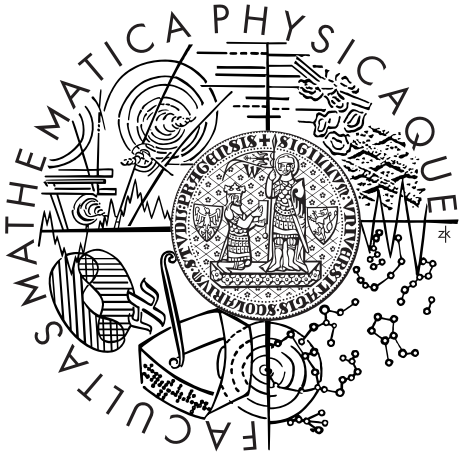
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Jaroslav Horáček

Přeurčené soustavy intervalových lineárních rovnic

Katedra aplikované matematiky

Vedoucí diplomové práce: Mgr. Milan Hladík Ph.D.
Studijní program: Informatika
Studijní obor: Teoretická informatika

Praha 2011

Děkuji především svému školiteli Mgr. Milanu Hladíkovi Ph.D. za obětavý přístup a cenné rady a konzultace, které mi vždy ochotně poskytl. Velký dík patří také mým rodičům a mému bratrovi za celkovou podporu.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Přeurčené soustavy intervalových lineárních rovnic

Autor: Jaroslav Horáček

Katedra: Katedra aplikované matematiky

Vedoucí diplomové práce: Mgr. Milan Hladík Ph.D., KAM

Abstrakt: Tato práce se zabývá přeurčenými soustavami intervalových lineárních rovnic. První část se skládá z úvodu do intervalové aritmetiky a intervalové lineární algebry a základní teorie intervalových lineárních systémů. Ve druhé části jsou popsány různé metody řešení přeurčených intervalových lineárních systémů. Řešením přeurčeného intervalového systému chápeme sjednocení všech řešení všech podsystémů. Jsou zde diskutovány známé i naše varianty algoritmů. Představíme naši vlastní metodu podčtverců. Všechny zmíněné metody jsou implementovány do jednoho toolboxu pro Matlab. Metody jsou otestovány na řešitelných a neřešitelných přeurčených systémech. Pro řešitelné systémy testujeme obálku řešení, čas a speciální vlastnosti metod. Pro neřešitelné systémy testujeme detekci neřešitelnosti. Na konci této práce poskytneme základní úvod do systému Intlab.

Klíčová slova: přeurčené systémy, intervalové lineární rovnice, intervalová analýza

Title: Overdetermined systems of interval linear equations

Author: Jaroslav Horáček

Department: Department of Applied Mathematics

Supervisor: Mgr. Milan Hladík Ph.D., KAM

Abstract: This work is focused on overdetermined systems of interval linear equations. First part consists of introduction to interval arithmetics and interval linear algebra and basic theory of interval linear systems. In the second part various methods for solving overdetermined interval linear systems are described. By solution of overdetermined interval system we mean union of all solutions of all subsystems. Known and our variants of algorithms are discussed. We introduce our subsquare method. All mentioned methods are implemented in one toolbox for Matlab. Methods are tested on solvable and unsolvable overdetermined systems. For solvable systems we test solution enclosure, time and special features of methods. For unsolvable systems we test detection of unsolvability. At the end of this work we provide basic introduction to Intlab.

Keywords: overdetermined systems, interval linear equations, interval analysis

Obsah

1. Úvod

1.1 Idea, historie a aplikace

Slovo interval se v běžné mluvě využívá zhruba třemi způsoby. Mohli bychom rozlišovat interval prázdný, plný a existenční. Prázdným intervalem můžeme chápat například mezeru či vzdálenost mezi dvěma prvky (hudební interval, interval mezi dvěma vlaky metra). Plný interval můžeme chápat jako soubor (nechceme říkat přímo množinu), pro jehož všechny prvky je splněna určitá vlastnost (funkce je spojitá na intervalu, potápěč vydržel 8 minut pod vodou). Nás však bude zajímat intervalový přístup existenční, tedy že se cosi nachází v nějakém souboru, který jsme schopni ohraničit (řešení leží v daném intervalu).

Samotná myšlenka tohoto intervalového přístupu k problémům by se dala sledovat hluboko do naší historie. Například Archimédés (287–212 př. n. l.) odhadl číslo π pomocí intervalu. Dokázal totiž pomocí mnohoúhelníků opsaných a vepsaných kružnici, že

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

Asi o 700 let později dokázal čínský matematik Cu Čchung-č', že

$$3.1415926 < \pi < 3.1415927.$$

Tento odhad se stal nejpřesnějším odhadem na budoucí téměř jedno tisíciletí.

Hlavní idea intervalového počítání je, že nepočítáme přímo s konkrétními čísly, ale s intervaly, ve kterých daná čísla určitě leží. Vezměme si například iracionální číslo $\sqrt{2}$. Toto je číslo s nekonečným desetinným rozvojem. Vždy když počítáme s tímto číslem na papíře, nedokážeme a mnohdy ani nechceme ho v nekonečném rozvoji reprezentovat. Většinou končíme s konečnou přesností na určitém desetinném řádu. Na středních školách se spokojíme s tím, že $\sqrt{2} \approx 1.41$, pro důležité fyzikální výpočty nám tato přesnost stačit nebude. Na to lze nahlížet také tak, že jsme dané číslo na papíře vlastně jen omezili nějakým malým intervalem. Například můžeme říci, že

$$\sqrt{2} = 1.41421356\ldots \in [1.41421355, 1.41421357]$$

Trochu jiná je situace ve strojové reprezentaci. Při počítání na papíře víme, že číslo $\sqrt{2}$ má nekonečný rozvoj a tušíme, že výsledná hodnota bude tudíž přesná tak, jak přesně zvolíme reprezentaci $\sqrt{2}$. Ve strojové reprezentaci kolikrát nevíme, jak přesně vnitřní reprezentace $\sqrt{2}$ odpovídá skutečnosti. Předchozí číslo není výjimkou, většinu čísel se nám nepodaří reprezentovat přesně. Díky omezenému počtu bitů pro reprezentaci dochází ke zkreslení uložených hodnot. Pokud máme k dispozici reprezentaci pomocí 64 bitů a reprezentujeme v binární soustavě, dokážeme reprezentovat maximálně 2^{64} různých čísel, což je zanedbatelný počet vzhledem k počtu reálných čísel, kterých je nespočetně mnoho. Taková čísla nemusí být nutně iracionální, například číslo 0.1 má nekonečný binární rozvoj. V praxi to vypadá tak, že nereprezentovatelné číslo se převádí určitým zaokrouhlením na číslo nejbližší reprezentovatelné. Takto však vzniká určitá chyba, která

se při větším množství opakování nastřádá do znatelných rozměrů a její důsledky mohou být katastrofální. Na stránce

`www.math.psu.edu/dna/disasters`

jsou podrobně uvedeny tři příklady selhání použití strojové reprezentace čísel. Jedná se o havárii rakety Ariane, selhání systému protiraketové obrany a zřícení ropné plošiny.

