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

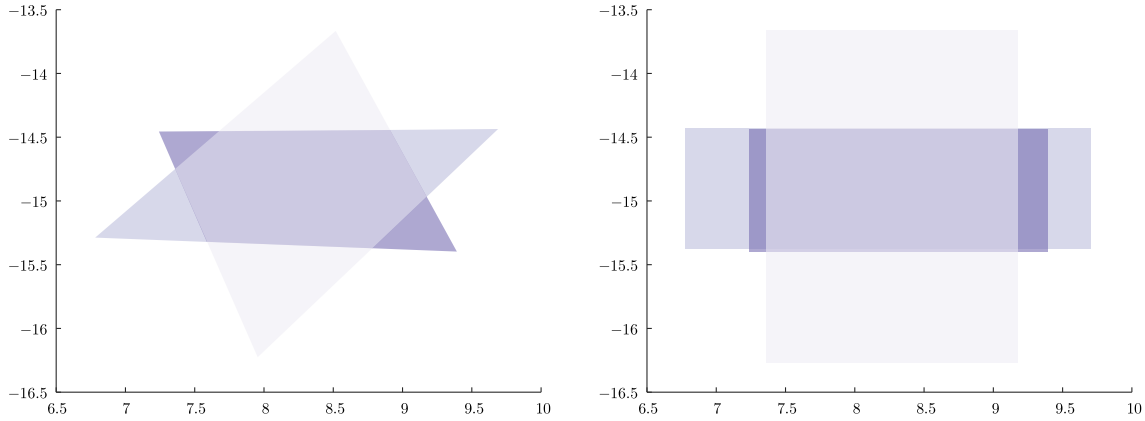

Figure 6.4: The subsquares from Example 6.11 – on the left there are the solution sets, on the right there are the hulls of $\boldsymbol{A}_{\{1,2\}}x = \boldsymbol{b}_{\{1,2\}}$ (the intermediate color), $\boldsymbol{A}_{\{1,3\}}x = \boldsymbol{b}_{\{1,3\}}$ (the darkest color) and $\boldsymbol{A}_{\{2,3\}}x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

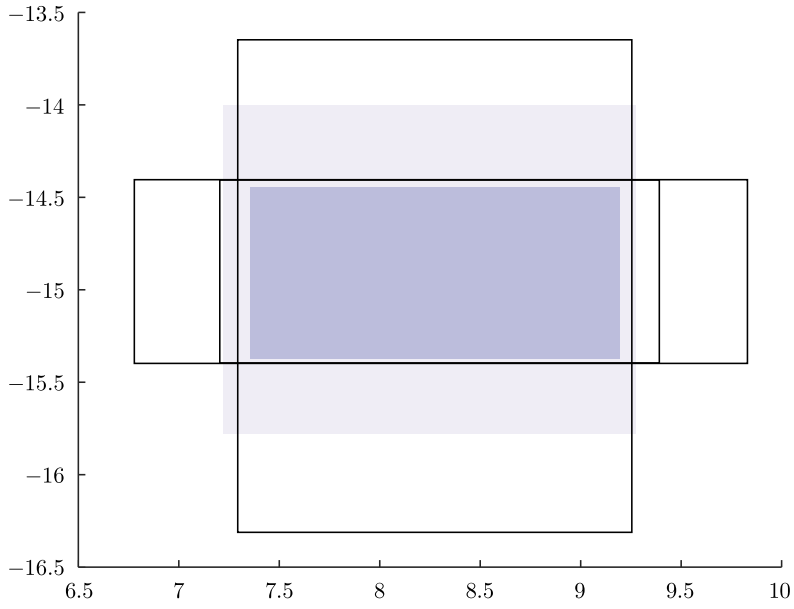

Figure 6.5: The enclosures of the subsquares from Example 6.11. Rectangles represent enclosures of subsquares computed by the HBR method. The darkest area represents the hull of the original overdetermined system, the lighter rectangle is an enclosure computed by Rohn's method.

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

   $$\texttt{Indices} := \texttt{Waiting} \cup \text{randsel}(n - |\texttt{Waiting}|, \texttt{Covered}).$$

5. Otherwise, set

   $$\texttt{Indices} := \text{randsel}(\texttt{overlap, Covered}) \cup \text{randsel}(n - \texttt{overlap}, \texttt{Waiting}).$$

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 $0 \notin \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}$ 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., $\operatorname{rank}(A) = n$. 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

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

Putting all together, we obtain

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

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

$$\|A \cdot B\| \le \|A\| \cdot \|B\|$$

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 $\varrho(\boldsymbol{A}) \leq \|\boldsymbol{A}\|_u$. According to (3.6) and (3.7) we have

$$\|\boldsymbol{A}\|_u \leq \alpha < 1 \iff \operatorname{mag}(\boldsymbol{A}) \cdot u \leq \alpha \cdot u < u. \tag{7.6}$$

The matrix $C = \operatorname{mag}(\boldsymbol{A})$ 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 $u_0 = (1, \ldots, 1)^T$).
2. Compute $u_{k+1} = \operatorname{mag}(\boldsymbol{A})u_k$ 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

$$\begin{aligned} u_0 &= (1, 1, 1)^T, \quad \|A\|_{u_0} = 1, \\ u_1 &= (1, 0.8, 1)^T, \quad \|A\|_{u_1} = 0.96 < 1. \end{aligned}$$

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 $(I - RA) \in (I - R\boldsymbol{A})$

$$\varrho(I - RA) \leq \varrho(\mathrm{mag}(I - R\boldsymbol{A})) \leq \|I - R\boldsymbol{A}\| < 1.$$

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 $|C| \le |A_c^+|A_\Delta$. Hence, together with *a)*, it gives

$$\varrho(C) \le \varrho\left(|A_c^+|A_\Delta\right) < 1.$$

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 $\operatorname{mag}(I - R\boldsymbol{A})u < u$ for a suitable R. Using the properties (3.1)–(3.5) we get

$$\operatorname{mag}(I - R\boldsymbol{A}) = |I - RA_c| + |R|A_\Delta$$

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 $\operatorname{sign}(x)$ 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.