint if added to the set of inequalities.

Proposition 10.5 (Hladík, Horáček [80]). *Let $x^0 \in \boldsymbol{x}$ be a nonvertex point of $\boldsymbol{x}$. Suppose that $\boldsymbol{A}$ and $\boldsymbol{B}$ do not depend on a selection of x^0 .*

1. *If $f_i(x), i = 1, \dots, k$ are convex, then the inequality (10.10) is a consequence of the corresponding inequalities derived by vertices of $\boldsymbol{x}$.*
2. *If $f_i(x), i = 1, \dots, k$ are concave, then the inequality (10.11) is a consequence of the corresponding inequalities derived by vertices of $\boldsymbol{x}$.*
3. *If $g_j(x), j = 1, \dots, l$ are convex, then the inequality (10.12) is a consequence of the corresponding inequalities derived by vertices of $\boldsymbol{x}$.*

Proof. We prove the case 3; the other cases are proved analogously. Let $x^1, x^2 \in \boldsymbol{x}$ and consider a convex combination $x^0 := \lambda x^1 + (1-\lambda)x^2$ for any $\lambda \in [0,1]$. It suffices to show that the inequality derived from x^0 is a convex combination of those derived from x^1 and x^2. For x^1 and x^2 the associated systems (10.12) read respectively

$$(B^c - B^\Delta D_{\alpha^1})x \le B^c x^1 + B^\Delta v^1 - g(x^1), \tag{10.15}$$

$$(B^c - B^\Delta D_{\alpha^2})x \le B^c x^2 + B^\Delta v^2 - g(x^2), \tag{10.16}$$

where $\alpha_i^1 = \frac{1}{x_i^\Delta}(x_i^c - x_i^1)$, $\alpha_i^2 = \frac{1}{x_i^\Delta}(x_i^c - x_i^2)$, $v_i^1 = \frac{1}{x_i^\Delta}(x_i^c x_i^1 - \overline{x}_i \underline{x}_i)$ and $v_i^2 = \frac{1}{x_i^\Delta}(x_i^c x_i^2 - \overline{x}_i \underline{x}_i)$. Multiplying (10.15) by λ and (10.16) by $(1-\lambda)$, and summing up, we get

$$(B^c - B^\Delta D_\alpha)x \le B^c x^0 + B^\Delta v^0 - \lambda g(x^1) - (1-\lambda) g(x^2),$$

where $\alpha_i = \frac{1}{x_i^\Delta}(x_i^c - x_i^0)$ and $v_i^0 = \frac{1}{x_i^\Delta}(x_i^c x_i^0 - \overline{x}_i \underline{x}_i)$. Convexity of g implies

$$(B^c - B^\Delta D_\alpha)x \le B^c x^0 + B^\Delta v^0 - g(x^0),$$

which is inequality (10.12) corresponding to x^0. □

The functions $f_i(x), -f_i(x)$ or $g_j(x)$ need not be convex (and mostly they are not). However, if it is the case, Proposition 10.3 is fruitful only when x^0 is a vertex of $\boldsymbol{x}$; otherwise the resulting inequalities are redundant. Notice that this may not be the case for the original interval inequalities (10.9). When $f_i(x), -f_i(x)$ or $g_j(x)$ are not convex, nonvertex selection of $x^0 \in \boldsymbol{x}$ may be convenient. Informally speaking, the more nonconvex the functions are the more desirable a selection of an interior x^0 might be.

10.9 Examples

First, we start with an example that can be viewed as a “hard” instance for the classical techniques because the initial box is so called 2B-consistent (the domains of variables cannot be reduced if we consider the constraints separately) [59]. Also the recommended preconditioning of the system by the inverse of the Jacobian matrix for the midpoint values [59] makes almost no progress.

Example 10.6. Let us have the nonlinear system

$$y - \sin(x) = 0, \tag{10.17}$$

$$y - \cos(x + \pi/2) = 0. \tag{10.18}$$

for $x \in \boldsymbol{x} = [-\frac{\pi}{2}, \frac{\pi}{2}]$ and $y \in \boldsymbol{y} = [-1, 1]$. When using linearization by the mean value form, $\boldsymbol{A}$ is the Jacobian evaluated for the initial box $\boldsymbol{x} \times \boldsymbol{y}$.

$$\begin{pmatrix} -\cos(\boldsymbol{x}) & 1 \\ \cos(\boldsymbol{x}) & 1 \end{pmatrix}.$$

Figure 10.2 bellow illustrates the linearization for diverse centers of linearization. Since in this example linearization does not depend on x_2^0, we set this second coordinate to 0.

The decreasing curve corresponds to condition (10.17) the increasing curve to (10.18). The darker convex areas depict the linearization of the corresponding curves on given interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$. By taking the hull of the intersection of the convex areas we obtain the new enclosure

$$\boldsymbol{x}' = [-0.5708, 0.5708], \ \boldsymbol{y}' = [-0.7854, 0.7854],$$

which is depicted in Figure 10.4 a). For this system application of slopes gives the same contracted box.

Example 10.7. Let us have the nonlinear system

$$\pi^2 y - 4x^2 \sin(x) = 0, \tag{10.19}$$

$$y - \cos(x + \pi/2) = 0. \tag{10.20}$$

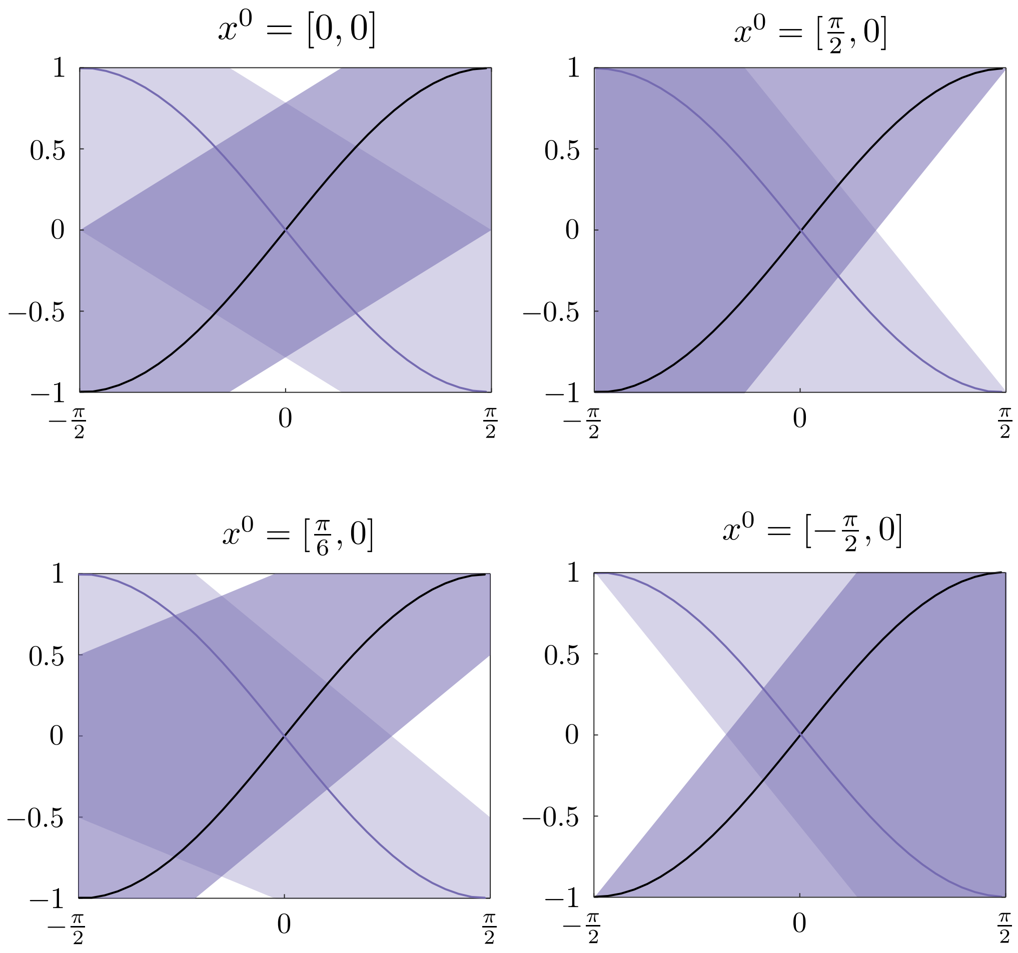


Figure 10.2: Four different linearizations depending on x^0 selection. The decreasing curve corresponds to constraint $y - \sin(x) = 0$ and the increasing curve to constraint $y - \cos(x + \pi/2) = 0$. The darker areas depicts the corresponding linearizations using the mean value form.

for $x \in \boldsymbol{x} = [-\frac{\pi}{2}, \frac{\pi}{2}]$, $y \in \boldsymbol{y} = [-1, 1]$. When using linearization by the mean value form, $\boldsymbol{A}$ is the Jacobian evaluated for the initial box $\boldsymbol{x} \times \boldsymbol{y}$

$$\begin{pmatrix} -8\boldsymbol{x}\sin(\boldsymbol{x}) - 4\boldsymbol{x}^2\cos(\boldsymbol{x}) & \pi^2 \\ \cos(\boldsymbol{x}) & 1 \end{pmatrix}.$$

Using this as an interval extension does not give narrow bounds (see Section 3.8). Hence, the initial enclosure can be reduced only by one dimension to

$$\boldsymbol{x}' = [-0.9597, 0.9597], \ \boldsymbol{y}' = [-1, 1].$$

In this example, the use of slopes helps. The linearization is depicted in Figure 10.3 and the resulting box is

$$\boldsymbol{x}'' = [-0.9597, 0.9597], \ \boldsymbol{y}'' = [-0.6110, 0.6110],$$

which is depicted in Figure 10.4 b).

10.10 Other reading

Many books address intervals in constraint satisfaction problems and global optimization, see, e.g., [59, 99, 105]. Various consistency techniques are introduced in, e.g. [17, 92]. Other techniques are, e.g., [25, 54]. For Jaulin's set inversion approach see [100]. For using intervals in quantified constraints [156] and for application of intervals to hybrid systems, see [65, 157].

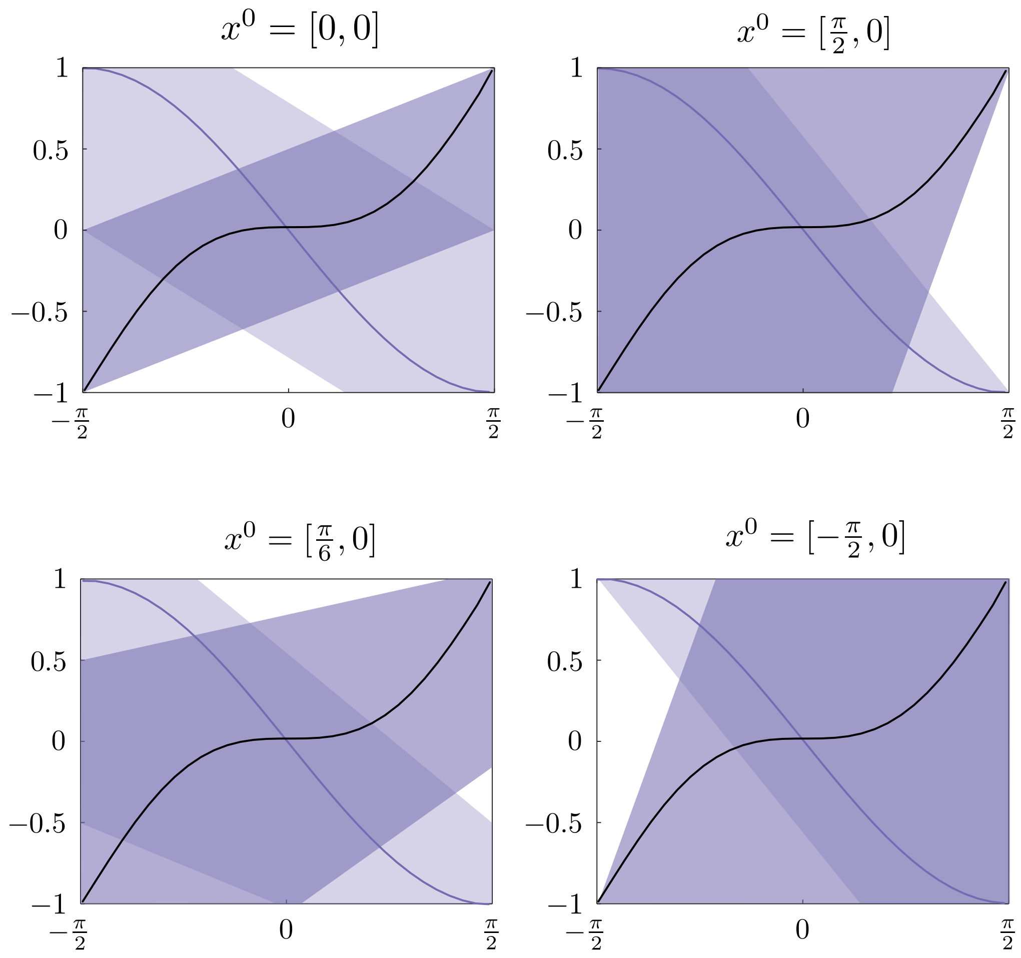


Figure 10.3: Four different linearizations depending on x^0 selection. The decreasing curve corresponds to constraint $\pi^2 y - 4x^2 \sin(x) = 0$ and the increasing curve to constraint $y - \cos(x + \pi/2) = 0$. The darker areas depicts the corresponding linearizations using slopes.

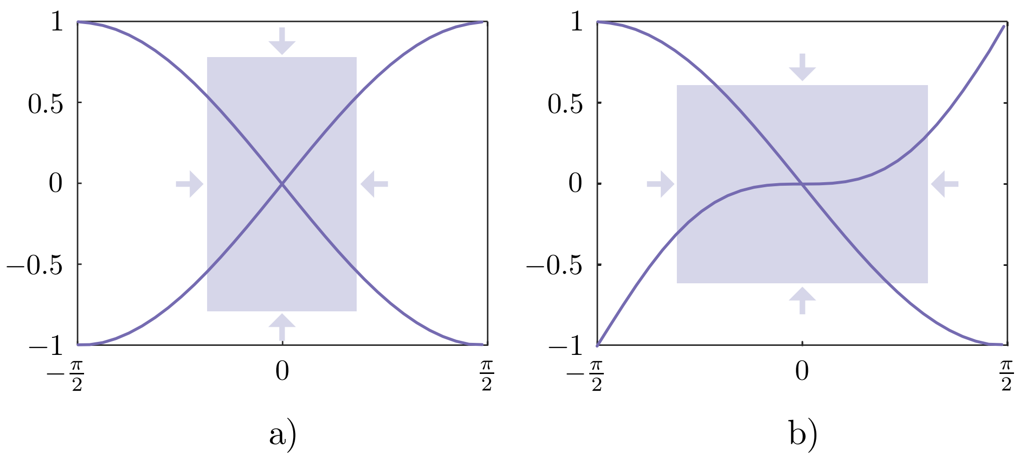


Figure 10.4: The resulting contracted boxes from the above examples; a) Example 10.6, b) Example 10.7.

11 Complexity of selected interval problems

- Brief introduction to computational complexity
- Complexity of various interval problems
- Polynomial cases and classes are characterized
- Sufficient conditions are pointed out

In the previous chapters we mentioned computational complexity issues of various problems. In this chapter we summarize more thoroughly the relation of computational complexity and interval analysis. Next, we gather the complexity results mentioned earlier in this work and we add some new topics of classical linear algebra – checking singularity, computing matrix inverse, bounding eigenvalues, checking positive (semi)definiteness or stability and some others.

Some questions may arise, when reading the previous works. Among all, it is the question about the equivalence of the notions NP-hardness and coNP-hardness. Some authors use these notions as synonyms, some authors distinguish between them. Another questions that may arise touches the representation and reducibility of interval problems in a given computational model. To shed more light (not only) on these issues we published a survey paper [85], which forms the basis of this chapter.