Při použití intervalové reprezentace zde určitou chybu máme také, ale narozdíl od předchozího výsledku, o kterém nejsme schopni říci, jak dobrý nebo špatný je, máme vždy jistotu, že se cílová hodnota nachází uvnitř výsledného intervalu.

Velký rozvoj intervalové matematiky nastal v době rozvoje počítačů a jejich výpočetní síly v 50. a 60. letech 20. století. Definice intervalových operací a jejich užití se tehdy objevují nezávisle v různých článcích či pracích. Ale motivace zavedení intervalového počítání se mnohdy liší. Zde jsou pro ukázku některé první práce, kde se využití intervalové aritmetiky objevilo:

- Rosaline Cecily Young se ve své knize *Mathematische annalen* z roku 1931 zabývá limitami funkcí, pro které platí $\liminf_{x\to x_0} f(x)$ a $\limsup_{x\to x_0} f(x)$ jsou různé.
- Mieczyslaw Warmus v knize *Calculus of Approximations* z roku 1956 buduje teoretický aparát pro numerické počítání.
- Ramon E. Moore ve své disertační práci z roku 1962 shrnuje základní intervalové operace, numerické řešení obyčejných diferenciálních rovnic a numerickou integraci. Ukazuje, že intervalové počítání dává rigorózní meze, ve kterých se objeví přesné výsledky.

Za počátek éry intervalového počítání je však mnohými autoritami považován rok 1966, kdy R. E. Moore vydal svou knihu *Interval Analysis*.

Intervalová analýza se rozvíjela jen velmi pozvolna. Důvodem bylo to, že výpočty s intervaly byly oproti reálným číslům pomalejší a hlavně získané intervaly, ve kterých se řešení nacházela, byly často velmi rozsáhlé – až skoro nicneříkající. To se postupem času změnilo díky usilovné vědecké činnosti v tomto oboru. Také se intervalové výpočty časem přestaly zatracovat kvůli své rychlosti, neboť jak píše Hansen v [2], otázka srovnání výpočetní rychlosti není příliš na místě, neboť reálné i intervalové počítání řeší každé různou úlohu. Reálné počítání nám dává nějaké řešení blíže neurčené přesnosti. Kdežto intervalové počítání nám dává rigorózní meze našeho řešení.

Intervalové počítání se dnes používá ve velké škále oborů. Častou oblastí aplikace je počítačová grafika a výpočetní geometrie. Pomocí něho se řeší například průniky přímek a ploch. Metoda raytracing[1] se dá nahradit intervalovou verzí. Místo několikanásobného bodového paprsku se použije jeden intervalový paprsek, čímž se značně urychlí výpočet. Další oblastí využití jsou důkazy za pomocí počítače. V roce 1998 pomocí intervalové analýzy Thomas Hales vytvořil důkaz *Keplerovy domněnky*, který se zdá býti definitivním. Keplerova domněnka je problém, který v roce 1611 zformuloval Johannes Kepler. Tvrdil, že nejhustější

[1] Grafická metoda pro vykreslování scén s použitím sledování dráhy letu paprsku světla skrz pixely.

uspořádání koulí v Eukleidovském prostoru je tím způsobem, jakým běžně trhovci na sebe skládají pomeranče. Dalšími vyřešenými problémy jsou například *Domněnka dvou bublin* či existence *Lorenzova atraktoru*. Intervalový přístup se používá také při zpřesňování fyzikálních konstant. Došlo tak například ke zpřesnění Newtonovy gravitační konstanty. Použití je však daleko širší. Dalšími oblastmi, kde se intervalové počítání používá jsou ekonomie, expertní systémy, robotika, teorie řízení či mechanika. Pro další příklady a doplňující informace je možno nahlédnout do [5], [6]. Od roku 2002 se uděluje Moorova cena za aplikace intervalového počítání. Roku 2004 získal cenu právě profesor Thomas C. Hales za důkaz výše zmíněné Keplerovy domněnky.

1.2 Cíl a struktura této práce

Intervalový přístup lze uplatnit i v lineární algebře při řešení soustav lineárních rovnic (lineárních systémů). Tato práce se konkrétně zabývá přeurčenými intervalovými lineárními systémy. Přeurčené intervalové lineární systémy jsou poměrně uzavřenou skupinou problémů. Je poměrně obtížné o nich něco dokázat, protože postrádají hezké vlastnosti čtvercových systémů (jako jsou M-matice, H-matice, diagonálně dominantní matice či matice pozitivně definitní). Proto leží lehce mimo střed zájmu. V praxi je však jejich řešení nutností, neboť se může stát, že při popisu problému dostaneme přeurčený lineární nebo nelineární systém. V následujících kapitolách poskytneme úvod do intervalového počítání. Jednak do intervalové aritmetiky samotné, poté i do intervalové lineární algebry a lineárních systémů. Následovat budou kapitoly věnované metodám řešení intervalových přeurčených lineárních systémů. Při použití termínu řešení přeurčených systémů se často vybaví metoda nejmenších čtverců. My zde budeme chápat množinu řešení jako sjednocení všech řešení jednotlivých systémů v intervalovém systému. Skupiny podobných metod jsou vždy popsány ve stejné kapitole. Každá kapitola obsahuje nejen popisy metod, ale i výsledky vědeckých článků týkající se daných metod, naše komentáře a vlastní výsledky, případně poznámky k implementaci. Součástí práce je implementace jednotlivých metod. Metody budou umístěny pohromadě do jednoho volně dostupného toolboxu pro Matlab. Na závěr porovnáme jednotlivé metody řešení z hlediska rychlosti, přesnosti a specifických vlastností pro různé typy systémů. Navrhneme případy, kdy se jednotlivé metody hodí a pokusíme se sestavit ideální metodu. Často se hodí poznat, že předložený systém je neřešitelný. Zkusíme proto též prozkoumat, pomocí kterých metod lze odhalit, že systém nemá řešení. Všechny algoritmy v naší knihovně jsou implementovány pomocí toolboxu Intlab. Intlab je toolbox pro Matlab, který umožňuje provádět intervalové výpočty a vytvářet algoritmy používajících intervalovou aritmetiku. Pokud je nám známo, v českém jazyce neexistuje žádný úvod pro použití Intlabu. V poslední kapitole proto poskytneme jednoduchý úvod do systému Intlab, ukážeme aplikace Intlabu a provedeme jeho srovnání s podobnými knihovnami nebo softwarem.

