Once again, for any x^0 the solution set to (10.8) and (10.9) is described by

$$|A_c(x - x^0) + f(x^0)| \leq A_\Delta |x - x^0|,$$
$$B_c(x - x^0) \leq B_\Delta |x - x^0| - g(x^0),$$

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

$$\alpha_i = \frac{1}{x_i^\Delta}(x_i^c - x_i^0),$$
$$v_i = \frac{1}{x_i^\Delta}(x_i^c x_i^0 - \overline{x}_i \underline{x}_i).$$

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

$$\alpha_i = \frac{1}{2x_i^\Delta}\left(|\overline{x}_i - x_i^0| - |\underline{x}_i - x_i^0|\right) = \frac{1}{2x_i^\Delta}\left(\overline{x}_i - x_i^0 - (x_i^0 - \underline{x}_i)\right) =$$
$$= \frac{1}{x_i^\Delta}\left(\frac{(\overline{x}_i - \underline{x}_i)}{2} - x_i^0\right) = \frac{1}{x_i^\Delta}\left(x_i^c - x_i^0\right),$$

$$\begin{aligned}\beta_i &= \frac{1}{2x_i^\Delta}\Big((\overline{x}_i - x_i^0)|\underline{x}_i - x_i^0| - (\underline{x}_i - x_i^0)|\overline{x}_i - x_i^0|\Big) = \\ &= \frac{1}{2x_i^\Delta}\Big((\overline{x}_i - x_i^0)(x_i^0 - \underline{x}_i) - (\underline{x}_i - x_i^0)(\overline{x}_i - x_i^0)\Big) = \\ &= \frac{1}{x_i^\Delta}(\overline{x}_i - x_i^0)(x_i^0 - \underline{x}_i).\end{aligned}$$

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

$$\begin{aligned}(A^c - A^\Delta D_\alpha)x &\le A^c x^0 + A^\Delta(-D_\alpha x^0 + \beta) - f(x^0), \\ (-A^c - A^\Delta D_\alpha)x &\le -A^c x^0 + A^\Delta(-D_\alpha x^0 + \beta) + f(x^0).\end{aligned}$$

□

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:

$$h(x) \subseteq \boldsymbol{A}(x - x^0) + b, \quad \forall x \in \boldsymbol{x}, \tag{10.13}$$

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

$$\begin{aligned} h(x) &\le A(x - x^0) + b + \overline{(\boldsymbol{A} - A)(\boldsymbol{x} - x^0)}, \\ h(x) &\ge A(x - x^0) + b + \underline{(\boldsymbol{A} - A)(\boldsymbol{x} - x^0)}. \end{aligned}$$

For $A = A_c, x^0 = x^c$ we particularly get

$$\begin{aligned} h(x) &\le A(x - x^c) + b + A^\Delta x^\Delta, \\ h(x) &\ge A(x - x^c) + b - A^\Delta x^\Delta. \end{aligned}$$