Nearly all problems become intractable when intervals are incorporated into matrices and vectors. However, there are many subclasses of problems that can be solved in a reasonable computational work.

For more complexity results or more depth we recommend, e.g., [40, 112, 173].

11.1 Complexity theory background

First, we present a brief introduction to computational complexity. Then, we return to interval linear algebra and introduce some well-known problems from the viewpoint of computational complexity.

11.1.1 Binary encoding and size of an instance

For a theoretic complexity classification of problems, it is a standard to use the Turing computation model. We assume that an instance of a computational problem is encoded in *binary encoding*, i.e., as a finite 0-1 sequence. Thus we cannot work with real-valued instances; instead we usually restrict ourselves to *rational numbers* expressed as fractions $\pm\frac{q}{r}$ with $q, r \in \mathbb{N}$ written down in binary and in the coprime form. Then, the *size* of a rational number $\pm\frac{q}{r}$ is understood as the number of bits necessary to represent the sign and both q and r (to be precise, one should also take care of delimiters). If an instance of a problem consists of multiple rational numbers $A = (a_1, \ldots, a_n)$ (e.g., when the input is a vector or a matrix), we define $\mathit{size}(A) = \sum_{i=1}^{n} \mathit{size}(a_i)$.

In interval problems, inputs of algorithms are usually interval numbers, vectors or matrices. When we say that an algorithm is to process an $m \times n$ interval matrix $\boldsymbol{A}$, we understand that the algorithm is given the pair $(\underline{A} \in \mathbb{Q}^{m\times n}, \overline{A} \in \mathbb{Q}^{m\times n})$ and that the size of the input is $L := \mathit{size}(\underline{A}) + \mathit{size}(\overline{A})$. Whenever we speak about *complexity* of such algorithm, we mean a function $\phi(L)$ counting the number of steps of the corresponding Turing machine as a function of the bit size L of the input $(\underline{A}, \overline{A})$.

Although the literature focuses mainly on the Turing model (and here we also do so), it is challenging to investigate the behavior of interval problems in other computational models, such as the Blum-Shub-Smale (BSS) model for real-valued computing [21] or the quantum model [9].

11.1.2 Function problems and decision problems

There are usually two kinds of problems:

- A *function problem* F is a total function $\mathsf{F} : \{0,1\}^* \to \{0,1\}^*$,
- A *decision problem* D is a total function $\mathsf{D} : \{0,1\}^* \to \{0,1\}$,

A function is total when it is defined for each input and $\{0,1\}^*$ is the set of all finite bit-strings.

Example 11.1 (Function problem)**.** Given a binary encoding for rational interval matrices and vectors define F as

$$\mathsf{F}(\boldsymbol{A}, \boldsymbol{b}) = \boldsymbol{x}, \text{ where } \boldsymbol{x} \text{ is the hull of } \boldsymbol{A}x = \boldsymbol{b}.$$

Example 11.2 (Decision problem)**.** Given a binary encoding for rational interval matrices and vectors define D as

$$\mathsf{D}(\boldsymbol{A}) = 1 \iff \boldsymbol{A} \text{ is regular.}$$

If for a problem A (either decision or functional) there exists a Turing machine computing $\mathsf{A}(x)$ for every $x \in \{0,1\}^*$, we say that A is *recursive.*

It is well known that many decision problems in mathematics are nonrecursive; e.g., deciding whether a given formula is provable in Zermelo-Fraenkel Set Theory is nonrecursive by the famous Gödel incompleteness theorem. Fortunately, a majority of decision problems in interval linear algebra are recursive. Such problems can usually be written down as arithmetic formulas (i.e., quantified formulas containing natural number constants, arithmetical operations $+, \times$, relations $=, \leq$ and propositional connectives). Such formulas are decidable (over the reals) by Tarski's quantifier elimination method (see [159, 160, 161]).

Example 11.3. Each matrix $A \in \boldsymbol{A}$ is nonsingular if and only if $(\forall A)[\underline{A} \leq A \leq \overline{A} \Rightarrow \det(A) \neq 0]$. This formula is arithmetical since $\det(\cdot)$ is a polynomial, and thus it is expressible in terms of $+, \times$.

Example 11.4. Is a given $\lambda \in \mathbb{Q}$ the largest eigenvalue of some symmetric $A \in \boldsymbol{A}$? This question can be written down as $(\exists A)[A = A^T \ \& \ \underline{A} \leq A \leq \overline{A} \ \& \ (\exists x \neq 0)[Ax = \lambda x] \ \& \ (\forall \lambda')\{(\exists x' \neq 0)[Ax' = \lambda' x'] \Rightarrow \lambda' \leq \lambda\}]$.

Although the quantifier elimination proves recursivity, it is a highly inefficient method from the practical viewpoint (the computation time can be doubly exponential in general). In spite of this, for many problems, reduction to the quantifier elimination is the only (and thus "the best") known algorithmic result.

11.1.3 Weak and strong polynomiality

Recursivity does not guarantee efficient solving of a problem. Usually, a problem A is said to be "efficiently" solvable if it is solvable in polynomial time, i.e., in at most $p(L)$ steps of the corresponding Turing machine, where p is a polynomial and L is the bit size of the input. The class of such problems is denoted by P.

Taking a more detailed viewpoint, this is a definition of polynomial-time solvability in the weak sense. In our context, we are usually processing a family $a_1, \ldots, a_n$ of rational numbers, where $L = \sum_{i=1}^{n} \mathit{size}(a_i)$, performing arithmetical operations $+, -, \times, \div, \leq$ on them. A *weakly polynomial* algorithm can perform at most $p_1(L)$ arithmetical operations with numbers of size at most $p_2(L)$ during its computation, where p_1, p_2 are polynomials.

If a polynomial-time algorithm satisfies the stronger property – that is, it performs at most $p_1(n)$ arithmetical operations with numbers of size at most $p_2(L)$ during its computation, we say that it is *strongly polynomial.* Simply said, the number of arithmetic operations of a strongly polynomial algorithm does not depend on the bit sizes of the inputs.

Example 11.5. Given a rational A and b, the question $(\exists x)(Ax = b)$ can be decided in strongly polynomial time (although it is not trivial to implement Gaussian elimination to yield a strongly polynomial algorithm, see [37]).

Example 11.6. On the contrary, the question $(\exists x)(Ax \leq b)$ (which is a form of linear programming) is known to be solvable in weakly polynomial time only and it is a major open question whether a strongly polynomial algorithm exists (this is Smales's Ninth Millenium Problem, see [207]).

Hence, whenever an interval-algebraic problem is solvable in polynomial time and requires linear programming (which is a frequent case), it is only a weakly polynomial result. This is why the cases, when interval-algebraic problems are solvable in strongly polynomial time, are of special interest.

11.1.4 NP and coNP

The class NP is the class of decision problems A with the following property: there is a polynomial p and a decision problem $\mathsf{B}(x, y)$, solvable in time polynomial in $size(x) + size(y)$, such that, for any instance $x \in \{0,1\}^*$,

$$\mathsf{A}(x) = 1 \iff \exists y \in \{0,1\}^{p(size(x))}\ \mathsf{B}(x, y) = 1. \tag{11.1}$$

where $\{0,1\}^{p(\cdot)}$ means that the size of the resulting 0-1 string is limited by a given polynomial. The string y is called a *witness* for the fact that $\mathsf{A}(x) = 1$. The algorithm for $\mathsf{B}(x, y)$ is called a *verifier.* Notice that such a verifier works in a polynomial time, however the algorithm for deciding $\mathsf{A}(x)$ does not have to do so. It is, in fact, still an open question whether $\mathsf{P} = \mathsf{NP}$. Philosophically, it goes with the intuition that coming up with the solution of the problem might be harder than just verifying that the solution is correct. Solving of such problems in NP is usually exponential with respect to the input size L.

Example 11.7. A lot of well-known problems are in NP: “Is a given graph colorable with 3 colors?”, “Does a given boolean formula have a satisfying assignment?”, “Does a given system $Ay \le b$ have an integral solution y?” For more problems see, e.g., [9, 43, 147].

The class coNP is characterized by replacement of the existential quantifier in (11.1):

$$\mathsf{A}(x) = 1 \iff \forall y \in \{0,1\}^{p(size(x))}\ \mathsf{B}(x, y) = 1.$$

It is easily seen that the class coNP is formed of complements of NP problems, and vice versa. (Recall that a decision problem A is a 0-1 function; its *complement* is defined as $\mathsf{coA} = 1 - \mathsf{A}$.)

Example 11.8. A well-known coNP problem is deciding whether a given boolean formula is a tautology.

It is easy to see that deciding a coNP-question can take exponential time since the $\forall$-quantifier ranges over a set exponentially large in the input size L. A lot of interval-based problems are in NP or coNP. Anyway, we should approach these problems with care.

Example 11.9 (Interval problem in NP)**.** Consider the problem of deciding whether an interval matrix is singular. More formally, let us have $\boldsymbol{A} \in \mathbb{Q}^{n\times n}$. We look for $A \in \boldsymbol{A}$ that is singular. A not completely correct statement would be that this problem belongs to NP, because a particular singular rational matrix $A_0 \in \boldsymbol{A}$ serves as a witness of singularity. To make it complete we need to prove that $size(A_0)$ is of polynomial size with respect to size of $\boldsymbol{A}$ (i.e., $L = size(\underline{A}) + size(\overline{A})$). Such a proof may be highly uncomfortable. We prefer to choose a different way. Using the Oettli–Prager theorem (Theorem 5.4) we have:

$$\begin{aligned}
&\exists A \in \boldsymbol{A} \text{ such that } A \text{ is singular},\\
\Leftrightarrow\quad &\exists A \in \boldsymbol{A},\ \exists x \ne 0\colon\ Ax = 0,\\
\Leftrightarrow\quad &\exists x \ne 0\colon\ -A_\Delta|x| \le A_c x \le A_\Delta|x|,\\
\Leftrightarrow\quad &\exists s \in \{\pm1\}^n\ \underbrace{\exists x\colon\ -A_\Delta D_s x \le A_c x \le A_\Delta D_s x,\ D_s x \ge 0,\ e^T D_s x \ge 1.}_{(*)}
\end{aligned} \tag{11.2}$$

Given an $s \in \{\pm 1\}^n$, the relation $(*)$ can be checked in polynomial time by linear programming. Thus, we can define the verifier $\mathsf{B}(\boldsymbol{A}, s)$ as the algorithm checking the validity of $(*)$. In fact, we have reformulated the $\exists$-question "is there a singular $A \in \boldsymbol{A}$?", into an equivalent $\exists$-question, "is there a sign vector $s \in \{\pm 1\}^n$ (orthant) such that $(*)$ holds true?", and now $\mathit{size}(s) \le L$ is obvious.

The method of (11.2) is known as *orthant decomposition* since it reduces the problem to inspection of orthants $D_s x \ge 0$, for every $s \in \{\pm 1\}^n$, and the work in each orthant is "easy" (here, the work in an orthant amounts to a single linear program). Many properties of interval data are described by sufficient and necessary conditions that use orthant decomposition. We have already met it in Section 5.2.

Example 11.10 (Interval problem in coNP)**.** Checking regularity is a complementary problem to checking singularity. Hence we immediately get that checking regularity of a general interval matrix is in coNP.

11.1.5 Decision problems: NP-, coNP-completeness

A decision problem A is *reducible* to a decision problem B (denoted $\mathsf{A} \le \mathsf{B}$) if there exists a polynomial-time computable function $g : \{0,1\}^* \to \{0,1\}^*$, called *reduction*, such that for every $x \in \{0,1\}^*$ we have

$$\mathsf{A}(x) = \mathsf{B}(g(x)). \tag{11.3}$$

Informally said, any algorithm for B can also be used for solving A – given an instance x of A, we can efficiently "translate" it into an instance $g(x)$ of the problem B and run the method deciding $\mathsf{B}(g(x))$, yielding the correct answer to $\mathsf{A}(x)$. Thus, any decision method for B is also a valid method for A, if we neglect a polynomial time for computation of the reduction g. In this sense we can say that if $\mathsf{A} \le \mathsf{B}$, then B is "as hard as A, or harder". If both $\mathsf{A} \le \mathsf{B}$ and $\mathsf{B} \le \mathsf{A}$, then problems A, B are called *polynomially equivalent*.

The relation $\le$ induces a partial ordering on classes of polynomially equivalent problems in NP and this ordering can be shown to have a maximum element. The problems in the maximum class are called NP-complete problems. And similarly, coNP has a class of coNP-complete problems. The classes are complementary – a problem A is NP-complete if and only if its complement is coNP-complete.

Let $\mathcal{X} \in \{\mathsf{NP}, \mathsf{coNP}\}$. If a problem B is $\mathcal{X}$-complete, any method for it can be understood as a universal method for any problem $\mathsf{A} \in \mathcal{X}$ (if we neglect a polynomial time needed for computing the reduction). Indeed, since B is the maximum element, we have $\mathsf{A} \le \mathsf{B}$ for any $\mathsf{A} \in \mathcal{X}$. It is generally believed that $\mathcal{X}$ contains problems that are not efficiently decidable. In NP, boolean satisfiability is a prominent example; in coNP, it is the tautology problem. Then, by $\le$-maximality, no $\mathcal{X}$-complete problem is efficiently decidable. This shows why a proof of $\mathcal{X}$-completeness of a newly studied problem is often understood as proof of its computational *intractability*.

From a practical perspective, a proof of $\mathcal{X}$ tells us that "nothing better than a superpolynomial-time algorithm can be expected". But formally we must distinguish between NP- and coNP-completeness because it is believed that NP-complete problems are not polynomially equivalent with coNP-complete problems (equivalence of these two classes is an open problem).

The usual way to prove the $\mathcal{X}$-completeness of a problem C is using the knowledge of some problem B being $\mathcal{X}$-complete and proving that 1) $\mathsf{B} \le \mathsf{C}$ and 2) $\mathsf{C} \in \mathcal{X}$. This is the method behind all $\mathcal{X}$-completeness proofs in this chapter.

11.1.6 Decision problems: NP- and coNP-hardness

Here we restrict ourselves to NP-hard problems as the reasoning for coNP-hard problems is analogous. In the previous section we spoke about NP-complete problems as the $\le$-maximum elements in NP.

We say that a decision problem H, not necessarily in NP, satisfying $\mathsf{C} \le \mathsf{H}$ for an NP-complete problem C, is NP-hard. Clearly, NP-complete problems are exactly those NP-hard problems which are in NP. But we might encounter a problem H for which we do not have a proof for $\mathsf{H} \in \mathsf{NP}$, but still it might be possible to prove $\mathsf{C} \le \mathsf{H}$. This also means a bad news for practical computing; the problem H is computationally intractable (but we might possibly need even worse computation time than for problems in NP).

Proving that a decision problem is NP-hard is a weaker theoretical result than a proof that a decision problem is NP-complete. It is followed by an inspection why it is difficult to prove the presence in NP. If we are unsuccessful in placing the problem in NP or coNP, being unable to write down the $\exists$- or $\forall$-definition, it might be appropriate to place the problem H into higher levels of the Polynomial Time Hierarchy, or even higher, such as the PSPACE-level; for details see, e.g., [9, 147].

11.1.7 Functional problems: efficient solvability and NP-hardness

Functional problems are problems of computing values of general functions, in contrast to decision problems where we expect only a YES/NO answers. We also want to classify functional problems from the complexity-theoretic perspective, whether they are “efficiently solvable”, or “intractable”, as we did with decision problems. Efficient solvability of a functional problem is again generally understood as polynomial-time computability. To define NP-hardness, we need the following notion of reduction: a decision problem D is *reducible* to a functional problem F, if there exist functions $g : \{0,1\}^* \to \{0,1\}^*$ and $h : \{0,1\}^* \to \{0,1\}$, both computable in polynomial time, such that

$$\mathsf{D}(x) = h(\mathsf{F}(g(x))) \quad \text{for every} \quad x \in \{0,1\}^*. \tag{11.4}$$

The role of g is analogous to (11.3): it translates an instance x of D into an instance $g(x)$ of F. What is new here is the function h. Since F is a functional problem, the value $\mathsf{F}(g(x))$ can be an arbitrary bit string (say, a binary representation of a rational number); then we need another efficiently computable function h translating the value $\mathsf{F}(g(x))$ into a 1-0 value giving the YES/NO answer to $\mathsf{D}(x)$.