2. Intervalová aritmetika

2.1 Definice intervalu

V této kapitole definujeme základní problematiku, kterou v této práci budeme využívat – intervaly a operace na intervalech. O aritmetice samotné se později již příliš zmiňovat nebudeme, ale budeme ji automaticky používat. Nejprve zavedeme pojem reálného intervalu a jeho mezí.

Definice (Reálný interval). *Mějme $\underline{x}, \overline{x} \in \mathbb{R}$, pro které platí $\underline{x} \leq \overline{x}$, pak množinu*

$$\boldsymbol{x} = [\underline{x}, \overline{x}] = \{y \in \mathbb{R}\,;\ \underline{x} \leq y \leq \overline{x}\}$$

nazveme reálným intervalem. Číslo $\underline{x}$ nazýváme dolní mez a číslo $\overline{x}$ horní mez.

Podobně můžeme definovat i komplexní intervaly. V této práci se však omezíme pouze na intervaly reálné. Pokud platí, že $\underline{x} = \overline{x}$, říkáme, že interval je *degenerovaný*. Pokud platí, že $\underline{x} = -\overline{x}$, říkáme, že interval je *symetrický*.

Definice. *Symbolem $\mathbb{IR}$ označujeme množinu reálných intervalů. Podobně množinu všech komplexních intervalů označujeme $\mathbb{IC}$.*

Před dalším postupem ještě zavedeme několik důležitých pojmů pojících se s intervaly. Pro přehlednost je zobrazíme v tabulce.

Definice (Intervalové pojmy). *Mějme reálný interval $\boldsymbol{x}$.*

Pojem	*Značení*	*Vyjádření*
šířka $\boldsymbol{x}$	$\mathrm{w}(\boldsymbol{x})$	$\overline{x} - \underline{x}$
střed $\boldsymbol{x}$	$\mathrm{m}(\boldsymbol{x})$	$\frac{1}{2}(\underline{x} + \overline{x})$
poloměr $\boldsymbol{x}$	$\mathrm{rad}(\boldsymbol{x})$	$\frac{\mathrm{w}(\boldsymbol{x})}{2}$
mignituda $\boldsymbol{x}$	$\mathrm{mig}(\boldsymbol{x})$	$\min(\lvert\underline{x}\rvert, \lvert\overline{x}\rvert)$
magnituda $\boldsymbol{x}$	$\mathrm{mag}(\boldsymbol{x})$	$\max(\lvert\underline{x}\rvert, \lvert\overline{x}\rvert)$

Magnitudě se někdy říká i *absolutní hodnota*. Naopak absolutní hodnotou intervalu se někdy též rozumí množina $|\boldsymbol{x}| = \{|y|;\ y \in \boldsymbol{x}\}$. Jelikož intervaly jsou množiny, lze s nimi provádět množinové operace.

2.2 Množinové operace a relace

Definice (Množinové operace). *Nechť $\boldsymbol{x} = [\underline{x}, \overline{x}], \boldsymbol{y} = [\underline{y}, \overline{y}]$ jsou reálné intervaly. Průnik $\boldsymbol{x}, \boldsymbol{y}$ je prázdný pokud platí buď $\overline{y} < \underline{x}$, nebo $\overline{x} < \underline{y}$. Píšeme*

$$\boldsymbol{x} \cap \boldsymbol{y} = \emptyset.$$

V opačném případě

$$\boldsymbol{x} \cap \boldsymbol{y} = \{\max(\underline{x}, \underline{y}), \min(\overline{x}, \overline{y})\}.$$

Místo sjednocení používáme obálku ($\cup$) definovanou

$$\boldsymbol{x} \cup \boldsymbol{y} = \{\min(\underline{x}, \underline{y}), \max(\overline{x}, \overline{y}\}.$$

Obálku používáme proto, že sjednocení $\boldsymbol{x} \cup \boldsymbol{y}$ v obecném případě nemusí být interval, ale několik intervalů. Platí

$$\boldsymbol{x} \cup \boldsymbol{y} \subseteq \boldsymbol{x} \underline{\cup} \boldsymbol{y}.$$

Pro intervaly můžeme také definovat relace rovnosti a uspořádání.

Definice (Binární relace). *Nechť* $\boldsymbol{x} = [\underline{x}, \overline{x}], \boldsymbol{y} = [\underline{y}, \overline{y}]$ *jsou reálné intervaly. Platí* $\boldsymbol{x} = \boldsymbol{y}$*, jestliže*

$$\underline{x} = \underline{y} \text{ a zároveň } \overline{x} = \overline{y}.$$

Platí $\boldsymbol{x} < \boldsymbol{y}$*, jestliže*

$$\overline{x} < \underline{y}.$$

Obdobně pro relaci $\leq$*.*

2.3 Aritmetické operace

Intervalové aritmetické operace jsou definovány poměrně intuitivně, tak, aby výsledné intervaly zahrnovaly hodnoty pro všechny možné volby prvků z intervalových operandů.

Definice. *Mějme dva reálné intervaly* $\boldsymbol{x} = [\underline{x}, \overline{x}]$ *a* $\boldsymbol{y} = [\underline{y}, \overline{y}]$*. Aritmetické operace* $+, *, -, /$ *jsou definovány následovně*

$$\begin{aligned}
\boldsymbol{x} + \boldsymbol{y} &= [\underline{x} + \underline{y}, \overline{x} + \overline{y}], \\
\boldsymbol{x} - \boldsymbol{y} &= [\underline{x} - \overline{y}, \overline{x} - \underline{y}], \\
\boldsymbol{x} * \boldsymbol{y} &= [\min(S), \max(S)], \quad \textit{kde } S = \{\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy}\}, \\
\boldsymbol{x} \,/\, \boldsymbol{y} &= \boldsymbol{x} * (1/\boldsymbol{y}), \quad \textit{kde } 1/\boldsymbol{y} = [1/\overline{y}, 1/\underline{y}], 0 \notin \boldsymbol{y}.
\end{aligned}$$

U násobení není nutné počítat všechny 4 součiny. Pokud testujeme znaménka $\underline{x}, \overline{x}, \underline{y}, \overline{y}$ dostáváme 9 případů, které mohou nastat. U 8 z nich stačí spočítat jen 2 součiny. Což sice nemusí být výhodou u softwarového zpracování, díky tomu, že zde máme větší množství if–else podmínek. Hodí se však pokud je tento postup zabudován hardwarově. Při dělení předpokládáme, že interval $\boldsymbol{y}$ neobsahuje 0. Existuje i takzvaná *rozšířená intervalová aritmetika*, která povoluje i některé nestandardní operace jako dělení tímto intervalem. Více je možné dozvědět se v [2].

