Computing Enclosures of Overdetermined Interval Linear Systems[*]

J. Horáček[†]
horacek@kam.mff.cuni.cz

M. Hladík[‡]
hladik@kam.mff.cuni.cz

Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University in Prague, Prague, Czech Republic

Abstract

This work considers special types of interval linear systems - overdetermined systems, systems consisting of more equations than variables. The solution set of an interval linear system is a collection of all solutions of all instances of an interval system. By the instance, we mean a point real system that emerges when we independently choose a real number from each interval coefficient of the interval system. Enclosing the solution set of these systems is in some ways more difficult than for square systems. This work presents various methods for computing enclosures of overdetermined interval linear systems. We would like to present them in an understandable way even for nonspecialists in the field of linear systems. The second goal is a numerical comparison of all mentioned methods on random interval linear systems regarding tightness of enclosures, computation times, and other special properties of methods.

Keywords: interval linear systems, enclosure methods, overdetermined systems
AMS subject classifications: 65G40, 65F99

1 Introduction

Real-life problems can be described by different means – by linear and nonlinear systems, by systems of difference and differential equations, etc. Nevertheless, the description often can be transformed to another one using only linear equalities (or

[*]Submitted: February 25, 2013; Revised: August 15, 2013; Accepted: November 18, 2013).

[†]Jaroslav Horáček was partially supported by the Czech Science Foundation under the contract 201/09/H057.

[‡]M. Hladík was partially supported by the grant GACR P402-13-10660S.

inequalities). To account for rounding errors or imprecise measurement of data, we can use tools of interval analysis. That is why interval linear systems are still a focus of research. There are plenty of methods for enclosing the solution set of square interval linear systems – systems in the form $\boldsymbol{A}x = \boldsymbol{b}$, where $\boldsymbol{A}$ is a square matrix (e.g., [4], [6, 7, 16]). That is because square matrices can posses some advantageous properties. They can be diagonally dominant, positive definite, M-matrices, and many more, and we know that many algorithms behave well for those cases. However, sometimes we encounter *overdetermined* systems, consisting of more equations than variables, and our favorite methods for square systems usually cannot be applied to them.

Fortunately, there are some methods for solving those systems – Gaussian elimination, classical iterative methods, Rohn's method, least squares methods, or linear programming. The main goal of this work is to present an overview of existing methods for computing enclosures of solution sets of overdetermined interval linear systems. To the best of our knowledge, that has not been done for overdetermined systems. We explain how these methods work in a brief but understandable way even for researchers from various fields not so familiar with interval linear systems. We also mention some pitfalls and specialties connected with these methods. Some of them behave in a useful way – they are able to compute a very narrow enclosure or an interval hull, or they can reveal unsolvability of an interval overdetermined system. On the other hand, many of them cannot decide whether a solution of an interval system is unbounded or whether the system is unsolvable. Moreover, sometimes if a system has certain properties, existing methods are not able to return any meaningful result. It is interesting to observe how some efficient methods fail when the radii of intervals change and, on the other hand, to see how some simple, one could say "stupid", methods rule. After introducing the methods, we provide results of numerical comparison of these methods concerning the speed, enclosure tightness, and some other properties. As always, not every method is useful for every problem. When describing the methods, we point out the cases when they are useful.

2 Basic notation and definitions

We start with basic notation. Here, we provide only a small part, which we will use subsequently. For a deeper introduction to interval analysis, see [4]. We denote interval structures in boldface ($\boldsymbol{A}, \boldsymbol{b}$). Point real structures are denoted in regular type (A, b). A coefficient of a matrix in i-th row and j-th column is denoted A_{ij} or $\boldsymbol{A}_{ij}$. Here we work only with closed real intervals $[c, d]$, where $c \leq d$. The set $\mathbb{IR}$ stands for the set of all real closed intervals. An interval $\boldsymbol{x}$ can be defined in two ways. The first one is by upper and lower bounds $\boldsymbol{x} = [\underline{x}, \overline{x}] = \{x \in \mathbb{R} \mid \underline{x} \leq x \leq \overline{x}\}$. The second is by midpoint and radius $\boldsymbol{x} = \langle x_c, x_\Delta \rangle = \{x \in \mathbb{R} \mid |x_c - x| \leq x_\Delta\}$.