Example 11.11. Let D be a problem of deciding whether a square rational matrix A is regular. It is reducible to the functional problem F of computing the rank r of A. It suffices to define $g(A) = A$ and $h(r) = 1 - \min\{n - r, 1\}$.

Now, a functional problem F is NP-hard if there is an NP-hard decision problem reducible to F. For example, the functional problem of counting the number of ones in the truth-table of a given boolean formula is NP-hard since this information allows us to decide whether or not the formula is satisfiable.

We could also try to define coNP-hardness of a functional problem G in terms of reducibility of a coNP-hard decision problem C to G via (11.4). But this is superfluous because here NP-hardness and coNP-hardness would coincide. Indeed, if we can reduce a coNP-hard problem C to a functional problem G via (g, h), then we can also reduce the NP-hard problem coC to G via $(g, 1 - h)$. Thus, in case of functional problems, we speak about NP-hardness only.

11.1.8 Decision problems: NP-hardness vs. coNP-hardness

In literature, the notions of NP-hardness and coNP-hardness are sometimes used quite freely even for decision problems. Sometimes we can read that a decision problem is "NP-hard", even if it would qualify as a coNP-hard problem under our definition based on the reduction (11.3). This is nothing serious as far as we are aware. It depends on how the authors understands the notion of a reduction between two decision problems. We have used the *many-one* reduction (11.3), known also as *Karp* reduction, between two decision problems. This is a standard in complexity-theoretic literature.

However, one could use a more general reduction between two decision problems A, B. For example, taking inspiration from (11.4), we could define

$$\mathsf{A} \le' \mathsf{B} \iff \mathsf{A}(x) = h(\mathsf{B}(g(x)))$$

for some polynomial-time computable functions g, h. Then the notions of $\le'$-NP-hardness and $\le'$-coNP-hardness coincide and need not be distinguished. Observe that h must be a function from $\{0, 1\}$ to $\{0, 1\}$ and there are only two such nonconstant functions: $h_1(\xi) = \xi$ and $h_2(\xi) = 1 - \xi$. If we admit only h_1, we get the many-one reduction; if we admit also the negation h_2, we have a generalized reduction under which a problem is NP-hard if and only if it is coNP-hard. Thus, the notions of NP-hardness and coNP-hardness based on many-one reductions do not coincide just because many-one reductions do not admit the negation of the output of $\mathsf{B}(g(x))$.

To be fully precise, one should always say "a problem A is $\mathcal{X}$-hard with respect to a particular reduction $\preceq$". For example, in the previous sections we spoke about $\mathcal{X}$-hard problems for $\mathcal{X} \in \{\mathsf{NP}, \mathsf{coNP}\}$ with respect to the many-one reduction (11.3). If another author uses $\mathcal{X}$-hardness with respect to $\le'$ (e.g., because she/he considers the ban of negation as too restrictive in her/his context), then she/he need not distinguish between NP-hardness and coNP-hardness.

For discussion on more types of reductions with respect to NP-hardness and coNP-hardness see, e.g., [85].

11.1.9 A reduction-free definition of hardness

For practical purposes, when we do not want to care too much about properties of particular reductions, we can define the notion of a "hard" problem H (either decision or functional) intuitively as a problem fulfilling this implication:

if H is decidable/solvable in polynomial time, then $\mathsf{P} = \mathsf{NP}$.

This is usually satisfactory for the practical understanding of the notion of computational hardness. (Under this definition: if $\mathsf{P} = \mathsf{NP}$, then every decision problem is hard; and if $\mathsf{P} \neq \mathsf{NP}$, then the class of hard decision problems is exactly the class of decision problems not decidable in polynomial time, including all NP-hard and coNP-hard decision problems.)

Even if we accept this definition and do not speak about reductions explicitly, all hardness proofs (at least implicitly) contain some kinds of reductions of previously known hard problems to the newly studied ones.

11.2 Interval linear algebra

In the following sections we will deal with various problems from the area of interval linear algebra. There are many interesting topics that are unfortunately beyond the scope of this work. We have met some of them in the previous chapters and we are going to remind them. Moreover, we add another basic topics from introductory courses to linear algebra – matrix inverse, eigenvalues and eigenvectors, positive (semi)definiteness and stability – the topics we touched only slightly. The rest of this chapter will offer a great disappointment and also a great challenge since introducing intervals into a classical linear algebra makes solving most of the problems intractable. That is why we look for solving relaxed problems, special feasible subclasses of problems or for sufficient conditions checkable in polynomial time. Interval linear algebra still offers many open problems and thus open space for further research. At the end of each section we present a summary of problems and their complexity. If we only know that a problem is weakly polynomial yet, we just write that it belongs to the class P. When complexity of a problem is not known to our best knowledge (or it is an open problem), we mark it with a question mark.

11.3 Regularity and singularity

Deciding regularity and singularity is a key task in interval linear algebra. It forms an initial step of many algorithms. We tackled this topic in Section 4.1.

Checking singularity is NP-hard [173]. In the Example 11.9 we saw a construction of a polynomial witness $s \in \{\pm 1\}^n$ certifying that an interval matrix is singular. Hence, we get that checking singularity of a general interval matrix is NP-complete. Clearly, checking regularity as the complementary problem to singularity is coNP-complete.

The sufficient and necessary conditions for checking regularity are of exponential nature. In [179] you can see 40 of them.

Fortunately, there are some sufficient conditions that are computable in polynomial time. Some of them were mentioned in Section 4.1. It is advantageous to have more conditions, because some of them may suit better to a certain class of matrices or limits of our software tools. Here we present two more sufficient conditions for checking regularity and four sufficient conditions for checking singularity.

Theorem 11.12 (Sufficient conditions for regularity)**.** *An interval matrix* $\boldsymbol{A}$ *is regular if at least one of the following conditions holds:*

1. $\lambda_{\max}(A_\Delta^T A_\Delta) < \lambda_{\min}(A_c^T A_c)$ [193] ,
2. $A_c^T A_c - \|A_\Delta^T A_\Delta\| I$ *is positive definite for some consistent matrix norm* $\|\cdot\|$ [164].

Theorem 11.13 (Sufficient conditions for singularity)**.** *An interval matrix* $\boldsymbol{A}$ *is singular if at least one of the following conditions holds:*

1. $\lambda_{\max}(A_c^T A_c) \leq \lambda_{\min}(A_\Delta^T A_\Delta)$ [164],
2. $\max_j(|A_c^{-1}|A_\Delta)_{jj} \geq 1$ [166],
3. $(A_\Delta - |A_c|)^{-1} \geq 0$ [173],
4. $A_\Delta^T A_\Delta \; - A_c^T A_c$ *is positive semidefinite* [164].

In Section 4.1 we have already met some classes of interval matrices that are regular (strictly diagonally dominant matrices, M-matrices and H-matrices). Checking that a matrix belongs to these classes can be done in strongly polynomial time.

11.3.1 Summary

Problem	**Complexity**
Is $\boldsymbol{A}$ regular?	coNP-complete
Is $\boldsymbol{A}$ singular?	NP-complete

11.4 Full column rank

Checking full column rank was addressed in Section 7.3. Deciding whether an interval matrix has full column rank is connected to checking regularity. If an interval matrix $\boldsymbol{A}$ of size $m \times n$, $m \geq n$, contains a regular interval sub-matrix of size n, then obviously $\boldsymbol{A}$ has full column rank. What is surprising is that the implication does not hold conversely (in contrast to real matrices). The following interval matrix by Irene Sharaya (see [204]) serves as a counterexample.

Example 11.14. The matrix

$$\boldsymbol{A} = \begin{pmatrix} 1 & [0,1] \\ -1 & [0,1] \\ [-1,1] & 1 \end{pmatrix},$$

has full column rank, but contains no regular submatrix of size 2.

For square matrices, checking regularity can can be polynomially reduced to checking full column rank (we just check whether $\boldsymbol{A}$ has full column rank). Therefore, checking full column rank is coNP-hard. Finding a polynomial certificate for an interval matrix not having full column rank can be done by orthant decomposition similarly as in the case of singularity. That is why, checking full column rank is coNP-complete.

Again, fortunately, we have some sufficient conditions that are computable in polynomial time. In Section 7.3 we mentioned several polynomially checkable conditions for an interval matrix having full column rank.

11.4.1 Summary

Problem	**Complexity**
Does $\boldsymbol{A}$ have full column rank?	coNP-complete

11.5 Solving a system of linear equations

Solving of interval linear systems was the main topic of Chapter 5 and 6. We have the following theorem by Rohn [171].

Theorem 11.15. *Computing an enclosure of the solution set of $\boldsymbol{A}x = \boldsymbol{b}$ when it is bounded, and otherwise returning an error message, is* NP*-hard.*

If such an algorithm existed, it could be used to decide regularity of an interval matrix, since regularity of $\boldsymbol{A}$ implies bounded solution set of $\boldsymbol{A}x = \boldsymbol{b}$ for arbitrary $\boldsymbol{b}$ [171].

Computing the optimal bounds (the hull) on the solution set is also NP-hard [184]. The problem stays NP-hard even if we limit widths of intervals of the system matrix with some $\delta > 0$, or allow the bounds to consist of 0 or 1 only [112]. Unfortunately, even computing various ε-approximations of the hull components is an NP-hard problem [112].

Theorem 11.16. *Let $\varepsilon > 0$, then computing the relative and absolute ε-approximation of the hull (its components) of $\boldsymbol{A}x = \boldsymbol{b}$ are both* NP*-hard problems.*

Fortunately, in Chapter 5 we saw various methods and special conditions on $\boldsymbol{A}, \boldsymbol{b}$ under which the hull can be computed in polynomial time.

11.5.1 Overdetermined systems

In Chapter 6 we defined overdetermined systems. The problem of computing the interval hull of Σ^{lsq} is NP-hard, since when $\boldsymbol{A}$ is square and regular, then $\Sigma^{lsq} = \Sigma$.

11.5.2 Restricted interval coefficients

We can try to identify some classes of systems with exact hull computation algorithms that run in polynomial time. If we restrict the right hand side $\boldsymbol{b}$ to contain only degenerate intervals, we have $\boldsymbol{A}x = b$. Such a problem is still NP-hard [112]. If we, however, restrict the matrix to be consisting only of degenerate intervals A and we have a system $Ax = \boldsymbol{b}$, then, computing the exact bounds of the solution set is polynomial, since it can be rewritten as a pair of linear programs

$$\max(\min)\ e_i^T x, \quad Ax \geq \underline{b}, \quad Ax \leq \overline{b},$$

for each variable x_i.

11.5.3 Structured systems

We can also explore band and sparse matrices.

Definition 11.17. A matrix $\boldsymbol{A}$ is a *w-band* matrix if $\boldsymbol{a}_{ij} = 0$ for $|i - j| \geq w$.

Band matrices with $d = 1$ are diagonal and computing the hull is clearly strongly polynomial. For $d = 2$ (tridiagonal matrix) it is an open problem. And for $d \geq 3$ it is already NP-hard [111]. We can also get strong polynomial time in case of bidiagonal systems.

Proposition 11.18 (Horáček et al., [85])**.** *For a bidiagonal matrix (the matrix with only the main diagonal and an arbitrary neighboring diagonal) computing the exact hull of $\boldsymbol{A}x = \boldsymbol{b}$ is strongly polynomial.*

Proof. Without the loss of generality let us suppose that a matrix $\boldsymbol{A}$ consists of the main diagonal and the one below it. By the forward substitution, we have $\boldsymbol{x}_1 = \frac{\boldsymbol{b}_1}{\boldsymbol{a}_{11}}$ and

$$\boldsymbol{x}_i = \frac{\boldsymbol{b}_i - \boldsymbol{a}_{i,i-1}\boldsymbol{x}_{i-1}}{\boldsymbol{a}_{ii}}, \quad i = 2, \dots, n.$$

By induction, $\boldsymbol{x}_{i-1}$ is optimally computed with no use of interval coefficients of the ith equation. Since an evaluation in interval arithmetic is optimal when there are no multiple occurrences of variables (Theorem 3.13), $\boldsymbol{x}_i$ is optimal as well. □

Definition 11.19. A matrix $\boldsymbol{A}$ is *d-sparse* if in each row i at most d elements $\boldsymbol{a}_{ij} \neq 0$.

For sparse matrices with $d = 1$ computing the hull is clearly strongly polynomial. For $d \geq 2$ it is again NP-hard [112]. Nevertheless, if we combine w-band matrix with system coefficient bounds coming from a given finite set of rational numbers, then we have a polynomial algorithm for computing the hull [112].

11.5.4 Parametric systems

A natural generalization of an interval linear system is by incorporating linear dependencies of coefficients. That is, we have a family of linear systems

$$A(p)x = b(p), \quad p \in \boldsymbol{p}, \tag{11.5}$$

where $A(p) = \sum_{k=1}^{K} A^k p_k$, $b(p) = \sum_{k=1}^{K} b^k p_k$ and K is number of parameters. Here, p is a vector of parameters varying in $\boldsymbol{p}$. Since this concept generalizes the standard interval systems, many related problems are intractable [206]. The reason is that an interval system $\boldsymbol{A}x = \boldsymbol{b}$ can be considered as a parametric system $A(\boldsymbol{p})x = b(\boldsymbol{p})$ with $n^2 + n$ interval parameters.

Nevertheless, we point out one particular efficiently solvable problem. Given $x \in \mathbb{R}^n$, deciding whether it is a solution of a standard interval system $\boldsymbol{A}x = \boldsymbol{b}$ is strongly polynomial. For systems with linear dependencies, the problem still stays polynomial (just by checking if x satisfies the Oettli–Prager theorem), but we can show weak polynomiality only; this is achieved by rewriting (11.5) as a linear program. For more information on parametric systems see, e.g., [67, 151, 206].

11.5.5 Summary

Problem	**Complexity**
Is x a solution of $\boldsymbol{A}x = \boldsymbol{b}$?	strongly P
Computing the hull of $\boldsymbol{A}x = \boldsymbol{b}$	NP-hard
Computing the hull of $\boldsymbol{A}x = b$	NP-hard
Computing the hull of $Ax = \boldsymbol{b}$	P
Computing the hull of $\boldsymbol{A}x = \boldsymbol{b}$, where $\boldsymbol{A}$ is regular	NP-hard
Computing the hull of $\boldsymbol{A}x = \boldsymbol{b}$, where $\boldsymbol{A}$ is an M-matrix	strongly P
Computing the hull of $\boldsymbol{A}x = \boldsymbol{b}$, where $\boldsymbol{A}$ is diagonal	strongly P
Computing the hull of $\boldsymbol{A}x = \boldsymbol{b}$, where $\boldsymbol{A}$ is bidiagonal	strongly P
Computing the hull of $\boldsymbol{A}x = \boldsymbol{b}$, where $\boldsymbol{A}$ is tridiagonal	?
Computing the hull of $\boldsymbol{A}x = \boldsymbol{b}$, where $\boldsymbol{A}$ is 3-band	NP-hard
Computing the hull of $\boldsymbol{A}x = \boldsymbol{b}$, where $\boldsymbol{A}$ is 1-sparse	strongly P
Computing the hull of $\boldsymbol{A}x = \boldsymbol{b}$, where $\boldsymbol{A}$ is 2-sparse	NP-hard
Computing the exact least squares hull of $\boldsymbol{A}x = \boldsymbol{b}$	NP-hard
Is Σ bounded?	coNP-complete
Computing the hull of $A(\boldsymbol{p})x = b(\boldsymbol{p})$	NP-hard
Is x a solution of $A(\boldsymbol{p})x = b(\boldsymbol{p})$?	P

11.6 Matrix inverse

Interval inverse matrix was defined in Section 4.2. For a square interval matrix $\boldsymbol{A}$ it can be computed using knowledge of inverses of 2^{2n-1} matrices in the form

$$A_{yz} = A_c - D_y A_\Delta D_z,$$

where y, z are n-dimensional vectors from Y_n; [169].