Nutno podotknout, že pro operace, tak jak jsme se definovali, množina $\mathbb{IR}$ není těleso. Přesto pro ní některé vlastnosti tělesa platí. Existuje zde nulový prvek $\mathbf{0} \equiv [0, 0]$ a jednotkový prvek $\mathbf{1} \equiv [1, 1]$. Tyto prvky se nerovnají a pro všechny intervaly $\boldsymbol{x}$ platí

$$\mathbf{0} + \boldsymbol{x} = \boldsymbol{x}$$

$$\mathbf{1} * \boldsymbol{x} = \boldsymbol{x}$$

$$\mathbf{0} * \boldsymbol{x} = \mathbf{0}$$

Násobení i sčítání je komutativní i asociativní.

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

Další vlastnosti již nemusí platit. Předně selhává existence inverzního a opačného prvku k intervalu X.

Pozorování. *Pro nedegenerovaný reálný interval* $\boldsymbol{x} = [\underline{x}, \overline{x}]$ *neexistuje opačný prvek.*

Důkaz. Podívejme se, jak by musel vypadat opačný prvek k reálnému intervalu $\boldsymbol{x}$. Nazvěme ho $\boldsymbol{y}$.

$$\mathbf{0} = [0,0] = \boldsymbol{x} + \boldsymbol{y} = [\underline{x}, \overline{x}] + [\underline{y}, \overline{y}] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}].$$

Pak ale musí platit $\underline{x} + \underline{y} = 0$ a $\overline{x} + \overline{y} = 0$. Z čehož dostáváme

$$\underline{y} = -\underline{x},\ \overline{y} = -\overline{x}.$$

Tedy interval $\boldsymbol{y}$ vypadá

$$\boldsymbol{y} = [-\underline{x}, -\overline{x}].$$

Což nesplňuje naší definici reálného intervalu, neboť jelikož v původním intervalu platí $\underline{x} \leq \overline{x}$, platí $-\underline{x} \geq -\overline{x}$. Definice je splněna jedině pro $\underline{x} = \overline{x}$, tedy pro degenerovaný interval. □

Existují však definice intervalové aritmetiky, které povolují mít intervaly se zcela libovolnou horní i dolní mezí. My je však v této práci nebudeme brát v úvahu.

Zároveň s neexistencí opačného prvku neplatí distributivita pro násobení. Tedy v obecném případě

$$\boldsymbol{x}(\boldsymbol{y} + \boldsymbol{z}) \neq \boldsymbol{x}\boldsymbol{y} + \boldsymbol{x}\boldsymbol{z}.$$

Například pro $\boldsymbol{x} = [1,2], \boldsymbol{y} = [1,1], \boldsymbol{z} = [-1,-1]$ levá strana vychází $[0,0]$ a pravá $[-1,1]$. Nicméně vždy platí

$$\boldsymbol{x}(\boldsymbol{y} + \boldsymbol{z}) \subseteq \boldsymbol{x}\boldsymbol{y} + \boldsymbol{x}\boldsymbol{z}.$$

2.4 Funkce

Do teď jsme používali intervaly v aritmetických operacích, množinových operacích i v binárních relacích uspořádání a rovnosti. Podívejme se, jak vypadá aplikace funkcí na intervaly.

Definice. *Mějme reálný interval* $\boldsymbol{x}$ *a zobrazení* $f : \mathbb{R} \to \mathbb{R}$. *Pak*

$$f(\boldsymbol{x}) = \{f(y);\ y \in \boldsymbol{x}\}.$$

Do některých funkcí lze přímo dosadit, například do monotónních funkcí. Pro $\boldsymbol{x} = [\underline{x}, \overline{x}]$ a funkci exp můžeme přímo dosadit dolní a horní mez

$$\exp(\boldsymbol{x}) = [\exp(\underline{x}), \exp(\overline{x})]$$

pro libovolný interval $\boldsymbol{x} \in \mathbb{IR}$. Funkce exp je rostoucí funkce. Kdyby funkce byla klesající, musíme krajní meze prohodit. Obecně spočítání mezí není vždy takto jednoduché. Pro větší detaily je možné nahlédnout do [2], [6].

2.5 Úskalí intervalové aritmetiky

Počítání s intervaly představuje několik úskalí. Jedním z nich je takzvaná *závislost*. Jestliže se na některých místech matematického výrazu objeví intervaly, znamená to, že tento výraz reprezentuje vlastně množinu výrazů. A to takových, které vzniknou postupným výběrem všech hodnot ze všech intervalů. Jelikož hodnoty z intervalů volíme nezávisle na sobě, může se nám stát, že při nevhodném matematickém zápisu dostaneme množinu výrazů větší než by nutně musela být. Tím pádem číselná hodnota výrazu může být taktéž nadhodnocená. Nemusíme hledat složité příklady. Stačí vzít interval $\boldsymbol{x} = [-2, 1]$ a funkci $f_1(x) = x^2$. Matematicky ekvivalentní zápis této funkce je $f_2(x) = x * x$. Pak ale dostáváme

$$\begin{aligned} f_1(\boldsymbol{x}) =& f_1([-2, 1]) = [-2, 1]^2 = [0, 4], \\ f_2(\boldsymbol{x}) =& f_2([-2, 1]) = [-2, 1] * [-2, 1] = [-2, 4]. \end{aligned}$$

Vidíme, že $f_1(\boldsymbol{x}) \subseteq f_2(\boldsymbol{x})$. Tento případ nastal díky tomu, že hodnoty z jednotlivých intervalů vybíráme nezávisle. Naším cílem by tedy mělo být, aby se ve výsledném výrazu nacházelo, co nejméně proměnných, za které budeme dosazovat intervaly. Například funkci

$$f(x) = \frac{x}{x + 1}$$

bude vhodnější (pokud to hodnoty dosazované za x dovolí, neboť obě funkce mají jiný definiční obor) přepsat na

$$f(x) = \frac{1}{1 + \frac{1}{x}}.$$

Toto jsou důvody, proč nelze libovolný neintervalový algoritmus přepsat na intervalou verzi prostým nahrazením floating point operací intervalovými operacemi. Při přechodu k intervalovým algoritmům je proto zapotřebí opatrnosti. V další kapitole se budeme věnovat intervalovým vektorům a maticím a poskytneme úvod do intervalové lineární algebry.