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

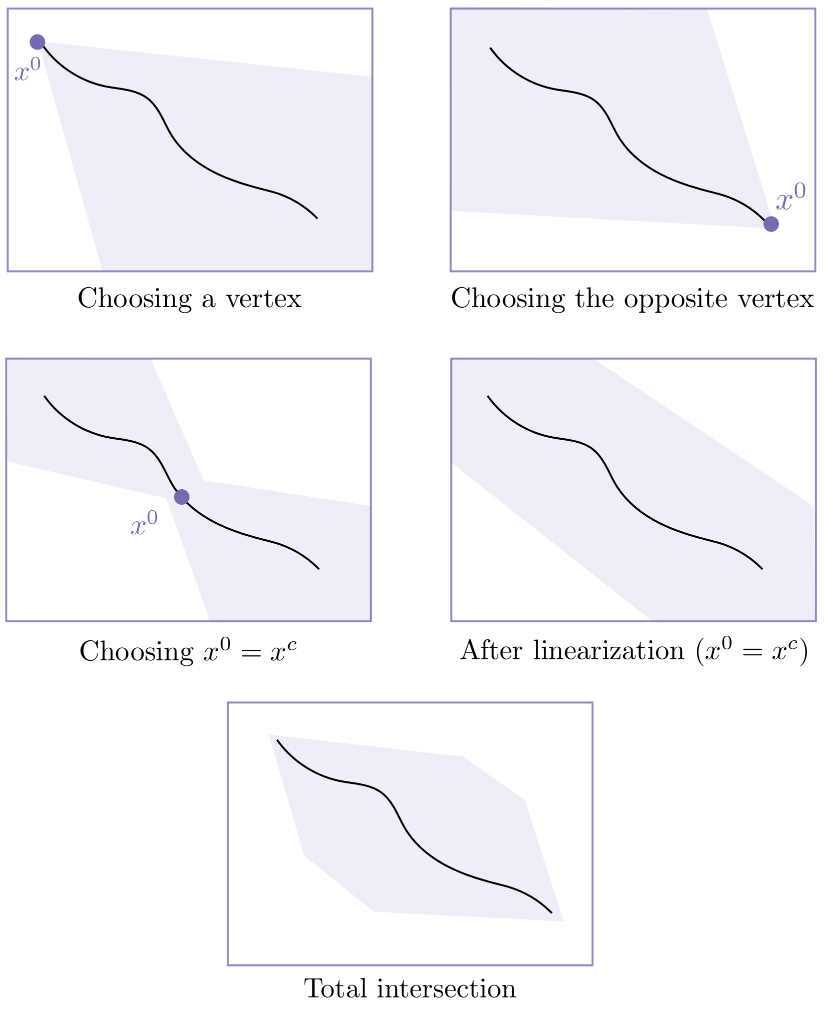

Figure 10.1: Illustration of relaxations obtained by selecting different centers of linearizations. The darker area is a linearized enclosure. The curve represents a set described by the constraints (10.1), (10.2)

.

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

$$\pi^2 y - 4x^2 \sin(x) = 0, \tag{10.19}$$
$$y - \cos(x + \pi/2) = 0. \tag{10.20}$$