Theorem 11.20. *Let $\boldsymbol{A}$ be regular. Then its inverse $\boldsymbol{A}^{-1} = [\underline{B}, \overline{B}]$ is described by*

$$\underline{B}_{ij} = \min_{y,\, z \in Y_n} \{(A_{yz}^{-1})_{ij}\},$$

$$\overline{B}_{ij} = \max_{y,\, z \in Y_n} \{(A_{yz}^{-1})_{ij}\},$$

for $i, j = 1, \ldots, n$.

Since ith column of the matrix inverse of $\boldsymbol{A}$ is equivalently computed as the hull of $\boldsymbol{A}x = e_i$, the problem is NP-hard (for another reasoning see [30]).

However, when $A_c = I$, we can compute the exact inverse in polynomial time according to the next theorem from [180].

Theorem 11.21. *Let $\boldsymbol{A}$ be a regular interval matrix with $A_c = I$. Let $M = (I - A_\Delta)^{-1}$. Then its inverse $\boldsymbol{A}^{-1} = [\underline{B}, \overline{B}]$ is described by*

$$\begin{aligned} \underline{B} &= -M + D_v, \\ \overline{B} &= M, \end{aligned}$$

where $v_j = \frac{2m_{jj}^2}{2m_{jj}-1}$ for $j = 1, \dots, n$, with m_{jj} being the main diagonal elements of M.

When an interval matrix is of uniform width, i.e., $\boldsymbol{A} = [A_c - \alpha E, A_c + \alpha E]$, for a sufficiently small $\alpha > 0$ the inverse can be also expressed explicitly [183].

If we wish to only compute an enclosure $\boldsymbol{B}$ of the matrix inverse we can use any method for computing enclosures of interval linear systems. We get the ith column of $\boldsymbol{B}$ by solving the systems $\boldsymbol{A}x = e_i$.

Not all interval matrix classes imply intractability. In Section 4.2 we showed that checking inverse nonnegativity and also computing the exact interval inverse of an inverse nonnegative matrix are strongly polynomial tasks (see Theorem 4.13).

11.6.1 Summary

Problem	**Complexity**
Computing the exact inverse of $\boldsymbol{A}$	NP-hard
Computing the exact inverse of $\boldsymbol{A}$, $A_c = I$	strongly P
Computing the exact inverse of $\boldsymbol{A}$, $A_\Delta = \alpha E$, α suff. small	strongly P
Is $\boldsymbol{A}$ inverse nonnegative?	strongly P
Computing the exact inverse of inverse nonnegative $\boldsymbol{A}$	strongly P

11.7 Solvability of a linear system

In Chapter 7 we distinguished between weak and strong solvability. Checking whether an interval linear system is solvable is an NP-hard problem [112]. The sign coordinates of the orthant containing the solution can serve as a polynomial witness and existence of a solution can be verified by linear programming, hence this problem is NP-complete. Checking unsolvability as its complement is coNP-complete. The problem of deciding strong solvability is also coNP-complete. It can be reformulated as checking unsolvability of a certain linear system using the well known Farkas lemma, see [178].

Sometimes, we look only for a nonnegative solution (i.e., *nonnegative solvability*). Checking whether an interval linear system has a nonnegative solution is weakly polynomial. We know the orthant in which the solution should lie (the positive one). Therefore, we can get rid of the absolute values in the Oettli–Prager theorem and apply linear programming. However, checking whether a system is nonnengative strongly solvable is still coNP-complete [40]. We summarize the results in the following table.

Theorem 11.22. *Checking various types of solvability of $\boldsymbol{A}x = \boldsymbol{b}$ is of the following complexity:*

	weak	*strong*
solvability	NP-*complete*	coNP-*complete*
nonnegative solvability	P	coNP-*complete*

In Chapter 7 we introduced several methods for detecting solvability and unsolvability that work in polynomial time.

11.7.1 Linear inequalities

Just for comparison, considering systems of interval linear inequalities, the problems of checking various types of solvability become much easier. The results from [40] are summarized in the following table.

Theorem 11.23. *Checking various types of solvability of $\boldsymbol{A}x \leq \boldsymbol{b}$ is of the following complexity.*

	weak	*strong*
solvability	NP-*complete*	P
nonnegative solvability	P	P

We also would like to mention an interesting nontrivial property of strong solvability of systems of interval linear inequalities. When a system $\boldsymbol{A}x \leq \boldsymbol{b}$ is strongly solvable (i.e., every $Ax \leq b$ has a solution), then there exists a solution x satisfying $Ax \leq b$ for every $A \in \boldsymbol{A}$ and $b \in \boldsymbol{b}$ [40].

11.7.2 $\forall\exists$-solutions.

Let us return to interval linear systems. The concept of a weak solution employs existential quantifiers: x is a solution if $\exists A \in \boldsymbol{A}$, $\exists b \in \boldsymbol{b} : Ax = b$. Nevertheless, in some applications, another quantification makes sense. In particular, $\forall\exists$ quantification was deeply studied [203]. For illustration of complexity of such solution, we will focus on two concepts of solutions – tolerance [40] and control solution [40, 200].

Definition 11.24. A vector x is a *tolerance* solution of $\boldsymbol{A}x = \boldsymbol{b}$ if

$$\forall A \in \boldsymbol{A}, \exists b \in \boldsymbol{b}, \quad Ax = b.$$

A vector x is a *control* solution of $\boldsymbol{A}x = \boldsymbol{b}$ if

$$\forall b \in \boldsymbol{b}, \exists A \in \boldsymbol{A}, \quad Ax = b.$$

Notice that a tolerance solution can equivalently be characterized as $\{Ax \mid A \in \boldsymbol{A}\} \subseteq \boldsymbol{b}$ and a control solution as $\boldsymbol{b} \subseteq \{Ax \mid A \in \boldsymbol{A}\}$.

Both solutions can be described by a slight modification of the Oettli–Prager theorem (one sign change in the Oettli–Prager formula) [40].

Theorem 11.25. *Let us have a system $\boldsymbol{A}x = \boldsymbol{b}$, then x is*

- *a tolerance solution if it satisfies $|A_c x - b_c| \leq -A_\Delta |x| + \delta$,*
- *a control solution if it satisfies $|A_c x - b_c| \leq A_\Delta |x| - \delta$.*

In case of tolerance solution this change makes checking whether a systems has this kind of solution decidable in weakly polynomial time. In the case of control solution the decision problem stays NP-complete [112].

11.7.3 Summary

Problem	Complexity
Is $\boldsymbol{A}x = \boldsymbol{b}$ solvable?	NP-complete
Is $\boldsymbol{A}x = \boldsymbol{b}$ strongly solvable?	coNP-complete
Is $\boldsymbol{A}x = \boldsymbol{b}$ nonnegative solvable?	P
Is $\boldsymbol{A}x = \boldsymbol{b}$ nonnegative strongly solvable?	coNP-complete
Is $\boldsymbol{A}x \leq \boldsymbol{b}$ solvable?	NP-complete
Is $\boldsymbol{A}x \leq \boldsymbol{b}$ strongly solvable?	P
Is $\boldsymbol{A}x \leq \boldsymbol{b}$ nonnegative solvable?	P
Is $\boldsymbol{A}x \leq \boldsymbol{b}$ nonnegative strongly solvable?	P
Is x a tolerance solution of $\boldsymbol{A}x = \boldsymbol{b}$?	strongly P
Is x a control solution of $\boldsymbol{A}x = \boldsymbol{b}$?	strongly P
Does $\boldsymbol{A}x = \boldsymbol{b}$ have a tolerance solution?	P
Does $\boldsymbol{A}x = \boldsymbol{b}$ have a control solution?	NP-complete

11.8 Determinant

Determinants of interval matrices were studied in Chapter 8. Several result about complexity of such a problem were stated there. Here we summarize the results in the following table.

11.8.1 Summary

Problem	Complexity
Computing $\underline{\det}(\boldsymbol{A})$	NP-hard
Computing $\overline{\det}(\boldsymbol{A})$	NP-hard
Computing relative ε-approximation of $\det(\boldsymbol{A})$ for $0 < \varepsilon < 1$	NP-hard
Computing absolute ε-approximation of $\det(\boldsymbol{A})$ for $0 < \varepsilon$	NP-hard
Computing $\overline{\det(\boldsymbol{A}^S)}$ for positive definite $\boldsymbol{A}^S$	P
Computing $\underline{\det}(\boldsymbol{A})$, where $\boldsymbol{A}$ is a tridiagonal H-matrix	strongly P
Computing $\overline{\det}(\boldsymbol{A})$, where $\boldsymbol{A}$ is a tridiagonal H-matrix	strongly P
Computing $\underline{\det}(\boldsymbol{A})$ for $\boldsymbol{A}$ regular with $A_c = I$	strongly P
Computing $\overline{\det}(\boldsymbol{A})$ for $\boldsymbol{A}$ regular with $A_c = I$	?

11.9 Eigenvalues

We briefly start with general matrices, then we continue with the symmetric case. Checking singularity of $\boldsymbol{A}$ can be polynomially reduced to checking whether 0 is an eigenvalue of some matrix $A \in \boldsymbol{A}$. Using the reasoning from Section 11.1.7, checking whether λ is an eigenvalue of some matrix $A \in \boldsymbol{A}$ is NP-hard. Surprisingly, checking an eigenvector is strongly polynomial [168].

How it is with the Perron theory? An interval matrix $\boldsymbol{A} \in \mathbb{IR}^{n\times n}$ is *nonnegative irreducible* if every $A \in \boldsymbol{A}$ is nonnegative irreducible (a definition can be found in [91]). For Perron vectors (positive vectors corresponding to the dominant eigenvalues), we have the following result [177].

Theorem 11.26. *Let $\boldsymbol{A}$ be nonnegative irreducible. Then the problem of deciding whether x is the Perron eigenvector of some matrix $A \in \boldsymbol{A}$ is strongly polynomial.*

For the sake of simplicity we mentioned only some results considering eigenvalues of a general matrix $\boldsymbol{A}$. We will go into more detail with symmetric matrices, which have real eigenvalues. In Chapter 8 we defined a symmetric interval matrix as a subset of all symmetric matrices in $\boldsymbol{A}$, that is,

$$\boldsymbol{A}^S := \{A \in \boldsymbol{A} \mid A = A^T\}.$$

For a symmetric $A \in \mathbb{R}^{n\times n}$, we denote its smallest and largest eigenvalue by $\lambda_{\min}(A)$ and $\lambda_{\max}(A)$ respectively. For a symmetric interval matrix $\boldsymbol{A}^S$, we define the smallest and largest eigenvalue as

$$\begin{aligned}\lambda_{\min}(\boldsymbol{A}^S) &:= \min\{\lambda_{\min}(A) \mid A \in \boldsymbol{A}^S\},\\ \lambda_{\max}(\boldsymbol{A}^S) &:= \max\{\lambda_{\max}(A) \mid A \in \boldsymbol{A}^S\}.\end{aligned}$$

Even if we consider the symmetric case, some problems remain NP-hard [112, 173].

Theorem 11.27. *Let $A_c \in \mathbb{Q}^{n\times n}$ be a symmetric positive definite and entry-wise nonnegative matrix, and $A_\Delta = E$. Then*

- *checking whether 0 is an eigenvalue of some matrix $A \in \boldsymbol{A}^S$ is NP-hard,*
- *checking $\lambda_{\max}(\boldsymbol{A}^S) \in (\underline{a}, \overline{a})$ for a given open interval $(\underline{a}, \overline{a})$ is coNP-hard.*

However, there are some known subclasses for which the eigenvalue range or at least one of the extremal eigenvalues can be determined efficiently [72]:

- If A_c is *essentially nonnegative*, i.e., $(A_c)_{ij} \geq 0\ \forall i \neq j$, then $\lambda_{\max}(\boldsymbol{A}^S) = \lambda_{\max}(\overline{A})$.

- If A_Δ is *diagonal*, then $\lambda_{\min}(\boldsymbol{A}^S) = \lambda_{\min}(\underline{A})$ and $\lambda_{\max}(\boldsymbol{A}^S) = \lambda_{\max}(\overline{A})$.

In contrast to the extremal eigenvalues $\lambda_{\min}(\boldsymbol{A}^S)$ and $\lambda_{\max}(\boldsymbol{A}^S)$, the largest of the minimal eigenvalues and the smallest of the largest eigenvalues,

$$\max\{\lambda_{\min}(A) \mid A \in \boldsymbol{A}^S\},$$
$$\min\{\lambda_{\max}(A) \mid A \in \boldsymbol{A}^S\},$$

can be computed with an arbitrary precision in polynomial time by semidefinite programming [98]. As in the general case, checking whether a given vector $0 \neq x \in \mathbb{R}^n$ is an eigenvector of some matrix in $\boldsymbol{A}^S$ is a polynomial time problem. Nevertheless, strong polynomiality has not been proved yet.

We already know that computing exact bounds on many problems with interval data is intractable. Since we can do no better, we can inspect the hardness of various approximations of their solutions. The terms absolute and relative approximation are meant in the same way as in Section 8.3. While doing this we use the following assumption: *Throughout this section, we consider a computational model, in which the exact eigenvalues of rational symmetric matrices are polynomially computable.* The table below from [72] summarizes the main results. We use the symbol ∞ in case there is no finite approximation factor with polynomial complexity.

Theorem 11.28. *Approximating the extremal eigenvalues of $\boldsymbol{A}^S$ is of the following complexity.*

	abs. error	*rel. error*
NP-*hard with error*	*any*	< 1
polynomial with error	∞	1

The table below, also from [72], gives analogous results for the specific case of approximating $\lambda_{\max}(\boldsymbol{A}^S)$ when A_c is positive semidefinite.

Theorem 11.29. *Approximating the extremal eigenvalues of $\boldsymbol{A}^S$ with A_c rational positive semidefinite is of the following complexity.*

	abs. error	*rel. error*
NP-*hard with error*	*any*	$1/(32n^4)$
polynomial with error	∞	$1/3$

The tables sums up the generalized idea behind several theorems on computing extremal eigenvalues. For more information and formal details see [72].

At the end of this section we mention spectral radius.

Definition 11.30. Let $\boldsymbol{A} \in \mathbb{IR}^{n\times n}$, we define the range of *spectral radius* naturally as

$$\varrho(\boldsymbol{A}) = \{\varrho(A) : A \in \boldsymbol{A}\}.$$

Notice that $\varrho(\boldsymbol{A})$ is a compact real interval due to continuity of eigenvalues. We define spectral radius for $\boldsymbol{A}^S$ similarly.

Complexity of computing $\overline{\varrho(\boldsymbol{A})}$ is an open problem (as Schur stability is; see Section 11.11), and, to the best of our knowledge, complexity of computing $\underline{\varrho(\boldsymbol{A})}$ has not been investigated yet.

Clearly, we have the two following polynomially solvable subclasses:

- If $\underline{A} \geq 0$, then $\varrho(\boldsymbol{A}) = [\varrho(\underline{A}), \varrho(\overline{A})]$ (follows from the Perron–Frobenius theory).
- If $\boldsymbol{A}$ is diagonal, then $\varrho(\boldsymbol{A}) = [\max_i \operatorname{mig}(\boldsymbol{a}_{ii}), \max_i \operatorname{mag}(\boldsymbol{a}_{ii})]$.

11.9.1 Summary

Problem	**Complexity**
Is λ eigenvalue of some $A \in \boldsymbol{A}$?	NP-hard
Is x eigenvector of some $A \in \boldsymbol{A}$?	strongly P
Is x Perron vector of nonnegative irreducible $\boldsymbol{A}$?	strongly P
Is 0 eigenvalue of some $A \in \boldsymbol{A}^S$?	NP-hard
Is x eigenvector of some $A \in \boldsymbol{A}^S$?	P
Does $\lambda_{\max}(\boldsymbol{A}^S)$ belong to a given open interval?	coNP-hard
Computing $\overline{\varrho(\boldsymbol{A})}$	?
Computing $\underline{\varrho(\boldsymbol{A})}$	?
Computing exact bounds on $\varrho(\boldsymbol{A})$ with $\boldsymbol{A}$ nonnegative	strongly P
Computing exact bounds on $\varrho(\boldsymbol{A})$ with $\boldsymbol{A}$ diagonal	strongly P

11.10 Positive definitness and semidefiniteness

We should not leave out positive definiteness and semidefiniteness. Here without the loss of the generality symmetric matrices are of the only interest. We distinguish between weak and strong definiteness.