3. Intervalová lineární algebra

3.1 Použité značení

Nejprve bude vhodné zmínit se o použitém značení v této práci. Písmenem $\mathbb{R}$ značíme těleso reálných čísel. V naší práci se budeme zabývat převážně reálnými čísly a objekty vybudovanými nad reálnými čísly. Netučnými písmeny A, B, C značíme matice o rozměrech $m \times n$ nad tělesem $\mathbb{R}$. Prvky matice A na pozici (i, j) značíme $a_{i,j}$ a říkáme jim *koeficienty*. Pro dvě matice A, B stejných rozměrů definujeme relace $\leq, <$ následovně

$$A \leq B \text{ (resp. } A < B), \text{ jestliže platí } a_{i,j} \leq b_{i,j} \text{ (resp. } a_{i,j} < b_{i,j}) \text{ pro všechna } i, j.$$

Písmeno $E \in \mathbb{R}^{m \times n}$ značí jedničkovou matici (matici složenou ze samých jedniček). Písmenem $e \in \mathbb{R}^m$ značíme vektor složený ze samých jedniček. Symbolem I_n značíme jednotkovou matici řádu n a e_j značíme její j-tý sloupec.

Absolutní hodnotu matice $A = (a_{i,j})$ definujeme opět po jednotlivých koeficientech $|A| = (|a_{i,j}|)$.

Vektor b budeme chápat jako matici tvaru $m \times 1$. Pro relace $\leq, <$ a absolutní hodnotu $|.|$ platí tudíž stejná definice jako u matic. Pro každý vektor $x \in \mathbb{R}^n$ definujeme jeho znaménkový vektor $\operatorname{sgn}(x)$ jako

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

Pro daný vektor $x \in \mathbb{R}^n$ značíme

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

Často se používá množina $\{\pm 1\}^n$ (značí se také Y_n), což je množina všech vektorů délky n, které se skládají pouze z prvků 1 nebo -1. Tato množina se dá jednoduše generovat pomocí libovolného počátečního $(-1/1)$ vektoru $p \neq e$. Z něj jsme schopni vygenerovat celou množinu Y_n přímo v 2^n krocích, tak, že v každém kroku dostaneme nový vektor změnou vždy jen jednoho koeficientu předešlého vektoru. Algoritmem, kterým lze Y_n generovat, se budeme více zabývat v kapitole o lineárním programování.

Důležitým pojmem, který použijeme, je spektrální poloměr matice. Pro matici A ho definujeme

$$\varrho(A) = \max\{|\lambda|;\ \lambda \text{ je vlastní číslo } A\}.$$

3.2 Značení intervalových matic

Dosavadní značení se týkalo matic klasických. Nyní přejdeme k maticím intervalovým. Intervalové matice můžeme vyjádřit dvěma způsoby. První způsob je pomocí horní a dolní meze.

Definice. *Nechť matice* $\underline{A}, \overline{A}$ *jsou dvě matice z* $\mathbb{R}^{m\times n}$*, takové že* $\underline{A} \leq \overline{A}$*. Pak množině*

$$\boldsymbol{A} = [\underline{A}, \overline{A}] = \{A;\ \underline{A} \leq A \leq \overline{A}\}$$

říkáme intervalová matice. Maticím $\underline{A}, \overline{A}$ *říkáme horní a dolní mez.*

Výše zmíněná nerovnost platí pro každý koeficient matice A. Intervalovou matici $\boldsymbol{A}$ je také možné chápat jako množinu všech matic, jejichž koeficienty tvoří všechny možné prvky z intervalových koeficientů matice $\boldsymbol{A}$. Nutno podotknout, že jednotlivé prvky vybíráme z intervalů nezávisle. Je zřejmé, že pokud $\underline{A} = \overline{A}$ dostáváme klasickou neintervalovou matici. Někdy se intervalové matice značí A^I a podobně intervalové vektory b^I. My zde budeme intervalové matice a vektory značit tučně – $\boldsymbol{A}$ a $\boldsymbol{b}$.

Druhým způsobem, jak lze intervalovou matici vyjádřit, je pomocí *středové matice* A^c a *matice poloměru* A^Δ. Pak intervalovou matici definujeme

$$\boldsymbol{A} = \{A;\ |A - A^c| \leq A^\Delta\}.$$

Ze znalosti horní a dolní meze intervalové matice se dá středová matice a matice poloměru snadno odvodit

$$A^c = \frac{1}{2}(\underline{A} + \overline{A}),$$
$$A^\Delta = \frac{1}{2}(\overline{A} - \underline{A}).$$

Podobně můžeme ze znalosti poloměru a středové matice odvodit horní a dolní mez

$$\underline{A} = A^c - A^\Delta,$$
$$\overline{A} = A^c + A^\Delta.$$

Obě formy zápisu budeme používat. Pro některé popisy problémů a důkazy je vhodnější první forma zápisu, pro některé se více hodí forma druhá.

Mějme intervalovou matici $\boldsymbol{A} = [A^c - A^\Delta, A^c + A^\Delta]$ o rozměrech $m \times n$, pak definujeme matici

$$A_{yz} = A^c - D_y A^\Delta D_z$$

pro všechny $y \in \{\pm 1\}^m$ a $z \in \{\pm 1\}^n$. Při podrobnějším zkoumání z definice plyne

$$(A_{yz})_{ij} = (A^c)_{ij} - y_i A^\Delta_{ij} z_j = \begin{cases} \overline{a}_{ij}, & \text{jestliže } y_i z_j = -1, \\ \underline{a}_{ij}, & \text{jestliže } y_i z_j = 1, \end{cases}$$

pro každé $(i = 1, \ldots, m, j = 1, \ldots, n)$. Tedy pro každé výše definované y, z matice $A_{yz} \in \boldsymbol{A}$. Podle definice platí $A_{yz} = A_{-y,-z}$. Jak později uvidíme, mnoho problémů lze díky této matici konečně popsat (A_{yz} je předpisem pro 2^{m+n-1} matic).

3.3 Regularita intervalových matic

V této podkapitole se budeme věnovat regularitě intervalových matic. Nejprve definujeme, co znamená regulární a singulární intervalová matice.

Definice. *Čtvercová intervalová matice $\boldsymbol{A}$ se nazývá regulární, jestliže pro každou reálnou matici $A \in \boldsymbol{A}$ platí, že je regulární.*

Definice. *Čtvercová intervalová matice $\boldsymbol{A}$ se nazývá singulární, jestliže existuje reálná matice $A \in \boldsymbol{A}$, která je singulární.*

Existuje mnoho postačujících a nutných podmínek pro regularitu nebo singularitu intervalových matic. Podmínky regularity i singularity lze vztáhnout k různým oblastem lineární algebry. K vlastním číslům, řešení rovnic, determinantům, apod. Profesor Jiří Rohn sepsal seznam 40 nutných a postačujících podmínek regularity v [12]. Následující je také z tohoto seznamu.