The relation $\leq$ is defined component-wise for matrices. For two $m \times n$ matrices A and B,

$$A \leq B \text{ if } A_{ij} \leq B_{ij} \text{ for all } i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}.$$

With intervals, we can build more complex structures – interval matrices (of which vectors are special case). An $m \times n$ interval matrix $\boldsymbol{A}$ is defined as

$$\boldsymbol{A} = [\underline{A}, \overline{A}] = \{A \in \mathbb{R}^{m \times n} \mid \underline{A} \leq A \leq \overline{A}\}.$$

Similarly as with single intervals, an interval matrix can be also defined using midpoint and radius $\boldsymbol{A} = \langle A_c, A_\Delta \rangle$. The matrix A_c is called the *midpoint matrix*, and A_Δ is called the *radius matrix*. The midpoint matrix coefficients consists of corresponding midpoints of intervals of $\boldsymbol{A}$; the radius matrix consists of corresponding radii of $\boldsymbol{A}$.

In the section devoted to comparison of various methods, we will use *width* of an interval $\boldsymbol{y} = [\underline{y}, \overline{y}]$ defined as

$$\mathrm{width}\,(\boldsymbol{y}) = \overline{y} - \underline{y}.$$

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

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

For a vector $x \in \mathbb{R}^n$, we denote

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

We continue with a definition of an overdetermined interval linear system.

Definition 2.1 *(Overdetermined interval linear system) Given an interval matrix $\boldsymbol{A} \in \mathbb{IR}^{m \times n}$ with $m > n$ and an interval vector $\boldsymbol{b} \in \mathbb{IR}^m$, we call*

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

an overdetermined interval linear system (OILS).

Simply, an overdetermined interval linear system is a general interval linear system that consist of more equations than variables. If $m = n$, the system is called *square*.

When we talk about interval linear systems, we must define what we mean by the solution of an interval linear system (ILS).

Definition 2.2 *(Solution set of ILS) The solution set Σ of an interval linear system $\boldsymbol{A}x = \boldsymbol{b}$ is*

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

In other words, it is a collection of all solutions to all instances of an interval linear system. By an *instance*, we mean a point real system that we get when independently choosing a real number from each interval coefficient of the interval system. This approach is different from the least squares approach, i.e.,

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

For more information about this approach and the relationship of Σ and Σ_{lsq}, see [5]. If any instance of the interval system has no solutions, we call the whole interval system *unsolvable*. To provide a better conception of the solution set, we show such a solution set at Figure 1 (plotted using the INTLAB routine `plotlinsol`).

$$\begin{array}{rcrcrcl} [-5,10]\,x & + & 10\,y & + & [15,20]\,z & = & [50,100], \\ 10\,x & + & -5\,y & + & [5,15]\,z & = & [-50,50], \\ 10\,x & + & [10,25]\,y & + & [-10,-5]\,z & = & [50,100]. \end{array}$$

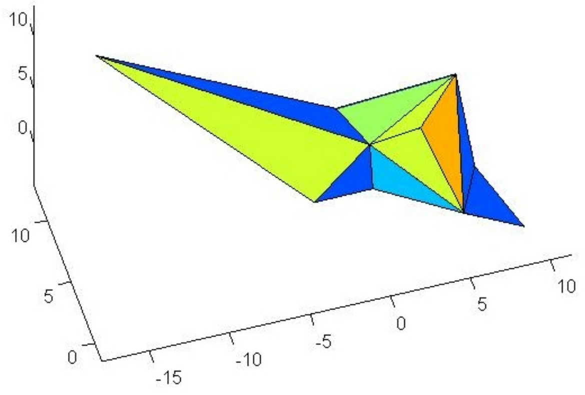

Figure 1: Graphical view of the solution set of the interval linear system above

The solution set is generally a polyhedral set, not necessarily convex. Nevertheless, it is convex in each orthant of the space. As we can see, this set is quite difficult to describe, and there are many ways it can be estimated. One can find an n-dimensional box (aligned with axes), as tight as possible, that contains the solution set. We call this box the *interval hull*. Unfortunately, it can be proved that computing the interval hull is NP-hard [13]. That is why we look for a box a little bit wider, still as narrow as possible, that contains the interval hull. We call this box an *interval enclosure*. There is a large variety of methods we can use to compute an interval enclosure. If we have two such enclosures returned by two different methods, we need some way to compare them. For n-dimensional enclosing boxes $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)^T$ and $\boldsymbol{y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_n)^T$, we define their *ratio* as

$$\text{ratio}(\boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n} \sum_{i=1}^{n} \frac{\text{width}\,(\boldsymbol{x}_i)}{\text{width}\,(\boldsymbol{y}_i)} \,.$$

The above concepts form a basic terminology of the subject, and we are able to move further to proper descriptions of the methods.