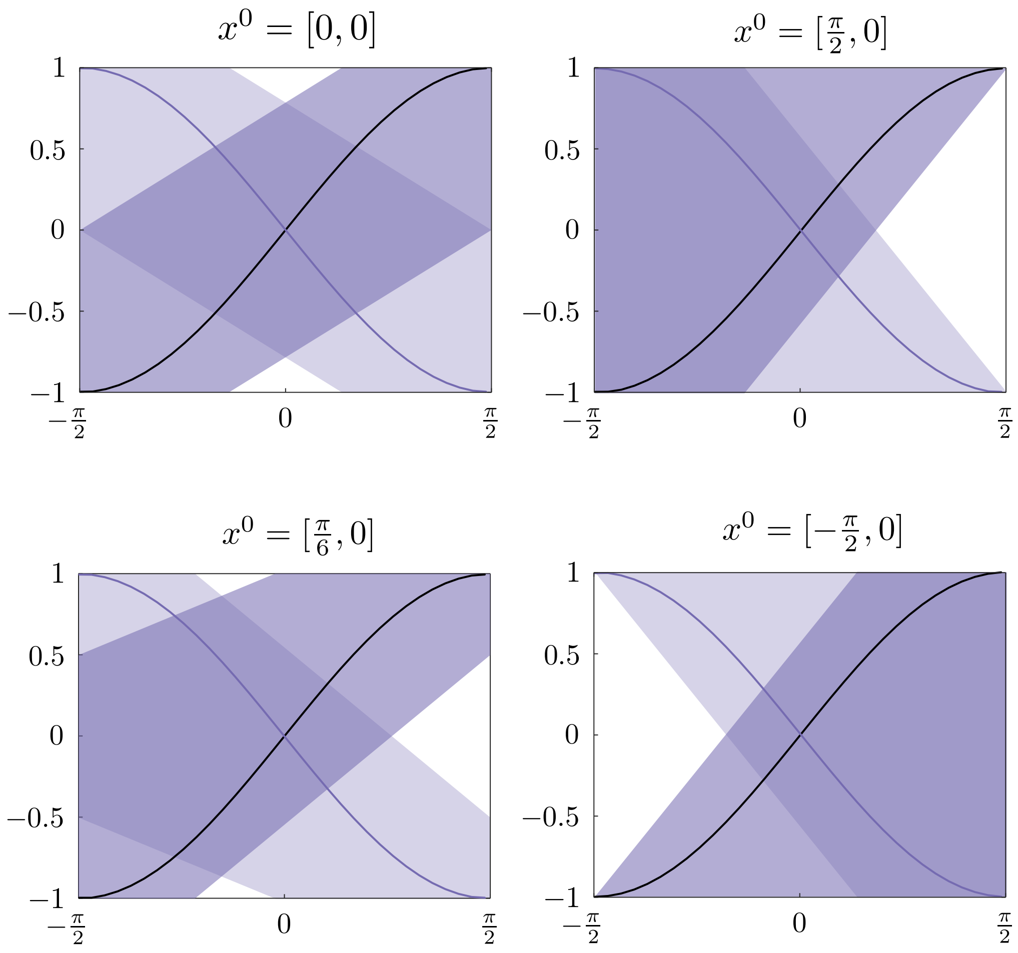

Figure 10.2: Four different linearizations depending on x^0 selection. The decreasing curve corresponds to constraint $y - \sin(x) = 0$ and the increasing curve to constraint $y - \cos(x + \pi/2) = 0$. The darker areas depicts the corresponding linearizations using the mean value form.

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].