Nutná a postačující podmínka regularity. *Intervalová matice*

$$\boldsymbol{A} = [A^c - A^\Delta, A^c + A^\Delta]$$

je regulární právě tehdy, když nerovnice

$$|A^c x| \leq A^\Delta |x|$$

má pouze triviální řešení.

Je vidět, že toto tvrzení se pro praktické výpočty příliš nehodí. Naštěstí existují dvě přívětivější podmínky regularity a singularity dohledatelné například v [8].

Postačující podmínka regularity. *Intervalová matice $\boldsymbol{A} = [A^c - A^\Delta, A^c + A^\Delta]$ je regulární, jestliže je A^c regulární a zároveň platí*

$$\varrho(|(A^c)^{-1}|A^\Delta) < 1.$$

Postačující podmínka singularity. *Intervalová matice $\boldsymbol{A} = [A^c - A^\Delta, A^c + A^\Delta]$ je singulární, jestliže je A^c regulární a zároveň platí*

$$\max_j (|(A^c)^{-1}|A^\Delta)_{jj} \geq 1.$$

Druhá podmínka je evidentně ověřitelná v polynomiálním čase. První podmínka je ekvivalentní s $(I - |(A^c)^{-1}|A^\Delta)^{-1} \geq 0$. Tato podmínka je také ověřitelná v polynomiálním čase. Dostáváme tedy podmínku pro regularitu a singularitu, které jsou v praxi použitelnější. V algoritmech testujících regularitu intervalových matic se nejprve testují podobné podmínky.

3.4 Inverzní intervalové matice

Dalším klasickým problémem je výpočet inverzní matice. Definice inverzní intervalové matice je poměrně intuitivní.

Definice. *Pro regulární čtvercovou intervalovou matici* $\boldsymbol{A}$ *definujeme intervalovou inverzní matici předpisem* $\boldsymbol{A}^{-1} = [\underline{B}, \overline{B}]$*, kde*

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

Získání přesné inverzní intervalové matice není jednoduchá záležitost. Důvodem tomu je, že v klasickém neintervalovém případě se změna jednoho koeficientu čtvercové matice v inverzní matici projeví nelineárně a ještě k tomu ve více koeficientech. Tedy dvě velmi blízké matice z nějaké intervalové matice mohou mít inverzní matice značně odlišné. Lze předpokládat (ale i dokázat), že výpočet inverzní matice je NP-těžký problém. Vypočítat se dá pomocí následující formule.

Předpis (inverzní intervalová matice). *Nechť* $\boldsymbol{A}$ *je regulární intervalová matice. Pak její inverzní matici* $\boldsymbol{A}^{-1} = [\underline{B}, \overline{B}]$ *vypočteme*

$$\underline{B}_{ij} = \min_{y,z \in Y_n} (A^{-1}_{yz})_{ij},$$

$$\overline{B}_{ij} = \max_{y,z \in Y_n} (A^{-1}_{yz})_{ij}.$$

Pokud se spokojíme s trochu nadhodnocenou inverzní maticí, můžeme ji spočítat v polynomiálním čase s pomocí intervalové Gauss-Jordanovy eliminace či řešení intervalových lineárních rovnic. Nebo výpočetně méně náročně s použitím pouze dvakrát operace inverze. Toto bylo dokázáno Rohnem, Hansenem a Bliekem. Jejich předpis dává překvapivě dobré výsledky. Odkaz na něj je možné dohledat v [9].

Předpis(Hansen-Rohn-Bliek). *Nechť* $\boldsymbol{A} = [A^c - A^\Delta, A^c + A^\Delta]$ *splňuje*

$$\varrho(|(A^c)^{-1}|A^\Delta) < 1.$$

Potom dostáváme

$$\boldsymbol{A}^{-1} \subseteq [\min\{\underline{\underline{B}}, D_\nu \underline{\underline{B}}\}, \max\{\overline{\overline{B}}, D_\nu \overline{\overline{B}}\}],$$

kde

$$\begin{aligned}
M &= (I - |(A^c)^{-1}|A^\Delta)^{-1}, \\
\mu &= (M_{11}, \ldots, M_{nn})^T, \\
D_\nu &= (2D_\mu - I)^{-1}, \\
\underline{\underline{B}} &= -M|(A^c)^{-1}| + D_\mu((A^c)^{-1} + |(A^c)^{-1}|), \\
\overline{\overline{B}} &= M|(A^c)^{-1}| + D_\mu((A^c)^{-1} - |(A^c)^{-1}|).
\end{aligned}$$

Užitečná je následující definice a s ní se pojící postačující i nutná podmínka. Obě jsou dohledatelné v [9].

Definice. *Regulární matici* $\boldsymbol{A}$ *nazýváme inverz-nezápornou, pokud pro každou* $A \in \boldsymbol{A}$ *platí* $A^{-1} \geq 0$*.*

Postačující a nutná podmínka. *Čtvercová intervalová matice* $\boldsymbol{A} = [\underline{A}, \overline{A}]$ *je inverz-nezáporná právě tehdy, když* $\underline{A}^{-1} \geq 0$ *a* $\overline{A}^{-1} \geq 0$*.*

To má několik výhod. Například pokud je $\boldsymbol{A} = [\underline{A}, \overline{A}]$ inverz-nezáporná, dostáváme přímo $\boldsymbol{A}^{-1} = [\underline{A}^{-1}, \overline{A}^{-1}]$. Také z této podmínky přímo vyplývá, že pokud $(\underline{A})^{-1} \geq 0$ a $(\overline{A})^{-1} \geq 0$, tak $\boldsymbol{A}$ je regulární.

4. Intervalové lineární systémy

4.1 Základní teorie

Definice. *Nechť* $\boldsymbol{A} \in \mathbb{IR}^{m\times n}$ *je intervalová matice a* $\boldsymbol{b} \in \mathbb{IR}^{m}$ *je intervalový vektor. Předpisu* $\boldsymbol{A}x = \boldsymbol{b}$ *říkáme intervalový lineární systém a rozumíme jím množinu*

$$\{Ax = b\,;\ x \in \mathbb{R}^n, A \in \boldsymbol{A}, b \in \boldsymbol{b}\}.$$

Místo pojmu intervalový lineární systém budeme též někdy používat pojem intervalová lineární soustava.