3 Gaussian Elimination (GE)

The first thing that can come to our mind is Gaussian elimination. The interval GE for overdetermined systems was proposed by Hansen in [1]. The idea is very similar to that for point real systems. We present a slightly modified version. We eliminate rows using interval operations, with the only difference that we eliminate the system $(\boldsymbol{A}|\boldsymbol{b})$ to the shape

$$(\boldsymbol{A} \mid \boldsymbol{b}) \sim \ldots \sim \left(\begin{array}{cc|c} \boldsymbol{C} & \boldsymbol{d} & \boldsymbol{e} \\ \boldsymbol{0} & \boldsymbol{f} & \boldsymbol{g} \end{array} \right),$$

where $\boldsymbol{C}$ is an $(n-1) \times (n-1)$ interval matrix in row echelon form (REF) with $[1,1]$ intervals in the pivot positions, $\boldsymbol{d}$ and $\boldsymbol{e}$ are $(n-1) \times 1$ interval vectors, $\boldsymbol{0}$ is an

$(m-n+1) \times (m-n+1)$ matrix composed of the intervals $[0,0]$, and $\boldsymbol{f}$ and $\boldsymbol{g}$ are $(m-n+1) \times 1$ interval vectors.

One thing often is not clear to people familiar with interval arithmetic when Gaussian elimination is first explained to them: Why in the pivot positions there are $[1,1]$ intervals, and why there are $[0,0]$ intervals in the $\mathbf{0}$ matrix. The reason is quite simple, but not obvious – when performing eliminating operations on pivots and elements beneath, we can think of it as performing them in all instances of an interval system separately (after elimination pivots in instances are equal to 1, elements under the pivots are equal to 0). When we assemble the eliminated point real systems back into an interval matrix, we know that we have $[1,1], [0,0]$ intervals on these positions. There is no need to overestimate them using conservative interval operations. For the other coefficients, we have to use interval arithmetic to compute their interval enclosures.

Now, we realize that vectors $\boldsymbol{f}$, $\boldsymbol{g}$ form $m-n+1$ interval equations in the shape

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

The solution of these equations is the intersection of all the intervals $\boldsymbol{g}_i / \boldsymbol{f}_i$, and we get an enclosure of the variable x_n. 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 that there is a huge overestimation due to large number of interval operations occurred. The enclosures for the other variables can be obtained using backward substitution computed with interval arithmetic.

This algorithm works only for very small $m \times n$ systems ($n \sim 4$). For larger systems, we get a huge overestimation. In Table 1, we can see the rapid growth of radii of variable enclosures as they are computed by backward substitution, caused by using interval operations with wider and wider intervals each step. In the last column, there are radii of another, also verified, enclosure computed by `verifylss` in INTLAB.

Table 1: The growing overestimation of variable enclosures (random system 15×13) with random radii $< 10^{-3}$ caused by Gaussian elimination and backward substitution

variable	radius (GE)	radius (`verifylss`)
x_1	257.47	0.012
x_2	165.19	0.014
x_3	116.93	0.012
x_4	100.56	0.010
x_5	46.58	0.017
x_6	44.94	0.001
x_7	33.38	0.007
x_8	6.52	0.010
x_9	14.97	0.017
x_{10}	7.63	0.009
x_{11}	3.71	0.009
x_{12}	2.54	0.007
x_{13}	4.68	0.011

The larger the system, the bigger is the overestimation. Therefore, it is necessary to use preconditioning. We use the preconditioner introduced by Hansen in [1].

$$C = \begin{bmatrix} A_1^c & 0 \\ A_2^c & I \end{bmatrix}^{-1},$$

where A_1^c consists of first n rows of A_c (an approximation of a midpoint matrix of $\boldsymbol{A}$) and A_2^c consists of the remaining $m-n$ rows of A_c. The operation $(.)^{-1}$ is an approximate inverse operation. Next, we solve a new system

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

Of course, the preconditioning can cause an overestimation of the resulting enclosure, but it is often significantly smaller when compared to performing GE without preconditioning. The preconditioning might enlarge the solution set of the system, and that is why unsolvable systems might become solvable after applying it. Therefore, if we want to check unsolvability, we cannot use preconditioning. If we use elimination to reduced row echelon form (RREF) and allow multiple right-hand sides, we are able to compute an interval matrix containing $\boldsymbol{A}^{-1}$. For a proper definition of the interval inverse matrix see [12].