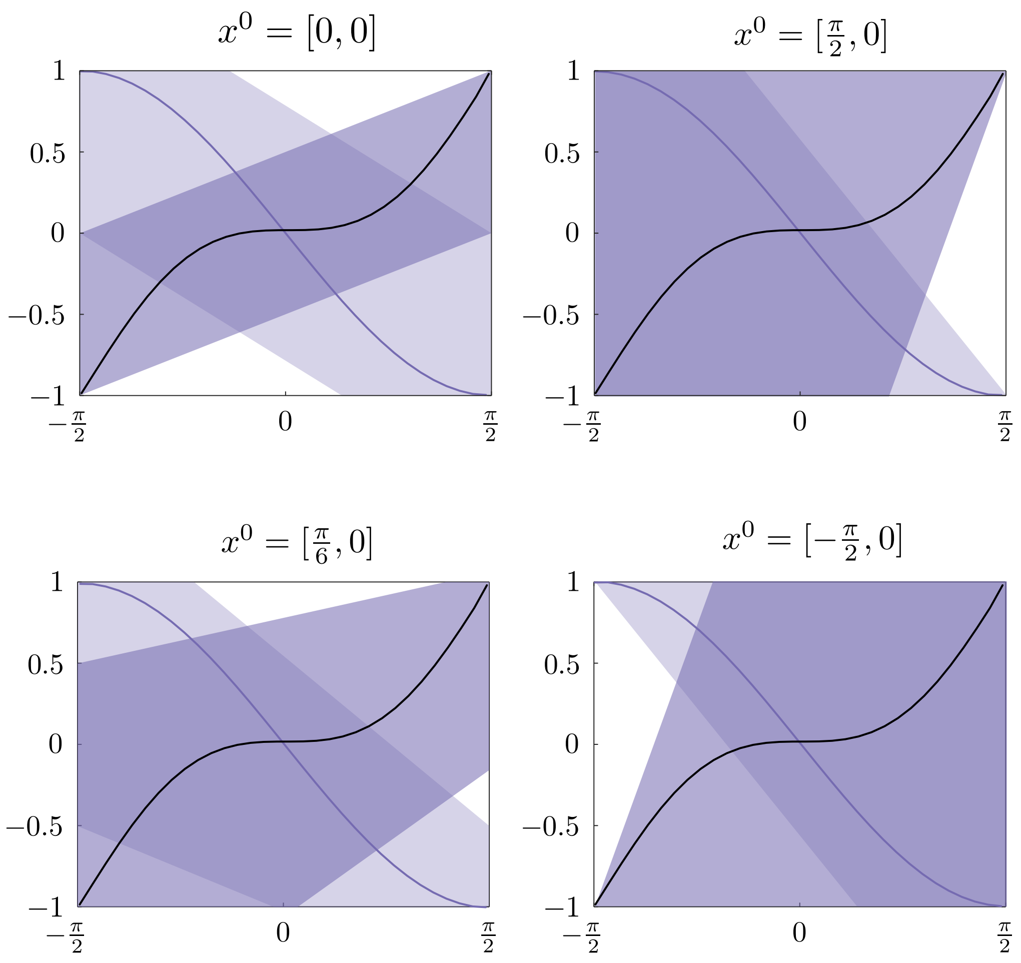

Figure 10.3: Four different linearizations depending on x^0 selection. The decreasing curve corresponds to constraint $\pi^2 y - 4x^2 \sin(x) = 0$ and the increasing curve to constraint $y - \cos(x + \pi/2) = 0$. The darker areas depicts the corresponding linearizations using slopes.

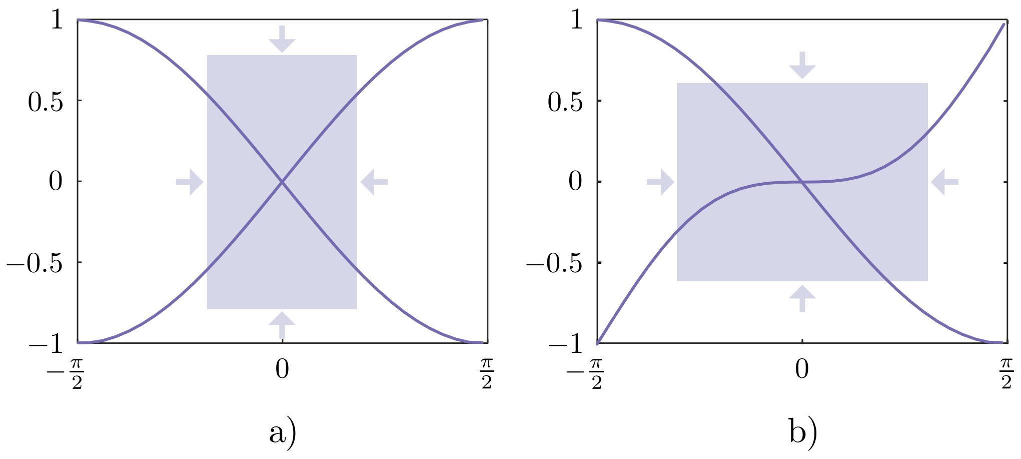

Figure 10.4: The resulting contracted boxes from the above examples; a) Example 10.6, b) Example 10.7.

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 $A = (a_1, \dots, a_n)$ (e.g., when the input is a vector or a matrix), we define $size(A) = \sum_{i=1}^{n} size(a_i)$.

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 $L := size(\underline{A}) + size(\overline{A})$. 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 $Ay \leq b$ 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:

$$\begin{aligned}
& \exists A \in \boldsymbol{A} \text{ such that } A \text{ is singular,} \\
\Leftrightarrow\quad & \exists A \in \boldsymbol{A},\ \exists x \neq 0\colon\ Ax = 0, \\
\Leftrightarrow\quad & \exists x \neq 0\colon\ -A_\Delta|x| \leq A_c x \leq A_\Delta|x|, \\
\Leftrightarrow\quad & \exists s \in \{\pm 1\}^n\ \underbrace{\exists x\colon\ -A_\Delta D_s x \leq A_c x \leq A_\Delta D_s x,\ D_s x \geq 0,\ e^T D_s x \geq 1.}_{(*)}
\end{aligned} \tag{11.2}$$

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 $size(s) \leq L$ is obvious.

The method of (11.2) is known as *orthant decomposition* since it reduces the problem to inspection of orthants $D_s x \geq 0$, 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 A $\leq$ 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 A $\leq$ B, then B is "as hard as A, or harder". If both A $\leq$ B and B $\leq$ A, then problems A, B are called *polynomially equivalent.*

The relation $\leq$ 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 A $\leq$ 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 $\leq$-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

$$\mathsf{A} \leq' \mathsf{B} \iff \mathsf{A}(x) = h(\mathsf{B}(g(x)))$$

for some polynomial-time computable functions g, h. Then the notions of $\leq'$-NP-hardness and $\leq'$-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 $\leq'$ (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