Definice (Množina řešení). *Nechť* $\boldsymbol{A}x = \boldsymbol{b}$ *je intervalový lineární systém. Pak množinu*

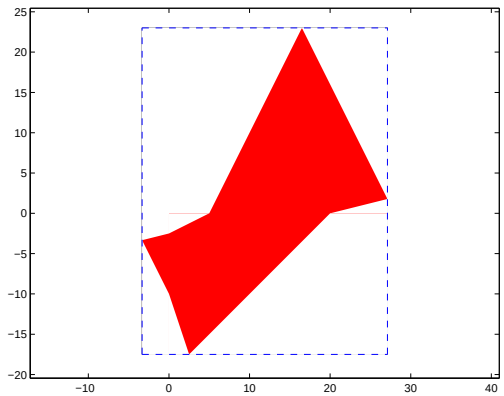
$$S = \{x;\ Ax = b \text{ pro nějaké } A \in \boldsymbol{A}, b \in \boldsymbol{b}\}$$

nazýváme množinou řešení intervalového lineárního systému $\boldsymbol{A}x = \boldsymbol{b}$.

Množinou řešení intervalového lineárního systému budeme tedy chápat sjednocení všech řešení všech reálných systémů obsažených v intervalovém systému. Množina řešení nemusí být konvexní, ale tvoří konvexní množinu v každém ortantu. Příkladem může být například systém

$$\begin{pmatrix} [5,\ 10] & [\text{-}20,\ \text{-}5] \\ [10,\ 15] & [5,\ 10] \end{pmatrix} x = \begin{pmatrix} [50,\ 100] \\ [\text{-}50,\ 280] \end{pmatrix},$$

jehož množina řešení je graficky znázorněna na obrázku 4.1.



Obrázek 4.1: Množina řešení intervalového systému

Obrázek 4.1 ukazuje také to, že množina S může nabývat složitých tvarů, které nelze jednoduše popsat. Často je proto cílem nalézt n-rozměrný kvádr (angl. box), ve kterém daná množina řešení leží (na obrázku je zobrazen přerušovanou čárou). Tento kvádr lze popsat intervalovým vektorem. Jinými slovy řečeno, hledáme intervalový vektor, který by co nejtěsněji ohraničil množinu řešení S. Takovému intervalovému vektoru budeme říkat *intervalový obal* (angl. interval hull).

Definice (Intervalový obal). *Nechť S je omezená neprázdná množina řešení intervalového lineárního systému $\boldsymbol{A}x = \boldsymbol{b}$, kde $\boldsymbol{A}$ je matice rozměrů $m \times n$. Pak intervalovému vektoru $\boldsymbol{h} = [\underline{h}, \overline{h}]$, definovanému*

$$\begin{aligned} \underline{h}_i &= \min_{x \in S} x_i, \\ \overline{h}_i &= \max_{x \in S} x_i, \qquad (i = 1 \ldots n) \end{aligned}$$

říkáme intervalový obal (interval hull).

Pro intervalový lineární systém obecně platí

1. Rozhodnout, zda je jeho množina řešení neprázdná, je NP-těžký problém.
2. Výpočet intervalového obalu jeho množiny řešení je NP-těžký problém.
3. Výpočet libovolné ϵ-aproximace intervalového obalu je NP-těžký problém.

Důkaz 1. lze nalézt v [1]. Odkazy na důkazy 2. a 3. lze dohledat v [4]. U některých speciálních typů matic lze přesto intervalový obal spočítat efektivně. Takové matice jsou například čtvercové M-matice, nebo matice diagonálně dominantní. Pokud matice nejsou ve výhodném tvaru, nebo se nejedná o čtvercové matice, spokojíme se s nějakou (pokud možno, co nejtěsnější) nadmnožinou či podmnožinou množiny S. Česky ji budeme nazývat *intervalová obálka* (angl. interval enclosure).

Definice (Vnitřní intervalová obálka). *Nechť S je množina řešení intervalového lineárního systému $\boldsymbol{A}x = \boldsymbol{b}$. Intervalovému vektoru $[\underline{x}, \overline{x}]$ říkáme vnitřní obálka množiny S, jestliže platí*

$$[\underline{x}, \overline{x}] \subseteq S.$$

Definice (Vnější intervalová obálka). *Nechť S je množina řešení intervalového lineárního systému $\boldsymbol{A}x = \boldsymbol{b}$. Intervalovému vektoru $[\underline{x}, \overline{x}]$ říkáme vnější obálka množiny S, jestliže platí*

$$S \subseteq [\underline{x}, \overline{x}].$$

Množinu řešení můžeme popsat i jinými způsoby. Pokud chceme dosáhnout přesnějších výsledků, můžeme ji popsat jako soubor boxů v prostoru. Tímto způsobem popisu se však v naší práci nebudeme zabývat. Dále nás bude zajímat pouze dosažení co nejtěsnější vnější obálky množiny řešení S. V textu budeme používat pojem intervalová obálka výhradně ve smyslu vnější intervalové obálky.

V první řadě má smysl si pokládat otázku, zda je daný systém $\boldsymbol{A}x = \boldsymbol{b}$ vůbec řešitelný. Dokonce se v intervalovém případě můžeme ptát na několik typů řešitelnosti.

Definice (Řešitelnost). *Řekneme, že intervalový lineární systém $\boldsymbol{A}x = \boldsymbol{b}$ je silně řešitelný, pokud pro každé $A \in \boldsymbol{A}$ a $b \in \boldsymbol{b}$ existuje x_0 tak, že platí $Ax_0 = b$.*

Řekneme, že intervalový lineární systém $\boldsymbol{A}x = \boldsymbol{b}$ je slabě řešitelný, pokud pro nějaké $A \in \boldsymbol{A}$ a $b \in \boldsymbol{b}$ existuje x_0 tak, že platí $Ax_0 = b$.

Pro úplnost definujeme řešení intervalového lineárního systému.

Definice (Řešení). *Reálný vektor x se nazývá (slabé) řešení intervalového lineárního systému, pokud platí*

$$Ax = b \text{ pro nějaké } A \in \boldsymbol{A}, b \in \boldsymbol{b}.$$

(Nebo jednoduše pokud platí $x \in S$).

Pojmy jako řešení, množina řešení a řešitelnost můžeme podobně definovat i pro systémy nerovnic. Jimi se v této práci příliš zabývat nebudeme, proto příslušné definice neuvádíme. Při srovnání oblasti intervalových systémů rovnic a nerovnic musíme uvést, že zde dochází k zajímavému paradoxu. Zatímco většina problémů týkající se řešitelnosti systémů rovnic je NP-těžkých, u systémů nerovnic jsou tyto problémy ve většině případů polynomiální.