Definition 11.31. A symmetric interval matrix $\boldsymbol{A}^S$ is weakly positive (semi)definite if some $A \in \boldsymbol{A}^S$ is positive (semi)definite.

Definition 11.32. A symmetric interval matrix $\boldsymbol{A}^S$ is strongly positive (semi)definite if every $A \in \boldsymbol{A}^S$ is positive (semi)definite.

Checking strong positive definiteness [170] and strong positive semidefiniteness [136] are both coNP-hard. Considering positive definiteness, there are some sufficient conditions that can be checked polynomially [172].

Theorem 11.33. *An interval matrix $\boldsymbol{A}^S$ is strongly positive definite if at least one of the following condition holds:*

- $\lambda_{\min}(A_c) > \varrho(A_\Delta)$,
- *A_c is positive definite and $\varrho(|A_c^{-1}|A_\Delta) < 1$.*

The second condition can be reformulated as $\boldsymbol{A}^S$ being regular and A_c positive definite. If the first condition holds with $\geq$, then $\boldsymbol{A}^S$ is strongly positive semidefinite.

In contrast to checking strong positive definiteness, weak positive definiteness can be checked in polynomial time by semidefinite programming [98]; this polynomial result holds also for a more general class of symmetric interval matrices with linear dependencies [76]. For positive semidefiniteness it need not be the case since semidefinite programming methods work only with some given accuracy.

11.10.1 Summary

Problem	**Complexity**
Is $\boldsymbol{A}^S$ strongly positive definite?	coNP-hard
Is $\boldsymbol{A}^S$ strongly positive semidefinite?	coNP-hard
Is $\boldsymbol{A}^S$ weakly positive definite?	P
Is $\boldsymbol{A}^S$ weakly positive semidefinite?	?

11.11 Stability

The last section is dedicated to an important and more practical problem – deciding a stability of a matrix. There are many types of stability. For illustration, we chose two of them – Hurwitz and Schur.

Definition 11.34. An interval matrix $\boldsymbol{A}$ is *Hurwitz stable* if every $A \in \boldsymbol{A}$ is Hurwitz stable (i.e., all eigenvalues have negative real parts).

Similarly, we define Hurwitz stability for symmetric interval matrices. Due to their relation to positive definiteness ($\boldsymbol{A}^S$ is Hurwitz stable if $-\boldsymbol{A}^S$ is positive definite), we could presume that the problem is coNP-hard [170]. The problem remains coNP-hard even if we limit the number of interval coefficients in our matrix [136].

Theorem 11.35. *Checking Hurwitz stability of $\boldsymbol{A}$ is* coNP*-hard on a class of interval matrices with intervals in the last row and column only.*

Likewise, as for checking regularity, also checking Hurwitz stability of $\boldsymbol{A}$ cannot be done by checking stability of matrices of type A_{yz} (see, e.g., [102]). On the other hand, it can be checked in this way for $\boldsymbol{A}^S$. For more discussion and historical context see [112] or [176]. As sufficient conditions we can use conditions for positive definiteness applied to $-\boldsymbol{A}$. For more sufficient conditions see, e.g., [122].

Definition 11.36. An interval matrix $\boldsymbol{A}$ is *Schur stable* if every $A \in \boldsymbol{A}$ is Schur stable (i.e., $\varrho(A) < 1$).

In a similar way, we define Schur stability for symmetric interval matrices. For general interval matrices, complexity of checking Schur stability is an open problem, however, for the symmetric case the problem is coNP-hard [170].

11.11.1 Summary

Problem	**Complexity**
Is $\boldsymbol{A}$ Hurwitz stable?	coNP-hard
Is $\boldsymbol{A}^S$ Hurwitz stable?	coNP-hard
Is $\boldsymbol{A}$ Schur stable?	?
Is $\boldsymbol{A}^S$ Schur stable?	coNP-hard

11.12 Further topics

We conclude the section about complexity with three particular problems:

- *Matrix power.* For an interval matrix $\boldsymbol{A}$ computing the exact bounds on $\boldsymbol{A}^2$ is strongly polynomial (just by evaluating by interval arithmetic), but computing the cube $\boldsymbol{A}^3$ turns out to be NP-hard [109].
- *Matrix norm.* Computing the range of $\|A\|$ when $A \in \boldsymbol{A}$ is a trivial task for vector ℓ_p-norms applied on matrices (including Frobenius norm or maximum norm) or for induced 1- and ∞-norms. On the other hand, determining the largest value of the spectral norm $\|A\|_2$ (the largest singular value) subject to $A \in \boldsymbol{A}$ is NP-hard [136].
- *Membership in matrix classes.* Based on Chapter 4 we can state the following results. Checking whether a matrix is nonnegative invertible, strictly diagonally dominant, Z-matrix, M-matrix or H-matrix can be done in strongly polynomial

time. Checking whether a matrix $\boldsymbol{A}$ is a P-matrix is coNP-complete [29]. Strong regularity is according to Theorem 4.33 equivalent to checking whether $A_c^{-1}\boldsymbol{A}$ is an H-matrix. Therefore, it is strongly polynomial.

11.12.1 Summary

Problem	**Complexity**
Compute $\boldsymbol{A}^2$	strongly P
Compute $\boldsymbol{A}^3$	NP-hard
Compute $\|\boldsymbol{A}\|_1$	strongly P
Compute $\|\boldsymbol{A}\|_\infty$	strongly P
Compute $\|\boldsymbol{A}\|_F$	strongly P
Compute $\|\boldsymbol{A}\|_2$	NP-hard
Is $\boldsymbol{A}$ a Z-matrix?	strongly P
Is $\boldsymbol{A}$ an M-matrix?	strongly P
Is $\boldsymbol{A}$ an H-matrix?	strongly P
Is $\boldsymbol{A}$ an strictly diagonally dominant?	strongly P
Is $\boldsymbol{A}$ a P-matrix?	coNP-complete
Is $\boldsymbol{A}$ strongly regular?	strongly P

12 LIME^2: interval toolbox

- History of LIME
- Features and goals
- Structure and packages
- Installation and use

In the last chapter we introduce our interval toolbox LIME (Library of Interval MEthods). It is not a direct part of this work, however it is strongly connected to it. Most of the methods mentioned in the previous chapters are implemented in LIME and the implementations are used for comparison of the methods. In this brief chapter we describe its history, background, goals and purpose. The overall structure and selected methods are discussed. At the end we mention installation, use and extension of LIME.

12.1 History

During our research there was a need to compare various algorithms solving a given task (e.g., computing determinants of interval matrices, computing enclosures of interval linear systems, etc.). LIME occurred as a by-product to keep all the implemented methods at one place.

LIME (it could be also called LIME^1) was originally a part of the master's thesis [82]. It was implemented for Matlab and Intlab [188]. It mostly contained methods related to solving square and overdetermined interval linear systems.

After Intlab became commercial, LIME was moved under Octave and its Interval package by Oliver Heimlich [62]. Since then new packages of LIME have appeared and methods have been rewritten many times with effort to make the code more efficient and clear. We sometimes refer to it as LIME^2. The last LIME logo is in Figure 12.1.

12.2 Features and goals

Most of the methods tested in this work are implemented in LIME. Many additional useful methods are also implemented. LIME has also instruments to produce graphical outputs. All graphical outputs are produced by LIME or Octave (however, most of

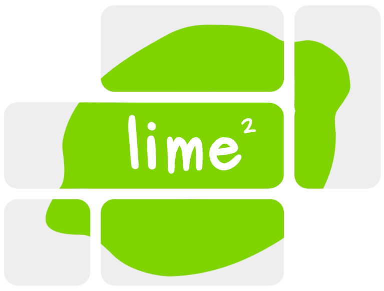


Figure 12.1: LIME^2 logo.

them were stored in an .svg file and further enhanced in Inkscape).

There are several goals of LIME:

- It is free for noncommercial use.
- It contains various methods solving one problem.
- The methods should be easy to use.
- It should be easy to develop and add new parts.
- It should be easily usable for interval educational purposes.
- The code should be clear and extensively documented (input and output parameters are described, implementation details are described, theorems on which the algorithms are based are cited, history of changes and known errors and future to do's are listed).
- It does not compete with existing interval toolboxes since their purpose is different.
- Packages are accompanied with examples or at least they are prepared for easy adding of new ones.

Most of the code is written solely by the author of this work. Some functions were implemented by other people (the author of the source code is referenced in each .m file). Unfortunately, there is still a lot of work to do (testing all the possible input cases for methods, correct handling of flags, adding more verification to some methods, etc.). Nevertheless the current state of LIME should allow other users to orient themselves quickly an to easily extend existing methods and functionality. The main goal of LIME is rather to share the code and be prepared for possible extension by others.

12.2.1 Verification and errors

Since LIME is a work of a few people it still might contain errors, even though we tried hard to catch most of the flaws. Some known errors are pointed out in .Todo. section

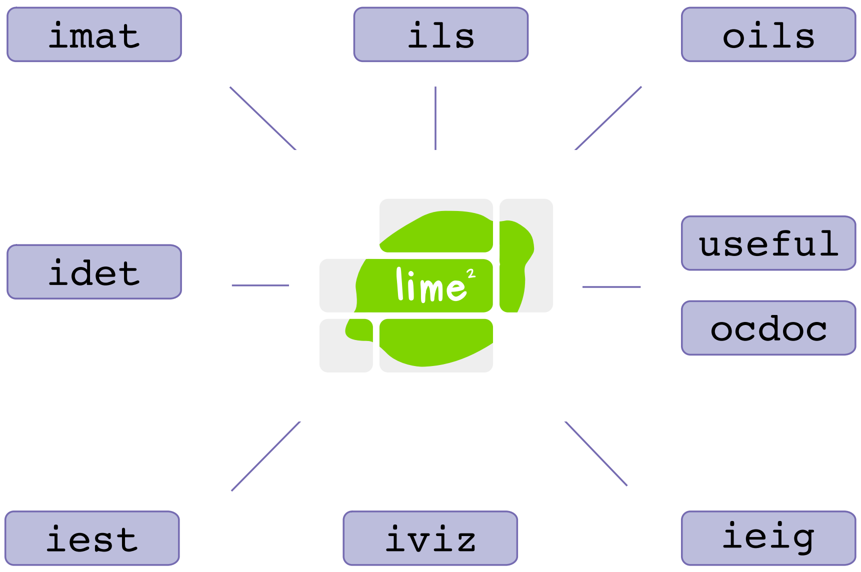


Figure 12.2: LIME structure.

of each `.m` file. There is an effort to make all the methods return verified results. In some cases it is not possible. In some cases verification is omitted to spare computational time. Such situations are documented in `.m` files if necessary. Most of the current code is implemented by the author of this work. Some functions were originally implemented by students supervised by David Hartman, Milan Hladík and Jaroslav Horáček (eigenvalues, matrix powers, interval estimations, interval determinants, etc.). Many more methods were written for Matlab and are waiting to be transfered to LIME (parametric interval systems, evaluation of polynomials, nonlinear solver, etc.)

12.3 Structure

For a better logical structure LIME is divided into several parts, we call them *packages*. They are depicted in Figure 12.2.

Here we list the packages with a brief description:

- `imat` – functions related to interval matrices,
- `ils` – methods connected to (square) interval linear systems,
- `oils` – methods connected to overdetermined interval linear systems,
- `idet` – methods for computing determinants of interval matrices,
- `iest` – interval data estimations and regressions,
- `ieig` – eigenvalues of (symmetric) interval matrices,
- `iviz` – methods for visualizations of intervals,

- `useful` – various methods that can be helpful,
- `ocdoc` – our minimalistic HTML documentation system.

Further we describe each package in bigger detail.

12.4 Packages

Each package is contained in a unique folder. It further contains three subfolders. The first is `doc` which contains the `.html` documentation of the package (every package has standalone documentation). The second folder is `test` that can contain examples corresponding to the package. For example in package `ils`, the folder `test` contains a function returning the example of interval linear system according to a given keyword. The origin of examples is referenced. The third is `develop` which contains functions under development.

12.4.1 `imat`

This package contains various methods working on interval matrices. Moreover, it contains methods for generating random matrices.

Function	**Description**
`ifcr`	full column rank test
`isregular`	regularity test
`issingular`	singularity test
`ismmatrix`	M-matrix test
`ishmatrix`	H-matrix test
`imatnorm`	various matrix norms
`imatinv`	inverse interval matrix
`vinv`	verified inverse of a real matrix
`imatpow`	power of an interval matrix

12.4.2 `ils`

Various methods connected to square interval linear systems are implemented here. Some methods work also for overdetermined interval systems (e.g., solvability and unsolvability testing, hull computation etc.).

Function	Description
`ilsenc`	general function for solving interval lin. systems
`ilsjacobienc`	enclosure based on the Jacobi method
`ilsgsenc`	enclosure based on the Gauss–Seidel method
`ilsgeenc`	enclosure based on Gaussian elimination
`ilskrawczykenc`	enclosure based on Krawczyk's method
`ilshbrenc`	the Hansen–Bliek–Rohn enclosure
`ilshullver`	verified hull
`ilshull`	unverified hull (faster)
`ige`	Gaussian elimination
`ibacksubst`	backward substitution
`ilsprecondinv`	preconditioning
`vsol`	verified solution of a real linear system
`isuns`	unsolvability test
`issolvable`	solvability test

12.4.3 `oils`

This package defines methods for overdetermined interval linear systems.

Function	Description
`oilsenc`	enclosure of an overdetermined int. lin. system
`oilshull`	same as `ilshull`
`oilsgeenc`	enclosure based on Gaussian elimination
`oilsrohnenc`	enclosure based on Rohn's method
`oilssubsqenc`	enclosure by subsquares method
`oilsmultijacenc`	the multi-Jacobi method
`oilslsqenc`	enclosure of the least squares

12.4.4 `idet`

The package is devoted to determinants of interval matrices. Some of the functions were written by Josef Matějka.

Function	Description
`idet`	main function for computing an int. determinant
`idethull`	hull of determinant
`idethad`	determinant enclosure by Hadamard's inequality
`idetcram`	determinant enclosure by Cramer's rule
`idetgauss`	determinant enclosure by Gaussian elimination
`idetgersch`	determinant enclosure by Gerschgorin discs
`idetencsym`	determinant enclosure for symmetric matrices

12.4.5 `iest`

This package covers various interval data regressions and estimations. Most of the functions in this package were implemented by Petra Pelikánová.

Function	Description
`iestlsq`	the least squares regression
`iest`	outer estimation
`iesttol`	tolerance interval regression

12.4.6 `ieig`

This package contains a few methods regarding eigenvalues. They are usually needed by other methods.

Function	Description
`eigsymdirect`	direct method for computing eigenvalues of a sym. matrix
`eigsymrohn`	fast outer enclosure of eigenvalues of a sym. matrix
`ieigbauerfike`	eigenvalues enclosure based on Bauer–Fike theorem
`igersch`	interval Gerschgorin discs
`vereigsym`	verified eigenvalues of a real sym. matrix
`l1upperb`	upper bound on the largest eigenvalue

12.4.7 `useful`

This package contains useful methods that do not fit into other packages.

Function	Description
`area`	computes a generalized volume of an interval vector
`compareenc`	compares two interval enclosures
`generateyn`	generates all Y_n vectors
`latextablesimple`	prints a LaTeXtable from data
`radfi`	uniformly random number from an interval
`randfim`	uniformly random matrix from an interval matrix
`randseln`	selects n random elements from a list

12.4.8 `iviz`

LIME contains also various methods enabling display of interval results.