4 Iterative Methods

There are many interval modifications of methods for square non-interval systems – Jacobi, Gauss-Seidel, Krawczyk, but we usually cannot use them when dealing with overdetermined systems. Moreover, without preconditioning, these methods often work really badly. Nevertheless, if we want to use our known iterative methods, we can use the preconditioning mentioned in the section 3. We can see that after this preconditioning, the matrix of the new system is usually of the shape

$$\begin{pmatrix} \boldsymbol{I}^{\sim} \\ \boldsymbol{0}^{\sim} \end{pmatrix},$$

where $\boldsymbol{I}^{\sim}$ is an $n \times n$ interval matrix (with very narrow intervals) containing the point identity matrix, and $\boldsymbol{0}^{\sim}$ is an $(m-n) \times n$ interval matrix (with very narrow intervals) containing the zero matrix. Then we can use our favorite iterative method for the upper square subsystem of the preconditioned system. Let us choose the Jacobi method, for example. We start with an initial enclosure $\boldsymbol{x}^{(0)}$ and iteratively "sharpen" this enclosure using the following formula (at each step, we simply express the variable $\boldsymbol{x}_i$ from the i-th equation, for more information, see [4])

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

After every iteration, we intersect with the old enclosure to obtain

$$\boldsymbol{x}_i^{(k+1)} = \boldsymbol{x}_i^{(k)} \cap \boldsymbol{x}_i^* \quad \text{for } i = (1, \ldots, n).$$

The Gauss-Seidel method differs in only one detail; we do not wait to intersect. Instead, we intersect after every computation of $\boldsymbol{x}_i^*$ and immediately use the new value for further computing. This generally leads to fewer iteration steps.

The problem is that when using these methods, intervals containing zero are not allowed on a diagonal (since there is the division with $\boldsymbol{A}_{ii}$). That case happens if the radii of the original matrix are "large" (even 10^{-2}).

This method looks rather simple. We loose some information due to precoditioning and chopping the last $m-n$ rows of the original system, but we can realize that the

Jacobi method can be parallelized effectively. If there is an effective way to determine $\boldsymbol{x}^{(0)}$, this method can be used for very fast sharpening of $\boldsymbol{x}^{(0)}$, for example in Constraint Programming as a sharpening step between of runs of another sharpening method.

5 Rohn's Method

We would like to review the method introduced by J. Rohn. Due to lack of space, we will describe it only briefly. For more information and theoretical insight, one can take a look in [10]. The basis of the method is the following theorem.

Theorem (Rohn) 1 *Let $\boldsymbol{A}x = \boldsymbol{b}$ be an IOLS with a solution set Σ ($\boldsymbol{A}$ is an $m \times n$ matrix). Let R be an arbitrary real $n \times m$ matrix, let x_0 and $d > 0$ be arbitrary n-dimensional real vectors such that*

$$Gd + g < d,$$

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_0. To compute d, we can, for example, rewrite the inequality as

$$d = Gd + g + f,$$

for some small vector $f > 0$. Then start with $d = 0$ and iteratively refine d. This algorithm will stop after a finite number of steps if the spectral radius of G is less than 1. Otherwise, we do not know what happens. During practical testing with random systems with radii of intervals close to 0.1, the vector $d > 0$ was rarely found.