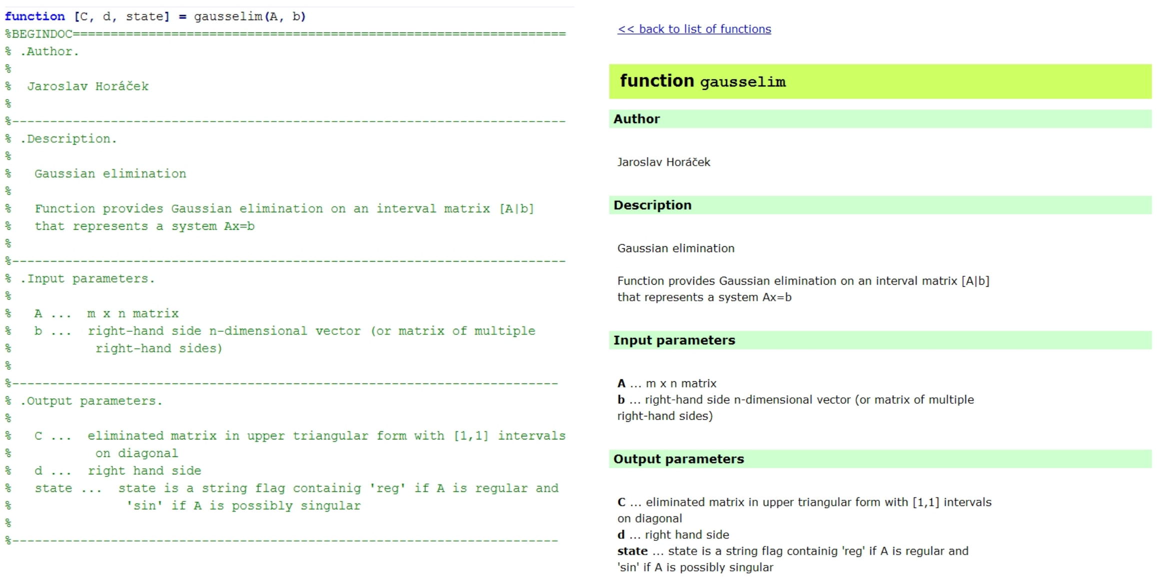
Function	Description
`plotboxes`	plotting interval boxes
`plotilssol`	plot solution set of an interval linear system

12.4.9 `ocdoc`

OcDoc is our own light-weight documentation system. To generate an `.html` documentation, go to a desired folder using the command `cd` in the Octave command line. Then simply call `ocdoc`. The function searches the current folder for `.m` files and for each such a file it generates an `.html` file containing documentation. It also generates a common `.html` index file for the whole folder. This way each package can be documented separately. To make OcDoc work it is necessary to keep the prescribed format of documentation in each .m file. A template `.m` file with the documentation structure is attached in the `doc` package. An example of an automatically generated documentation can be seen in Figure 12.3.

Even though, it is demanding to fully keep the structure of the file, it is favorable to do so, at least for the sake of future users. The documentation comments contain the following blocks:

- `.Author.` – name of the author(s),
- `.Input parameters.` – description of input parameters,
- `.Output parameters.` – description of output parameters,
- `.Implementation details.` – explanation of tricky details with references to papers and literature,
- `.History.` – history of changes,
- `.Todo.` – known mistakes, errors, future to do's and improvements.



```
function [C, d, state] = gausselim(A, b)
%BEGINDOC=================================================================
% .Author.
%
%  Jaroslav Horáček
%
%-------------------------------------------------------------------------
% .Description.
%
%   Gaussian elimination
%
%   Function provides Gaussian elimination on an interval matrix [A|b]
%   that represents a system Ax=b
%
%-------------------------------------------------------------------------
% .Input parameters.
%
%   A ...  m x n matrix
%   b ...  right-hand side n-dimensional vector (or matrix of multiple
%           right-hand sides)
%
%------------------------------------------------------------------------
% .Output parameters.
%
%   C ...  eliminated matrix in upper triangular form with [1,1] intervals
%           on diagonal
%   d ...  right hand side
%   state ...  state is a string flag containig 'reg' if A is regular and
%               'sin' if A is possibly singular
%
%------------------------------------------------------------------------
```

<< back to list of functions

function `gausselim`

Author

Jaroslav Horáček

Description

Gaussian elimination

Function provides Gaussian elimination on an interval matrix [A|b]
that represents a system Ax=b

Input parameters

A ... m x n matrix
b ... right-hand side n-dimensional vector (or matrix of multiple
right-hand sides)

Output parameters

C ... eliminated matrix in upper triangular form with [1,1] intervals
on diagonal
d ... right hand side
state ... state is a string flag containig 'reg' if A is regular and
'sin' if A is possibly singular

Figure 12.3: Example of OcDoc text documentation structure within an `.m` file (left) and the resulting `.html` page (right).

12.5 Access, installation, use

12.5.1 Installation

LIME is accessible online at

`https://kam.mff.cuni.cz/~horacek/lime`[1]

To install it run `install.m` file. The only action it executes is adding the LIME directories into Octave PATH variable.

To make LIME work, one needs to have Octave Interval package installed. Detailed information, how to do that is provided at

`https://octave.sourceforge.io/interval/package_doc/index.html`[2]

12.5.2 User modifications

For the sake of modifying the existing code of LIME we give some recommendations that can contribute to preserving the overall structure of the toolbox.

To get a basic idea how each file is structured, there are prepared empty template files in the folder `ocdoc`.

LIME is divided into packages. Each package has its distinct functionality. Although, for some functions it might be arguable where they should be placed. If new functions are of common special functionality are designed, then a new package

[1]Accessed on March 22nd, 2019.
[2]Accessed on March 22nd, 2019.

(folder) is recommended to be created. Remember, that the OcDoc documentation tool creates an `.html` documentation for each file in a given folder.

Functions have a simple naming convention – the name is composed using lower case abbreviations to describe its functionality. No upper case, no dashes, no hyphens are used. The first part of a function name usually consists of the name of the package the function comes from – `ilsgeenc` (comes from the package `ils`), `imatinv` (comes from the package `imat`). Then the rest of the function name is composed of functionality specification (e.g., `norm` for a function computing a norm, `hull` for a function computing the hull) – `imatnorm`, `idethull`. Also the specification of implementation of a function is usually added – `ilsjacobienc`, `ilshbrenc`.

We remind again, that in order to make the automatic documentation work, the structure of the file must be kept.

Here we add some more recommendations:

- Methods do not always succeed. To indicate the state of the result we use the output variable `state`. We use short (mostly three-letter) string flags. The most common flags are `'vec'` – a finite vector or scalar is returned, `'sin'` – possible singularity occurred, `'zer'` – zero division occurred or pivot contains zero, `'inf'` – infinite result returned, `'exc'` – maximum number of iterations exceeded, `'empty'` – empty solution returned. For flags that can be returned by a given method see its `.m` file or documentation.
- Methods do not always return verified results. We use the output variable `ver` to indicate verified result. The value 1 means verification, the value 0 means the opposite.
- To indicate that a variable is an interval variable we use the prefix `i`. Hence `ix` (`iA`) is an interval vector (matrix).
- Most of the interval methods work only for interval input, when they are to be used for a real input, it must be intervalized first. It is a responsibility of a user to properly handle that (simply calling the Octave Interval function `infsupdec` on a real input might not be sufficient).
- If a method is implemented that has similar functionality to some existing method, first see its input and return parameters to make it consistent for other methods that can possibly use this method too.

Here is an example of a function definition satisfying the above recommendations:

```
[ix, state, it] = ilskrawczykenc(iA, ib, e, maxit, ioldx)
```

12.5.3 LIME uder Matlab

The new version LIME^2 has not been migrated back to Matlab. Even though, we tried to keep the things similar. In LIME some Intlab names of interval functions can be used. Such mirror functions are contained in folder `octave`. Some of the methods cause problems and cannot be simply renamed (it is mainly the case of `intval`, `isnan`).

For the case when there is a need to migrate LIME methods to Matlab and Intlab we could provide a few hints:

1. First it may be favorable to rename or delete the folder `octave`.
2. Some methods that have different names in Intlab and Octave may cause problems (for the list of such methods see Octave WIKI (`https://wiki.octave.org/Interval_package`).[3]
3. We use Octave way of constructing intervals with flavors (`infsupdec` function), earlier versions of Intlab did not have such a method. However, if we can do without flavors, we can replace it with `infsup`.
4. There are some Matlab functions that are not currently implemented in Octave yet[4].

[3]Accessed on March 22nd, 2019.

[4]See https://www.gnu.org/software/octave/missing.html, Accessed on March 22nd, 2019.

13 Additional materials

- ► List of author's publications
- ► Supervised students

13.1 List of author's publications

13.1.1 Journal papers

[83] J. Horáček and M. Hladík. Computing enclosures of overdetermined interval linear systems. *Reliable Computing*, 19:143, 2013.

[86] J. Horáček, M. Hladík, and J. Matějka. Determinants of interval matrices. *Electronic Journal of Linear Algebra*, 33:99–112, 2018.

[89] J. Horáček, V. Koucký, and M. Hladík. Novel approach to computerized breath detection in lung function diagnostics. *Computers in Biology and Medicine*, 101:1–6, 2018.

13.1.2 Conference and workshop papers

[80] M. Hladík and J. Horáček. Interval linear programming techniques in constraint programming and global optimization. In M. Ceberio and V. Kreinovich, editors, *Constraint Programming and Decision Making*, pages 47–59. Springer, 2014.

[81] M. Hladík and J. Horáček. A shaving method for interval linear systems of equations. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, editors, *Parallel Processing and Applied Mathematics*, volume 8385 of *Lecture Notes in Computer Science*, pages 573–581. Springer, 2014.

[84] J. Horáček and M. Hladík. Subsquares approach – A simple scheme for solving overdetermined interval linear systems. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, editors, *Parallel Processing and Applied Mathematics*, volume 8385 of *Lecture Notes in Computer Science*, pages 613–622. Springer, Springer, 2014.

[85] J. Horáček, M. Hladík, and M. Černý. Interval linear algebra and computational complexity. In N. Bebiano, editor, *International Conference on Matrix Analysis and its Applications*, pages 37–66. Springer, 2015.

[87] J. Horáček, J. Horáček, and M. Hladík. Detecting unsolvability of interval linear systems. In M. Martel, N. Damouche, and J. A. D. Sandretto, editors, *TNC'18. Trusted Numerical Computations*, volume 8 of *Kalpa Publications in Computing*, pages 54–69. EasyChair, 2018.

[88] J. Horáček, V. Koucký, and M. Hladík. Children lung function diagnostics – New methods for handling of clinical data. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 174–176. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

13.1.3 Unpublished work

[90] J. Horáček, V. Koucký, and M. Hladík. Contribution of interval linear algebra to the ongoing discussions on multiple breath washout test. *arXiv preprint arXiv:1902.09026*, 2019.

13.2 Defended students

In this section defended Bachelor's theses supervised by the author of this work are listed. The joint research collaboration influenced the further research [90] and created a starting point for extension and further publication [86].

[125] M. Mečiar. Visualisation of interval data (in czech). Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2015.

[127] M. Milota. Psychologically-plausible and connectionism-friendly implementation of long-term memory (in czech). Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2016.

[123] J. Matějka. Determinants of interval matrices (in czech). Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2017.

[148] P. Pelikánová. Estimating data with use of interval analysis (in czech). Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2017.

[210] A. Szabo. Application of branch and bound approach to parametric interval linear systems (in czech). Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2018.

Bibliography

[1] M. Adm and J. Garloff. Intervals of totally nonnegative matrices. *Linear Algebra and its Applications*, 439(12):3796–3806, 2013.

[2] M. Adm and J. Garloff. Intervals of special sign regular matrices. *Linear and Multilinear Algebra*, 64(7):1424–1444, 2016.

[3] G. Alefeld and J. Herzberger. *Introduction to Interval Computations*. Computer Science and Applied Mathematics. Academic Press, New York, 1983.

[4] G. Alefeld, V. Kreinovich, and G. Mayer. On the solution sets of particular classes of linear interval systems. *Journal of Computational and Applied Mathematics*, 152(1-2):1–15, 2003.

[5] G. Alefeld and G. Mayer. Interval analysis: Theory and applications. *Journal of Computational and Applied athematics*, 121(1-2):421–464, 2000.

[6] E. Althaus, B. Becker, D. Dumitriu, and S. Kupferschmid. Integration of an LP solver into interval constraint propagation. In W. Wang, X. Zhu, and D.-Z. Du, editors, *International Conference on Combinatorial Optimization and Applications*, pages 343–356. Springer, 2011.

[7] M. Araki. Application of M-matrices to the stability problems of composite dynamical systems. *Journal of Mathematical Analysis and Applications*, 52(2):309–321, 1975.

[8] I. Araya, G. Trombettoni, and B. Neveu. A contractor based on convex interval Taylor. In N. Beldiceanu, N. Jussien, and É. Pinson, editors, *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 1–16. Springer, 2012.

[9] S. Arora and B. Barak. *Computational complexity: A modern approach*. Cambridge University Press, 2009.

[10] Y.-W. Bai, C.-L. Tsai, and S.-C. Wu. Design of a breath detection system with multiple remotely enhanced hand-computer interaction devices. In *2012 IEEE 16th International Symposium on Consumer Electronics*, pages 1–5. IEEE, 2012.

[11] I. Bar-On, B. Codenotti, and M. Leoncini. Checking robust nonsingularity of tridiagonal matrices in linear time. *BIT Numerical Mathematics*, 36(2):206–220, 1996.

[12] W. Barth and E. Nuding. Optimale Lösung von Intervallgleichungssystemen. *Computing*, 12:117–125, 1974.

[13] J. Bates, G. Schmalisch, D. Filbrun, and J. Stocks. Tidal breath analysis for infant pulmonary function testing. ERS/ATS task force on standards for infant respiratory function testing. European Respiratory Society/American Thoracic Society. *European Respiratory Journal*, 16(6):1180–1192, 2000.

[14] O. Beaumont. Solving interval linear systems with linear programming techniques. *Linear Algebra and its Applications*, 281(1-3):293–309, 1998.

[15] O. Beaumont. An algorithm for symmetric interval eigenvalue problem. Technical Report IRISA-PI-00-1314, Institut de recherche en informatique et systèmes aléatoires, Rennes, France, 2000.

[16] H. Beeck. *Linear programming with inexact data.* Technische Universität München. Institut für Statistik und Unternehmensforschung, 1978.

[17] F. Benhamou, D. A. McAllester, and P. Van Hentenryck. CLP (Intervals) Revisited. In M. Bruynooghe, editor, *Proceedings of the International Logic Programming Symposium*, pages 124–138, 1994.

[18] A. Berman, M. Neumann, and R. J. Stern. *Nonnegative matrices in dynamic systems. Volume 3 of Pure and applied mathematics.* Wiley-Interscience, 1989.

[19] A. Berman and R. J. Plemmons. *Nonnegative matrices in the mathematical sciences.* SIAM, 1994.

[20] S. Białas and J. Garloff. Intervals of P-matrices and related matrices. *Linear Algebra and its Applications*, 58:33–41, 1984.

[21] L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and real computation.* Springer Science & Business Media, 2012.

[22] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[23] J. Brunner, G. Wolff, H. Langenstein, and G. Cumming. Reliable detection of inspiration and expiration by computer. *Journal of Clinical Monitoring and Computing*, 1(4):221–226, 1985.

[24] M. Černý, J. Antoch, and M. Hladík. On the possibilistic approach to linear regression models involving uncertain, indeterminate or interval data. *Information Sciences*, 244:26–47, 2013.

[25] G. Chabert and L. Jaulin. Contractor programming. *Artificial Intelligence*, 173:1079–1100, 2009.

[26] L. Chen, A. Miné, J. Wang, and P. Cousot. Interval polyhedra: An abstract domain to infer interval linear relationships. In J. Palsberg and Z. Su, editors, *International Static Analysis Symposium*, pages 309–325. Springer, 2009.

[27] K. P. Cohen, W. M. Ladd, D. M. Beams, W. S. Sheers, R. G. Radwin, W. J. Tompkins, and J. G. Webster. Comparison of impedance and inductance ventilation sensors on adults during breathing, motion, and simulated airway obstruction. *IEEE Transactions on Biomedical Engineering*, 44(7):555–566, 1997.

[28] J. E. Cotes, D. J. Chinn, and M. R. Miller. *Lung function: Physiology, measurement and application in medicine.* John Wiley & Sons, 2009.

[29] G. E. Coxson. The P-matrix problem is coNP-complete. *Mathematical Programming*, 64(1-3):173–178, 1994.

[30] G. E. Coxson. Computing exact bounds on elements of an inverse interval matrix is NP-hard. *Reliable Computing*, 5(2):137–142, 1999.

[31] X. Daoyi. Simple criteria for stability of interval matrices. *International Journal of Control*, 41(1):289–295, 1985.

[32] Y. David, W. W. Von Maltzahn, M. R. Neuman, and J. D. Bronzino. *Clinical engineering.* CRC Press, 2003.

[33] J. C. Davies, S. Cunningham, E. W. Alton, and J. Innes. Lung clearance index in CF: A sensitive marker of lung disease severity. *Thorax*, 63(2):96–97, 2008.

[34] F. d. A. T. de Carvalho, E. d. A. L. Neto, and C. P. Tenorio. A new method to fit a linear regression model for interval-valued data. In *Annual Conference on Artificial Intelligence*, pages 295–306. Springer, 2004.

[35] J. A. dit Sandretto and M. Hladík. Solving over-constrained systems of non-linear interval equations – And its robotic application. *Applied Mathematics and Computation*, 313:180–195, 2017.

[36] P. S. Dwyer. *Linear computations.* Wiley, 1951.

[37] J. Edmonds. Systems of distinct representatives and linear algebra. *Journal of Research of the National Bureau of Standards. Section B*, 71(4):241–245, 1967.

[38] R. Farhadsefat, T. Lotfi, and J. Rohn. A note on regularity and positive definiteness of interval matrices. *Open Mathematics*, 10(1):322–328, 2012.

[39] M. Fiedler. *Special matrices and their applications in numerical mathematics.* Courier Corporation, 2008.

[40] M. Fiedler, J. Nedoma, J. Ramik, J. Rohn, and K. Zimmermann. *Linear optimization problems with inexact data.* Springer, 2006.

[41] M. Fiedler and V. Pták. On matrices with non-positive off-diagonal elements and positive principal minors. *Czechoslovak Mathematical Journal*, 12(3):382–400, 1962.

[42] A. Frommer. A feasibility result for interval Gaussian elimination relying on graph structure. In G. Alefeld, J. Rohn, S. Rump, and T. Yamamoto, editors, *Symbolic Algebraic Methods and Verification Methods*, pages 79–86. Springer, 2001.

[43] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. W. H. Freeman New York, 2002.

[44] J. Garloff. Totally nonnegative interval matrices. In *Interval Mathematics 1980*, pages 317–327. Elsevier, 1980.

[45] J. Garloff. Criteria for sign regularity of sets of matrices. *Linear Algebra and its Applications*, 44:153–160, 1982.

[46] J. Garloff. Convergent bounds for the range of multivariate polynomials. In K. Nickel, editor, *Interval Mathematics 1985*, pages 37–56. Springer Lecture Notes in Computer Science, 1986.

[47] J. Garloff. Solution of linear equations having a Toeplitz interval matrix as coefficient matrix. *Opuscula Mathematica*, 2:33–45, 1986.

[48] J. Garloff. Block methods for the solution of linear interval equations. *SIAM Journal on Matrix Analysis and Applications*, 11(1):89–106, 1990.

[49] J. Garloff. Interval Gaussian elimination with pivot tightening. *SIAM Journal on Matrix Analysis and Applications*, 30(4):1761–1772, 2009.

[50] J. Garloff, M. Adm, and J. Titi. A survey of classes of matrices possessing the interval property and related properties. *Reliable Computing*, 22:1–10, 2016.

[51] J. Garloff et al. *Interval mathematics: A bibliography.* Institut für Angewandte Mathematik, 1982.

[52] J. Garloff and K.-P. Schwierz. A bibliography on interval mathematics. *Journal of Computational and Applied Mathematics*, 6(1):67–79, 1980.

[53] W. Gerlach. Zur Lösung linearer Ungleichungssysteme bei Störung der rechten Seite und der Koeffizientenmatrix. *Mathematische Operationsforschung und Statistik. Serie Optimization*, 12(1):41–43, 1981.

[54] A. Goldsztejn and F. Goualard. Box consistency through adaptive shaving. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 2049–2054. ACM, 2010.

[55] N. Govindarajan and O. Prakash. Breath detection algorithm in digital computers. *Journal of Clinical Monitoring and Computing*, 7(1):59–64, 1990.

[56] K. Green, F. F. Buchvald, J. K. Marthin, B. Hanel, P. M. Gustafsson, and K. G. Nielsen. Ventilation inhomogeneity in children with primary ciliary dyskinesia. *Thorax*, 67(1), 2011.

[57] E. Hansen. Bounding the solution of interval linear equations. *SIAM Journal on Numerical Analysis*, 29(5):1493–1503, 1992.

[58] E. Hansen and R. Smith. Interval arithmetic in matrix computations, part II. *SIAM Journal on Numerical Analysis*, 4(1):1–9, 1967.

[59] E. Hansen and G. W. Walster. *Global optimization using interval analysis.* Marcel Dekker, New York, second edition, 2004.

[60] E. Hansen and G. W. Walster. Solving overdetermined systems of interval linear equations. *Reliable computing*, 12(3):239–243, 2006.

[61] G. I. Hargreaves. Interval analysis in Matlab. In *Numerical Analysis Reports*, volume 416, pages 1–49. Manchester Institute for Mathematical Sciences, University of Manchester, 2002.

[62] O. Heimlich. GNU Octave Interval Package. Version 3.2.0, 2018.

[63] J. A. Heinen. Sufficient conditions for stability of interval matrices. *International Journal of Control*, 39(6):1323–1328, 1984.

[64] D. Hertz. The extreme eigenvalues and stability of real symmetric interval matrices. *IEEE Transactions on Automatic Control*, 37(4):532–535, 1992.

[65] T. J. Hickey and D. K. Wittenberg. Rigorous modeling of hybrid systems using interval arithmetic constraints. In R. Alur and G. J. Pappas, editors, *International Workshop on Hybrid Systems: Computation and Control*, pages 402–416. Springer, 2004.

[66] C. Hirt, S. Claessens, T. Fecher, M. Kuhn, R. Pail, and M. Rexer. New ultrahigh-resolution picture of Earth's gravity field. *Geophysical Research Letters*, 40(16):4279–4283, 2013.

[67] M. Hladík. Enclosures for the solution set of parametric interval linear systems. *International Journal of Applied Mathematics and Computer Science*, 3(22):561–574, 2012.

[68] M. Hladík. Interval linear programming: A survey. *Linear programming-new frontiers in theory and applications*, pages 85–120, 2012.

[69] M. Hladík. Bounds on eigenvalues of real and complex interval matrices. *Applied Mathematics and Computation*, 219(10):5584–5591, 2013.

[70] M. Hladík. Weak and strong solvability of interval linear systems of equations and inequalities. *Linear Algebra and its Applications*, 438(11):4156–4165, 2013.

[71] M. Hladík. New operator and method for solving real preconditioned interval linear equations. *SIAM Journal on Numerical Analysis*, 52(1):194–206, 2014.

[72] M. Hladík. Complexity issues for the symmetric interval eigenvalue problem. *Open Mathematics*, 13(1):157–164, 2015.

[73] M. Hladík. On relation between P-matrices and regularity of interval matrices. In N. Bebiano, editor, *International Conference on Matrix Analysis and its Applications*, pages 27–35. Springer, 2015.

[74] M. Hladík. Optimal preconditioning for the interval parametric Gauss–Seidel method. In *International Symposium on Scientific Computing, Computer Arithmetic, and Validated Numerics*, pages 116–125. Springer, 2015.

[75] M. Hladík. An overview of polynomially computable characteristics of special interval matrices. *arXiv preprint arXiv:1711.08732*, 2017.

[76] M. Hladík. Positive semidefiniteness and positive definiteness of a linear parametric interval matrix. In M. Ceberio and V. Kreinovich, editors, *Constraint Programming and Decision Making: Theory and Applications*, pages 77–88. Springer, 2018.

[77] M. Hladík and M. Černý. Interval regression by tolerance analysis approach. *Fuzzy Sets and Systems*, 193:85–107, 2012.

[78] M. Hladík, D. Daney, and E. Tsigaridas. Bounds on real eigenvalues and singular values of interval matrices. *SIAM Journal on Matrix Analysis and Applications*, 31(4):2116–2129, 2010.

[79] M. Hladík, D. Daney, and E. Tsigaridas. A filtering method for the interval eigenvalue problem. *Applied Mathematics and Computation*, 217(12):5236–5242, 2011.

[80] M. Hladík and J. Horáček. Interval linear programming techniques in constraint programming and global optimization. In M. Ceberio and V. Kreinovich, editors, *Constraint Programming and Decision Making*, pages 47–59. Springer, 2014.

[81] M. Hladík and J. Horáček. A shaving method for interval linear systems of equations. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, editors, *Parallel Processing and Applied Mathematics*, volume 8385 of *Lecture Notes in Computer Science*, pages 573–581. Springer, 2014.

[82] J. Horáček. Overdetermined systems of interval linear equations (in Czech). Master's thesis, Charles University, Faculty of Mathematics and Physics, 2011.

[83] J. Horáček and M. Hladík. Computing enclosures of overdetermined interval linear systems. *Reliable Computing*, 19:143, 2013.

[84] J. Horáček and M. Hladík. Subsquares approach – A simple scheme for solving overdetermined interval linear systems. In R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, editors, *Parallel Processing and Applied Mathematics*, volume 8385 of *Lecture Notes in Computer Science*, pages 613–622. Springer, Springer, 2014.

[85] J. Horáček, M. Hladík, and M. Černý. Interval linear algebra and computational complexity. In N. Bebiano, editor, *International Conference on Matrix Analysis and its Applications*, pages 37–66. Springer, 2015.

[86] J. Horáček, M. Hladík, and J. Matějka. Determinants of interval matrices. *Electronic Journal of Linear Algebra*, 33:99–112, 2018.

[87] J. Horáček, J. Horáček, and M. Hladík. Detecting unsolvability of interval linear systems. In M. Martel, N. Damouche, and J. A. D. Sandretto, editors, *TNC'18. Trusted Numerical Computations*, volume 8 of *Kalpa Publications in Computing*, pages 54–69. EasyChair, 2018.

[88] J. Horáček, V. Koucký, and M. Hladík. Children lung function diagnostics – New methods for handling of clinical data. In *Proceedings of the 9th EAI International Conference on Bio-inspired Information and Communications Technologies (formerly BIONETICS)*, pages 174–176. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2016.

[89] J. Horáček, V. Koucký, and M. Hladík. Novel approach to computerized breath detection in lung function diagnostics. *Computers in Biology and Medicine*, 101:1–6, 2018.

[90] J. Horáček, V. Koucký, and M. Hladík. Contribution of interval linear algebra to the ongoing discussions on multiple breath washout test. *arXiv preprint arXiv:1902.09026*, 2019.

[91] R. A. Horn and C. R. Johnson. *Matrix analysis.* Cambridge University Press, 1990.

[92] S. Ilog. Revising hull and box consistency. In D. D. Schreye, editor, *Logic Programming: Proceedings of the 1999 International Conference on Logic Programming*, page 230. MIT press, 1999.

[93] C. Jansson. Interval linear systems with symmetric matrices, skew-symmetric matrices and dependencies in the right hand side. *Computing*, 46(3):265–274, 1991.

[94] C. Jansson. Calculation of exact bounds for the solution set of linear interval systems. *Linear Algebra and Its Applications*, 251:321–340, 1997.

[95] C. Jansson. Rigorous lower and upper bounds in linear programming. *SIAM Journal on Optimization*, 14(3):914–935, 2004.

[96] C. Jansson and J. Rohn. An algorithm for checking regularity of interval matrices. *SIAM Journal on Matrix Analysis and Applications*, 20(3):756–776, 1999.

[97] L. Jaulin. Reliable minimax parameter estimation. *Reliable Computing*, 7:231–246, 2001.

[98] L. Jaulin and D. Henrion. Contracting optimally an interval matrix without loosing any positive semi-definite matrix is a tractable problem. *Reliable Computing*, 11(1):1–17, 2005.

[99] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter. *Applied interval analysis: With examples in parameter and state estimation, robust control and robotics.* Springer-Verlag London, 2001.

[100] L. Jaulin and E. Walter. Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica*, 29(4):1053–1064, 1993.

[101] R. Jensen, K. Green, P. Gustafsson, P. Latzin, J. Pittman, F. Ratjen, P. Robinson, F. Singer, S. Stanojevic, and S. Yammine. Standard operating procedure: Multiple breath nitrogen washout. *EcoMedics AG, Duernten, Switzerland*, 2013.

[102] W. C. Karl, J. P. Greschak, and G. C. Verghese. Comments on 'A necessary and sufficient condition for the stability of interval matrices'. *International Journal of Control*, 39(4):849–851, 1984.

[103] R. B. Kearfott. Preconditioners for the interval Gauss–Seidel method. *SIAM Journal on Numerical Analysis*, 27(3):804–822, 1990.

[104] R. B. Kearfott. Interval computations: Introduction, uses, and resources. *Euromath Bulletin*, 2(1):95–112, 1996.

[105] R. B. Kearfott. *Rigorous global search: Continuous problems.* Springer US, 1996.

[106] C. Keil. Lurupa-rigorous error bounds in linear programming. In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2006.

[107] L. V. Kolev. Outer interval solution of the eigenvalue problem under general form parametric dependencies. *Reliable Computing*, 12(2):121–140, 2006.

[108] L. V. Kolev. Eigenvalue range determination for interval and parametric matrices. *International Journal of Circuit Theory and Applications*, 38(10):1027–1061, 2010.

[109] O. Kosheleva, V. Kreinovich, G. Mayer, and H. T. Nguyen. Computing the cube of an interval matrix is NP-hard. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 1449–1453. ACM, 2005.

[110] R. Krawczyk. Newton-algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken. *Computing*, 4(3):187–201, 1969.

[111] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl. *Computational complexity and feasibility of data processing and interval computations.* Kluwer, 1998.

[112] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl. *Computational Complexity and Feasibility of Data Processing and Interval Computations.* Kluwer, Dordrecht, 1998.

[113] B. J. Kubica. Interval software, libraries and standards. In *Interval Methods for Solving Nonlinear Constraint Satisfaction, Optimization and Similar Problems*, pages 91–99. Springer, 2019.

[114] U. W. Kulisch. Complete interval arithmetic and its implementation on the computer. In A. Cuyt, W. Krämer, W. Luther, and P. Markstein, editors, *Numerical Validation in Current Hardware Architectures*, pages 7–26. Springer, 2009.

[115] U. W. Kulisch and W. L. Miranker. *Computer arithmetic in theory and practice.* Academic Press, 2014.

[116] L. Kupriyanova. Inner estimation of the united solution set of interval linear algebraic system. *Reliable Computing*, 1(1):15–31, 1995.

[117] J. R. Kuttler. A fourth-order finite-difference approximation for the fixed membrane eigenproblem. *Mathematics of Computation*, 25(114):237–256, 1971.

[118] 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.

[119] H. Leng and Z. He. Computing eigenvalue bounds of structures with uncertain-but-non-random parameters by a method based on perturbation theory. *Numerical Methods in Biomedical Engineering*, 23(11):973–982, 2007.

[120] P. Leonard, N. R. Grubb, P. S. Addison, D. Clifton, and J. N. Watson. An algorithm for the detection of individual breaths from the pulse oximeter waveform. *Journal of Clinical Monitoring and Computing*, 18(5-6):309–312, 2004.

[121] K. A. Macleod, A. R. Horsley, N. J. Bell, A. P. Greening, J. A. Innes, and S. Cunningham. Ventilation heterogeneity in children with well controlled asthma with normal spirometry indicates residual airways disease. *Thorax*, 64(1):33–37, 2009.