We still have to determine x_0 and R. For the start, we can take

$$x_0 \approx Rb_c,$$

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

but not necessarily. The theorem provides a clever instrument for iterative improvement of an enclosure. We do not have to use only A_c to compute R; we can take any (e.g., random) $A \in \boldsymbol{A}$, compute an enclosure, and then intersect it with the old one. We can repeat this process as many times as we want and obtain an iterative improvement of the enclosure. For smaller systems, the iterative improvement works pretty well. Table 2 shows the ratios of enclosure widths returned by iterative versions for 10, 100, and 1000 iterations and by the non-iterative Rohn method. In the following text, we will call the first method the *basic method* and use the notion *basic enclosure*. When we use the iterative improvement ,we will talk about the *iterative method* and the *iterative enclosure*.

Table 2: Rohn's method – ratios of iterative enclosures (10, 100, and 1000 iterations) and basic enclosure

system	10 it.	100 it.	1000 it.
5×3	0.73	0.57	0.50
15×10	0.89	0.82	0.76
25×21	0.94	0.90	0.87
35×23	0.95	0.92	0.90
50×35	0.97	0.94	0.93
70×55	0.98	0.96	0.95
100×87	0.98	0.97	0.97

6 Least Squares Method

This approach can be found in [5] or [14]. It can be proved that

$$\text{the hull of } \Sigma \;\subseteq\; \text{the hull of } \Sigma_{lsq}.$$

Hence, we can use the least squares approach to compute a rigorous enclosure of the solution set to an OILS. We use the least squares formula $A^TAx = A^Tb$ for point real systems. The interval analogue

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

does not work because of the two interval matrix multiplications. Even if we use some preconditioner C and write

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

that does not work either. However, we can use an equivalent expression of the least squares formula for point real systems. If we express it for interval systems, we get

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

Now, we have a square system, and we can apply some suitable method for square systems. We compute the vector solution $(\boldsymbol{y}, \boldsymbol{x})^T \in \mathbb{IR}^{m+n}$ and take $\boldsymbol{x}$ as the enclosure of the solution. It can be seen that the returned interval vector contains the solution of the interval least squares problem. Therefore, this method returns a solution even if the system is unsolvable. Another drawback is that if the original system is of size $m \times n$, we have to solve a new system of size $(m+n) \times (m+n)$. On the other hand, this method computes very sharp interval enclosures. It is used in the routine `verifylss` of INTLAB 6 [15]. For more detailed description of this routine, see [2]. When solving the system in this way, we can notice the dependencies in the new system. Each interval coefficient from the original system is used twice in the new system; when we choose one number from the first interval, we should choose the same value in the second one to avoid overestimation. We may be able to use methods dealing with dependencies between coefficients in interval linear systems (e.g.,. [3, 8, 16]).

7 Linear Programming (LP)

It is possible to compute the interval hull of the ILS solution set using linear programming, using the famous theorem by Oettli-Prager, which can be found with proof in [9].

Theorem 7.1 *(Oettli-Prager) Consider an interval linear system* $\boldsymbol{A}x = \boldsymbol{b}$. *A vector* $x \in \mathbb{R}^n$ *is a solution to this system* ($x \in \Sigma$) *if and only if*

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

Unfortunately, we are still not able to use LP because of the absolute values. Now we demonstrate how to rewrite this problem using linear inequalities only. We can get rid of the first one by decomposing it into two cases

$$A_c x - b_c \ \le A_\Delta |x| + b_\Delta, \tag{2}$$

$$-(A_c x - b_c) \ \le A_\Delta |x| + b_\Delta. \tag{3}$$

The second absolute value can be rewritten with the use of knowledge of the orthant we currently "are" inside. We have

$$|x| = D_z x, \text{ where } z = \operatorname{sgn} x,$$

giving a rise to the condition

$$0 \le D_z x. \tag{4}$$

For every orthant, the conditions (2), (3), and (4) form a system of linear inequalities. Therefore, we can use linear programming. Unfortunately, we have to solve ($2^n \times 2n$) linear programming problems (we compute the upper and lower bounds in each coordinate). That is obviously too much computing. However, we can compute the enclosure of the solution set of the system with some other method (least squares, Rohn) and then apply linear programming to only those orthants where this enclosure lies. This approach is often much faster. Table 3 illustrates a large speedup when solving the system with different methods before applying LP. The sign '–' means that we omitted the testing because of enormous computing time when compared to LP with presolving.

Table 3: Comparison of times of LP in all orthants and LP with presolving with a different method (here `verifylss`)

system	time LP	time LP presolved
5×3	6 sec	1 sec
9×5	43 sec	1.68 sec
13×7	5 min	3.59 sec
15×9	28 min	4.1 sec
25×21	-	13 sec
35×23	-	19 sec
45×31	-	43 sec
55×35	-	1 min
73×55	-	9 min

8 Comparison of Methods

The second goal of this paper is a numerical comparison of methods for enclosing solutions of overdetermined systems. All subsequent tests were computed using the following hardware and software:

- Processor – AMD Phenom(tm) II X6 1090T
- Memory – 15579 MB
- Matlab R2010b
- INTLAB 6 (see [15])
- Versoft 10 (see [11]) for linear programming

It is difficult to imagine rectangular matrices in some special shape. Therefore, we test the methods on systems composed of random matrices and vectors. In this work we use four parameters:

- *maximum radius,*
- *midpoint range,*
- *stopping parameter,* and
- *maximum number of iterations.*

Most of these parameters are used to generate random interval systems with certain properties. The key parameter is *maximum radius* of interval coefficients of an interval linear system. For example, when this parameter is -4, every interval coefficient $\boldsymbol{y} = \langle y_c, y_\Delta \rangle$ in the system satisfies $y_\Delta \leq 10^{-4}$. *Maximum radius* is usually negative, because the radii greater than 1 lead to singularities. The radii in a system are not the same; they are chosen randomly with respect to the maximum radius parameter.

Another important parameter is the *midpoint range*, which specifies the range of midpoints of intervals. There might be differences in behaviour of the methods between the case when all the intervals in the system have relatively close midpoints and the case when the midpoints are quite different, as in the vector $(10, 1, 11234, 0.1)^T$. We will test on two cases: i) with midpoints uniformly chosen from $[-25, 25]$, and ii) with midpoints uniformly chosen from $[-1000, 1000]$.

Some methods need a small real positive number ϵ as a *stopping parameter* (some methods need a small positive vector, but we can use $f = (\epsilon, \epsilon, \ldots, \epsilon)^T$). We will choose $\epsilon = 10^{(\text{maximum radius}-2)}$. After performing some practical tests, this seems to be a reasonable choice.

Some iterative methods use the parameter *maximum of iterations* as a safety precaution for the cases when the stopping parameter does not work. We use the value 20 for all the tested cases.

We will test systems up to size about $m = 1000$, that is, up to a thousand of linear equations. We chose this upper bound on size of the tested systems after consulting with a colleague from the Faculty of Civil Engineering. These are maybe the largest linear systems they need to solve for many technical purposes. If a solution of larger system is needed, then it is usually better to split the problem into parts, solve the parts separately, and then assemble the solution. For the sake of clarity, we establish the following identifiers of the methods:

- *IGSpre* - interval Gauss-Seidel iterative method with preconditioning
- *IGEpre* - interval Gaussian elimination with preconditioning
- *Rohn* - basic Rohn method
- *Verifylss* - method verifylss from toolbox INTLAB by Rump
- *LPver* - linear programming from toolbox Versoft by Rohn

8.1 Comparison of enclosures

We are going to compare the ratios of enclosures (defined in Section 2) returned by all the mentioned methods. For each method and each system size, we compute ratios of enclosures for 100 random systems. The numbers displayed in the subsequent tables for each system size and method are average ratios of all 100 ratios.

We divide the testing in two phases. For larger systems, it is very time consuming to compute many enclosures using *LPver*. Moreover, for small radii of intervals, the method returns a *NaN* solution. That is why we tested the methods on small systems first, and the ratios of enclosures were computed according to *LPver* results (interval hull). In second phase (for larger systems), they were computed according to *Verifylss* enclosures. Table 4 displays the ratios of small systems. Clearly, the lower the ratio, the better the enclosure.

Table 4: Ratios of widths of enclosures, *LPver* is 1 (maximum radii $\leq 10^{-3}$)

system	*IGSpre*	*IGEpre*	*Rohn*	*Verifylss*
5×3	2.9968	2.9969	1.2347	1.1893
15×9	7.7057	7.7069	1.1601	1.1500
35×23	8.3591	8.3602	1.1276	1.1249
55×35	11.5026	11.5064	1.1336	1.1331
73×55	17.6528	17.6624	1.0828	1.0848

We can see that *IGEpre* and *IGSpre* are doing really badly, and that their enclosure ratios are about the same value. We attribute that to the use of the same preconditioner. *Rohn* and *Verifyllss* provide good results. *Verifylss* is a little bit better, but the difference is slowly disappearing with the growth of system size. That is confirmed by Table 5. We made the radii smaller ($\leq 10^{-5}$) since *IGEpre* often returned infinite solution due to a large number of interval operations.