[122] M. Mansour. Robust stability of interval matrices. In *Proceedings of the 28th IEEE Conference on Decision and Control*, volume 1, pages 46 –51, Tampa, Florida, 1989.

[123] J. Matějka. Determinants of interval matrices (in czech). Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2017.

[124] G. Mayer. A unified approach to enclosure methods for eigenpairs. *Zeitschrift für Angewandte Mathematik und Mechanik*, 74(2):115–128, 1994.

[125] M. Mečiar. Visualisation of interval data (in czech). Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2015.

[126] C. D. Meyer. *Matrix analysis and applied linear algebra.* SIAM, 2000.

[127] M. Milota. Psychologically-plausible and connectionism-friendly implementation of long-term memory (in czech). Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2016.

[128] S. Miyajima, T. Ogita, and S. Oishi. Fast verification for respective eigenvalues of symmetric matrix. In V. G. Ganzha, E. W. Mayr, and E. V. Vorozhtsov, editors, *International Workshop on Computer Algebra in Scientific Computing*, pages 306–317. Springer, 2005.

[129] S. Miyajima, T. Ogita, S. M. Rump, and S. Oishi. Fast verification for all eigenpairs in symmetric positive definite generalized eigenvalue problems. *Reliable Computing*, 14(1):24–25, 2010.

[130] R. E. Moore. *Interval arithmetic and automatic error analysis in digital computing.* PhD thesis, Department of Mathematics, Stanford University, 1962.

[131] R. E. Moore. *Interval analysis.* Prentice-Hall, 1966.

[132] R. E. Moore. *Methods and applications of interval analysis.* SIAM, 1979.

[133] R. E. Moore, R. B. Kearfott, and M. J. Cloud. *Introduction to Interval Analysis.* SIAM, 2009.

[134] A. Narkawicz, J. Garloff, A. P. Smith, and C. A. Munoz. Bounding the range of a rational functiom over a box. *Reliable Computing*, 17:34–39, 2012.

[135] M. Nehmeier. libieeep1788: A C++ implementation of the IEEE interval standard P1788. In *2014 IEEE Conference on Norbert Wiener in the 21st Century (21CW)*, pages 1–6. IEEE, 2014.

[136] A. Nemirovskii. Several NP-hard problems arising in robust stability analysis. *Mathematics of Control, Signals, and Systems*, 6(2):99–105, 1993.

[137] A. Neumaier. New techniques for the analysis of linear interval equations. *Linear Algebra and its Applications*, 58:273–325, 1984.

[138] A. Neumaier. Linear interval equations. In *Interval Mathematics 1985*, pages 109–120. Springer, 1986.

[139] A. Neumaier. *Interval methods for systems of equations.* Cambridge University Press, 1990.

[140] A. Neumaier. A simple derivation of the Hansen–Bliek–Rohn–Ning–Kearfott enclosure for linear interval equations. *Reliable Computing*, 5(2):131–136, 1999.

[141] A. Neumaier and O. Shcherbina. Safe bounds in linear and mixed-integer linear programming. *Mathematical Programming*, 99(2):283–296, 2004.

[142] C. D. Nguyen, J. Amatoury, J. C. Carberry, and D. J. Eckert. An automated and reliable method for breath detection during variable mask pressures in awake and sleeping humans. *PloS one*, 12(6):e0179030, 2017.

[143] S. Ning and R. B. Kearfott. A comparison of some methods for solving linear interval equations. *SIAM Journal on Numerical Analysis*, 34(4):1289–1305, 1997.

[144] W. Oettli and W. Prager. Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides. *Numerische Mathematik*, 6(1):405–409, 1964.

[145] T. Ogita. Accurate and verified numerical computation of the matrix determinant. *International Journal of Reliability and Safety*, 6(1-3):242–254, 2011.

[146] A. Ostrowski. Über die Determinanten mit überwiegender Hauptdiagonale. *Commentarii Mathematici Helvetici*, 10(1):69–96, 1937.

[147] C. H. Papadimitriou. *Computational complexity.* John Wiley and Sons Ltd., 2003.

[148] P. Pelikánová. Estimating data with use of interval analysis (in czech). Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2017.

[149] K. B. Petersen, M. S. Pedersen, et al. The matrix cookbook. *Technical University of Denmark*, 7(15):510, 2008.

[150] R. J. Plemmons. M-matrix characterizations. I – nonsingular M-matrices. *Linear Algebra and its Applications*, 18(2):175–188, 1977.

[151] E. Popova and W. Krämer. Visualizing parametric solution sets. *BIT Numerical Mathematics*, 48(1):95–115, 2008.

[152] E. D. Popova. On the solution of parametrised linear systems. In W. Krämer and J. W. von Gudenberg, editors, *Scientific Computing, Validated Numerics, Interval Methods*, pages 127–138. Kluwer, 2001.

[153] E. D. Popova. Strong regularity of parametric interval matrices. In I. Dimovski et al., editors, *Proceedings of 33rd Spring Conference of the Union of Bulgarian Mathematicians, Mathematics and Education in Mathematics*, 2004.

[154] E. D. Popova. Improved solution enclosures for over- and underdetermined interval linear systems. In I. Lirkov, S. Margenov, and J. Waśniewski, editors, *International Conference on Large-Scale Scientific Computing*, pages 305–312. Springer, 2005.

[155] J. D. Pryce and G. F. Corliss. Interval arithmetic with containment sets. *Computing*, 78(3):251–276, 2006.

[156] S. Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic (TOCL)*, 7(4):723–748, 2006.

[157] S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Transactions on Embedded Computing Systems (TECS)*, 6(1):8, 2007.

[158] H. Ratschek and J. Rokne. *Geometric Computations with Interval and New Robust Methods. Applications in Computer Graphics, GIS and Computational Geometry.* Horwood Publishing, Chichester, 2003.

[159] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. Part I: Introduction. Preliminaries. The geometry of semi-algebraic sets. The decision problem for the existential theory of the reals. *Journal of Symbolic Computation*, 13(3):255–299, 1992.

[160] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. Part II: The general decision problem. Preliminaries for quantifier elimination. *Journal of Symbolic Computation*, 13(3):301–327, 1992.

[161] J. Renegar. On the computational complexity and geometry of the first-order theory of the reals. Part III: Quantifier elimination. *Journal of Symbolic Computation*, 13(3):329–352, 1992.

[162] N. Revol. Introduction to the IEEE 1788-2015 standard for interval arithmetic. In A. Abate and S. Boldo, editors, *International Workshop on Numerical Software Verification*, pages 14–21. Springer, 2017.

[163] G. Rex and J. Rohn. A note on checking regularity of interval matrices. *Linear and Multilinear Algebra*, 39(3):259–262, 1995.

[164] G. Rex and J. Rohn. Sufficient conditions for regularity and singularity of interval matrices. *SIAM Journal on Matrix Analysis and Applications*, 20(2):437–445, 1998.

[165] D. Říha. Powers of interval matrices (in czech). Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2018.

[166] J. Rohn. Systems of linear interval equations. *Linear algebra and its applications*, 126:39–78, 1989.

[167] J. Rohn. Cheap and tight bounds: The recent result by E. Hansen can be made more efficient. *Interval Computations*, 1993(4):13–21, 1993.

[168] J. Rohn. Interval matrices: Singularity and real eigenvalues. *SIAM Journal on Matrix Analysis and Applications*, 14(1):82–91, 1993.

[169] J. Rohn. Inverse interval matrix. *SIAM Journal on Numerical Analysis*, 30(3):864–870, 1993.

[170] J. Rohn. Checking positive definiteness or stability of symmetric interval matrices is NP-hard. *Commentationes Mathematicae Universitatis Carolinae*, 35(4):795–797, 1994.

[171] J. Rohn. Enclosing solutions of linear interval equations is NP-hard. *Computing*, 53(3-4):365–368, 1994.

[172] J. Rohn. Positive definiteness and stability of interval matrices. *SIAM Journal on Matrix Analysis and Applications*, 15(1):175–184, 1994.

[173] J. Rohn. Checking properties of interval matrices. Technical Report 686, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 1996.

[174] J. Rohn. Enclosing solutions of overdetermined systems of linear interval equations. *Reliable Computing*, 2(2):167–171, 1996.

[175] J. Rohn. Bounds on eigenvalues of interval matrices. *Zeitschrift für Angewandte Mathematik und Mechanik*, 78(3):S1049, 1998.

[176] J. Rohn. A handbook of results on interval linear problems, 2005.

[177] J. Rohn. Perron vectors of an irreducible nonnegative interval matrix. *Linear Multilinear Algebra*, 54(6):399–404, 2006.

[178] J. Rohn. Solvability of systems of interval linear equations and inequalities. In *Linear optimization problems with inexact data*, pages 35–77. Springer, 2006.

[179] J. Rohn. Forty necessary and sufficient conditions for regularity of interval matrices: A survey. *Electronic Journal of Linear Algebra*, 18(500-512):2, 2009.

[180] J. Rohn. Explicit inverse of an interval matrix with unit midpoint. *Electronic Journal of Linear Algebra*, 22(1):8, 2011.

[181] J. Rohn. Verification of linear (in)dependence in finite precision arithmetic. *Mathematics in Computer Science*, 8(3–4):323–328, 2014.

[182] J. Rohn. An explicit enclosure of the solution set of overdetermined interval linear equations. *Reliable Computing*, 24(1):1–10, 2017.

[183] J. Rohn and R. Farhadsefat. Inverse interval matrix: A survey. *Electronic Journal of Linear Algebra*, 22(1):46, 2011.

[184] J. Rohn and V. Kreinovich. Computing exact componentwise bounds on solutions of lineary systems with interval data is NP-hard. *SIAM Journal on Matrix Analysis and Applications*, 16(2):415–420, 1995.

[185] J. Rohn and G. Rex. Interval P-matrices. *SIAM Journal on Matrix Analysis and Applications*, 17(4):1020–1024, 1996.

[186] J. Rohn and S. P. Shary. Interval matrices: Regularity generates singularity. *Linear Algebra and its Applications*, 540:149–159, 2018.

[187] R. G. Rossing, M. B. Danford, E. L. Bell, and R. Garcia. Mathematical models for the analysis of the nitrogen washout curve. Technical report, DTIC Document, 1967.

[188] S. Rump. INTLAB - INTerval LABoratory. In T. Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999. http://www.ti3.tu-harburg.de/rump/.

[189] S. Rump and E. Kaucher. Small bounds for the solution of systems of linear equations. In G. Alefeld and R. D. Grigorieff, editors, *Fundamentals of Numerical Computation (Computer-Oriented Numerical Analysis)*, pages 157–164. Springer, 1980.

[190] S. M. Rump. *Kleine Fehlerschranken bei Matrixproblemen.* PhD thesis, Universität Karlsruhe, Karlsruhe, Baden-Württemberg, Germany, 1980.

[191] S. M. Rump. Solving algebraic problems with high accuracy. In *A new approach to scientific computation*, pages 51–120. Elsevier, 1983.

[192] S. M. Rump. Validated solution of large linear systems. In *Validation Numerics*, pages 191–212. Springer, 1993.

[193] S. M. Rump. Verification methods for dense and sparse systems of equations. In J. Herzberger, editor, *Proceedings of the IMACS-GAMM International Workshop on Validated Computation*, pages 63–135. Elsevier Science, 1994.

[194] S. M. Rump. Rigorous and portable standard functions. *BIT Numerical Mathematics*, 41(3):540–562, 2001.

[195] S. M. Rump. Computer-assisted proofs and self-validating methods. In *Accuracy and Reliability in Scientific Computing*, pages 195–240. SIAM, 2005.

[196] S. M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, 2010.

[197] R. C. Sá and Y. Verbandt. Automated breath detection on long-duration signals using feedforward backpropagation artificial neural networks. *IEEE Transactions on Biomedical Engineering*, 49(10):1130–1141, 2002.

[198] M. Schmidt, B. Foitzik, R. Wauer, F. Winkler, and G. Schmalisch. Comparative investigations of algorithms for the detection of breaths in newborns with disturbed respiratory signals. *Computers and Biomedical Research*, 31(6):413–425, 1998.

[199] M. E. Sezer and D. D. Siljak. On stability of interval matrices. *IEEE Transactions on Automatic Control*, 39(2):368–371, 1994.

[200] S. P. Shary. On controlled solution set of interval algebraic systems. *Interval Computations*, 6(6), 1992.

[201] S. P. Shary. On optimal solution of interval linear equations. *SIAM Journal on Numerical Analysis*, 32(2):610–630, 1995.

[202] S. P. Shary. Algebraic solutions to interval linear equations and their applications. *Mathematical Research*, 89:224–233, 1996.

[203] S. P. Shary. A new technique in systems analysis under interval uncertainty and ambiguity. *Reliable computing*, 8(5):321–418, 2002.

[204] S. P. Shary. On full-rank interval matrices. *Numerical Analysis and Applications*, 7(3):241–254, 2014.

[205] V. Sit, M. Poulin-Costello, and W. Bergerud. *Catalogue of curves for curve fitting*. Forest Science Research Branch, Ministry of Forests, 1994.

[206] I. Skalna. *Parametric Interval Algebraic Systems*. Springer, 2018.

[207] S. Smale. Mathematical problems for the next century. *Mathematical Intelligencer*, 20:7–15, 1998.

[208] L. B. Smith. Interval arithmetic determinant evaluation and its use in testing for a Chebyshev system. *Communications of the ACM*, 12(2):89–93, 1969.

[209] T. Sunaga. Theory of interval algebra and its application to numerical analysis. *RAAG memoirs*, 2(29-46):209, 1958.

[210] A. Szabo. Application of branch and bound approach to parametric interval linear systems (in czech). Bachelor's thesis, Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic, 2018.

[211] H. Tanaka and H. Lee. Interval regression analysis by quadratic programming approach. *IEEE Transactions on Fuzzy Systems*, 6(4):473–481, 1998.

[212] M. H. Tawhai and P. J. Hunter. Multibreath washout analysis: Modelling the influence of conducting airway asymmetry. *Respiration physiology*, 127(2-3):249–258, 2001.

[213] Q.-V. Tran, S.-F. Su, C.-C. Chuang, V.-T. Nguyen, and N.-Q. Nguyen. Real-time non-contact breath detection from video using adaboost and Lucas–Kanade algorithm. In *Fuzzy Systems Association and 9th International Conference on Soft Computing and Intelligent Systems (IFSA-SCIS), 2017 Joint 17th World Congress of International*, pages 1–4. IEEE, 2017.

[214] G. Trombettoni, Y. Papegay, G. Chabert, and O. Pourtallier. A box-consistency contractor based on extremal functions. In *International Conference on Principles and Practice of Constraint Programming*, pages 491–498. Springer, 2010.

[215] W. Tucker. Validated numerics for pedestrians. In *European Congress of Mathematics*, pages 851–860. European Mathematical Society, Zürich, 2005.

[216] R. R. Uhl and F. J. Lewis. Digital computer calculation of human pulmonary mechanics using a least squares fit technique. *Computers and Biomedical Research*, 7(5):489–495, 1974.

[217] X.-H. Vu, D. Sam-Haroud, and B. Faltings. Enhancing numerical constraint propagation using multiple inclusion representations. *Annals of Mathematics and Artificial Intelligence*, 55(3-4):295, 2009.

[218] G. W. Walster and E. Hansen. Method and apparatus for solving systems of linear inequalities, September 27 2005. US Patent 6,950,844.

[219] M. Warmus. Calculus of approximations. *Bulletin de l'Academie Polonaise de Sciences*, 4(5):253–257, 1956.

[220] A. Wilson, C. Franks, and I. Freeston. Algorithms for the detection of breaths from respiratory waveform recordings of infants. *Medical and Biological Engineering and Computing*, 20(3):286–292, 1982.

[221] N. Yamamoto. A simple method for error bounds of eigenvalues of symmetric matrices. *Linear Algebra and its Applications*, 324(1-3):227–234, 2001.

[222] R. C. Young. The algebra of many-valued quantities. *Mathematische Annalen*, 104(1):260–290, 1931.