Table 5: Ratios of widths of enclosures, *Verifylss* is 1 (maximum radii $\leq 10^{-5}$)

system	*IGSpre*	*IGEpre*	*Rohn*
100×45	21.1668	21.1668	1.0237
100×87	9.2420	9.2420	1.0062
180×125	26.6744	26.6744	1.0064
180×170	15.0262	15.0262	1.0020
290×190	37.6976	37.6977	1.0044
290×260	26.9361	26.9361	1.0018
380×275	40.5159	40.5160	1.0028
380×360	32.9500	32.9504	1.0009
500×350	66.9764	66.9771	1.0022
500×470	34.2512	34.2516	1.0007

8.2 Enclosure computation time comparison

We now compare the computation times of the methods *IGSpre*, *IGEpre*, *Rohn*, and *Verifylss*. Midpoint matrices and vectors are chosen uniformly from the interval

$[-1000, 1000]$. Tables 6 and 7 show the average CPU times measured by Matlab functions *tic* and *toc* for maximum radius $\leq 10^{-3}$ and $\leq 10^{-5}$, respectively. Each method was tested for each system size on 100 random systems.

Table 6: Average times (in seconds) of all methods for maximum radius $\leq 10^{-3}$

system	*IGSpre*	*IGEpre*	*Rohn*	*Verifylss*
5×3	0.0113	0.0117	0.00065	0.0021
15×13	0.0367	0.0685	0.00086	0.0023
35×23	0.0636	0.2840	0.0012	0.0036
50×35	0.0983	0.5806	0.0019	0.0052
100×87	0.3063	2.4631	0.0115	0.0169
200×170	0.6632	9.8570	0.0523	0.0761
380×275	1.3253	34.5861	0.1765	0.3257
500×470	2.9486	69.6341	0.6235	0.9154

Table 7: Average times in seconds of all methods for maximum radius $\leq 10^{-5}$

system	*IGSpre*	*IGEpre*	*Rohn*	*Verifylss*
5×3	0.0113	0.0118	0.00066	0.0021
15×13	0.0356	0.0687	0.00086	0.0023
35×23	0.0608	0.2836	0.0012	0.0036
50×35	0.0911	0.5809	0.0019	0.0053
100×87	0.2416	2.5224	0.0117	0.0171
200×170	0.5718	10.6842	0.0561	0.0760
380×275	1.2515	37.0210	0.1924	0.3427
500×470	2.5774	75.5171	0.6616	0.9761

We can see that the methods do not show any remarkable sensibility to the changes of radii of the intervals if we keep the *stopping criterion* equal to (*maximum radius* -2). Also, there was no extraordinary change when lowering the *midpoint range* to something much smaller (e.g., $[-25, 25]$). The complete time winner is *Rohn*, and the second is *Verifylss*. The method *IGEpre* is doing badly. We shall look more carefully at the comparison between *Rohn* and *Verifylss*. The method *Rohn* is almost always faster. According to Table 5, we can object that it pays off to take a little bit more time and compute more exact enclosure using *Verifylss*. But let us look closer at Table 8. It displays the ratios of computation times of *Rohn* and *Verifylss* for systems of a broader spectrum of sizes. We did the tests for many random systems; here we selected some interesting ones. It is possible to see that, for $m \times n$-systems, where m is much bigger than n, *Rohn* is much more faster. For example, on a system of size 278×35, we can call the iterative *Rohn* with 13 iterations within the same time *Verifylss* needs to do the computation, and we can possibly sharpen the enclosure. The question whether this procedure pays off is answered by Table 2. For relatively small systems, it definitely pays off. For larger systems, we might get not so significant improvements, and it is safer to use *Verifylss*.

Table 8: Ratios of *Verifylss* and *Rohn* computation times

system	$\frac{t(\textit{Verifylss})}{t(\textit{Rohn})}$
80×41	2.8
189×166	1.5
278×35	13.8
377×319	1.6
525×285	2.7
712×271	4.5
807×68	46.5
894×8	423.7
894×797	1.5
906×128	19.1
978×235	9.4
1000×663	2.0

9 Conclusion

We introduced several methods for solving overdetermined interval linear systems — Gaussian elimination, iterative methods, Rohn's method, the least squares method, and linear programming. We mentioned some advantageous and unfavourable properties of these methods. At the end of the work, we showed the comparison of all the mentioned methods. As usual, it is hard to tell which method is actually the best. Linear programming is suitable when we want to compute the interval hull of an interval system and have enough time. When our system consists of narrow intervals (their radii is smaller than 10^{-3}) or when our system is thin (its matrix is of size $m \times n$, where $m >> n$), it is advantageous to use Rohn's method. Otherwise, it is favourable to use the least squares method for fast computing of a sharp interval enclosure. Iterative methods can be used when we already know some enclosure and want to try to sharpen it (e.g., as a part of some constraint solver) and then passing it as an input to another method e.g., LP. Gaussian elimination seems to work poorly, but when we consider the solvability of the system, it might give us valuable information about the nonexistence of the solution.

Computing an enclosure of the solution set of overdetermined interval linear systems can occur as a subproblem of many computational tasks (e.g., constraint satisfaction problems). Often, information about the solvability of a system is often important (e.g., in system validation, technical computing). There is still much work to do in this area. We believe that even faster and sharper algorithms can be developed. Simultaneously, we are working on methods checking the solvability of overdetermined interval linear systems. Hopefully, sufficient or necessary conditions for solvability or unsolvability of OILS can be found. Since preconditioning is used for many methods, it would be useful to test different types of preconditioners. Some methods use randomness, e.g., the iterative Rohn method. There is a question of whether we can effectively derandomize these methods.

Acknowledgements

The research is supported by the Grant Agency of the Charles University (GAUK) grant no. 712912. Many thanks to Jakub Kraus from Czech Technical University in Prague, Faculty of Civil Engineering, for consultation about practical use of linear systems in civil engineering.

References

[1] Eldon. R. Hansen and G. William Walster. Solving overdetermined systems of interval linear equations. *Reliable Computing*, 12(3):239–243, 2006.

[2] Gareth. I. Hargreaves. Interval analysis in MATLAB. 2002.

[3] Milan Hladík. Enclosures for the solution set of parametric interval linear systems. *Int. J. Appl. Math. Comput. Sci.*, 3(22):561–574, 2012.

[4] Ramon E. Moore, Ralph Baker Kearfott, and Michael J. Cloud. *Introduction to Interval Analysis*. Society for Industrial Mathematics, 2009.

[5] Arnold Neumaier. Linear interval equations. In *Interval Mathematics 1985*, pages 109–120. Springer, 1986.

[6] Arnold Neumaier. *Interval Methods for Systems of Equations*. Cambridge University Press, Cambridge, 1990.

[7] Shucheng Cheng Ning and Ralph Baker Kearfott. A comparison of some methods for solving linear interval equations. *SIAM J. Numer. Anal.*, 34(4):1289–1305, 1997.

[8] Evgenija D. Popova. On the solution of parametrised linear systems. In Walter Krämer and Jürgen Wolff von Gudenberg, editors, *Scientific Computing, Validated Numerics, Interval Methods*, pages 127–138. Kluwer, 2001.

[9] Jiří Rohn. Systems of linear interval equations. *Linear Algebra and Its Applications*, 126:39–78, 1989.

[10] Jiří Rohn. Enclosing solutions of overdetermined systems of linear interval equations. *Reliable Computing*, 2(2):167–171, 1996.

[11] Jiří Rohn. VERSOFT: Verification software in MATLAB / INTLAB, version 10, 2009.

[12] Jiří Rohn. A handbook of results on interval linear problems. Technical Report 1163, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, 2012.

[13] Jiří Rohn and Vladik 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.

[14] Siegfried M. Rump. Solving algebraic problems with high accuracy. In *Proc. of the Symposium on a New Approach to Scientific Computation*, pages 51–120, San Diego, CA, USA, 1983. Academic Press Professional, Inc.

[15] Siegfried M. Rump. INTLAB - INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999. `http://www.ti3.tu-harburg.de/rump/`.

[16] Siegfried M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numer.*, 19:287–449, 2010.