4.2 Přeurčené systémy

Definice. *O intervalovém lineárním systému $\boldsymbol{A}x = \boldsymbol{b}$, kde $\boldsymbol{A} \in \mathbb{U}^{m \times n}$ a $\boldsymbol{b} \in \mathbb{U}^m$ řekneme, že je přeurčený (angl. overdetermined), jestliže $m > n$. ($\mathbb{U}$ může být třeba $\mathbb{R}$, $\mathbb{C}$, $\mathbb{IR}$ nebo $\mathbb{IC}$).*

Jednoduše řečeno, přeurčený systém lineárních rovnic obsahuje více rovnic než proměnných. V této práci se zabýváme intervalovými lineárními systémy přeurčenými.

Neintervalový přeurčený systém $Ax = b$ nemá řešení, má právě jedno řešení nebo má nekonečně mnoho řešení v závisloti na počtu nezávislých řádků matice $[A|b]$. Podobně je tomu v případě intervalovém. Avšak s jedním rozdílem. Díky tomu, že koeficienty intervalové matice jsou intervaly a výběr hodnoty z jednoho koeficientu nezávisí na hodnotě z žádného jiného koeficientu, dostáváme pro různý výběr hodnot různé závislostní vztahy mezi řádky konkrétní vybrané matice $[A|b]$. Například pro intervalové $\boldsymbol{A}, \boldsymbol{b}$, jejichž jednotlivé koeficienty leží v intervalu

$$[1 - 0.1, 1 + 0.1],$$

systém

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} x = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \text{ má právě jedno řešení.}$$

Zatímco systém

$$\begin{pmatrix} 1 \\ 1.0001 \end{pmatrix} x = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \text{ nemá žádné řešení.}$$

Pro čtvercové systémy existuje daleko větší škála algoritmů, jejich různých vylepšení a variant. Je to proto, že o čtvercových systémech lze daleko více teoreticky říci. Čtvercové matice se chovají mnohdy výhodně a jejich chování lze tudíž lépe popsat. Nyní bude následovat série kapitol, ve kterých se podíváme na různé metody hledání obálky intervalových přeurčených systémů. Často se pod danou metodou skrývá celá skupina podobných algoritmů. Tuto skupinu podobných metod vždy popíšeme v samostatné kapitole. Každá metoda bude podrobně vysvětlena a doplněna našimi komentáři a úpravami, případně poznámkami ohledně implementace.

5. Gaussova eliminace

5.1 Druhy Gaussovy eliminace

Uvažme nejprve případ, kdy je matice A čtvercová neintervalová a b je neintervalový vektor. Mějme dánu soustavu $Ax = b$ a chceme získat její řešení. Velmi užitečným a všestranným nástrojem je Gaussova eliminace. Pod pojmem Gaussova eliminace (GE) se často chápe několik různých algoritmů. V zahraniční literatuře lze často pod tímto pojmem nalézt to, čemu se jinak říká LU-rozklad. My zde Gaussovu eliminaci budeme chápat stejným způsobem, kterým se běžně vyučuje na českých středních a vysokých školách. Gaussova eliminace převádí matici do odstupňovaného tvaru (pro regulární matici do horního trojúhelníkového tvaru). Následující algoritmus provádí GE pro regulární matice. Navíc ještě zanechává na diagonále prvky 1. Pokud nalezneme sloupec, ve kterém jsou samé nuly, končíme. Matice nemá v tomto sloupci na diagonále nenulový prvek (pivota) a tudíž systém s touto maticí nemá omezené řešení (hodnost matice je menší než počet řádků, matice je sigulární). Nás budou zajímat hlavně systémy, jejichž řešení bude možné konečně omezit.

Algoritmus (GE). *Mějme dánu lineární soustavu $Ax = b$, kde matice A je rozměru $n \times n$ a b je vektor.*

1. *Sestavíme rozšířenou matici soustavy $[A \,|\, b]$ a položíme $k = 1$.*
2. *Pokud je $a_{ik} = 0$ pro všechna $i \geq k$ skončíme, A je singulární.*
3. *Jinak nalezneme $a_{ik} \neq 0$, $i \geq k$ a vyměníme i-tý a k-tý řádek.*
4. *Přenásobíme k-tý řádek $\frac{1}{a_{kk}}$, nyní je na pozici pivota hodnota 1.*
5. *Pro každý i-tý řádek $i > k$ položme $A_{i*} = A_{i*} - a_{ik}A_{k*}$.*
6. *Položme $k = k + 1$ a pokud $k \leq n$ jdeme zpět na krok 2.*

Pro získání řešení ze soustavy v odstupňovaném tvaru použijeme zpětnou substituci.

Algoritmus (Zpětná substituce). *Mějme lineární soustavu $Ax = b$, kde matice A má rozměry $n \times n$ a je v horním trojúhelníkovém tvaru a na diagonále má samé jedničky. Chceme nalézt vektor řešení $x = (x_1, \ldots, x_n)$.*

1. *Položme $k = n$.*
2. *$x_k = b_k - \sum_{j=k+1}^{n} a_{kj}x_j$.*
3. *Položme $k = k - 1$, pokud $k > 0$ opakujeme od kroku 2.*
4. *$x = (x_1, \ldots, x_n)$ je výsledným řešením.*

Můžeme rozlišovat různé typy Gaussovy eliminace podle způsobu výběru pivota – pivotizace. Máme tři možnosti

- Bez pivotizace
- S částečnou pivotizací
- S úplnou pivotizací

Bez pivotizace znamená, že nevybíráme konkrétního pivota. Matici upravujeme eliminačními operacemi, tak jak jsme ji dostali na vstupu. Tento postup však může selhat, když se nám octne na pozici A_{ii} číslo 0, kterým musíme dělit. S částečnou pivotizací znamená, že vybíráme takový řádek, který má potenciálního pivota nenulového a zároveň takový, aby se na pozici pivota dostal prvek s největší absolutní hodnotou. Nutno poznamenat, že výměna řádků je ekvivalentní úprava, která nemění množinu řešení. Při postupu s úplnou pivotizací vyměňujeme navíc i sloupce matice A. To je také ekvivalentní úprava pouze v případě, že současně prohodíme i příslušné prvky vektoru řešení x. U tohoto přístupu je potřeba provést větší množství floating point operací. I když je tento přístup stabilnější, v praxi se příliš nepoužívá.

Gaussovy eliminace jsou jednoduché algoritmy, které lze s menšímy úpravami přenést do prostředí intervalů. Budeme k tomu potřebovat intervalovou aritmetiku, tak jak jsme ji zavedli v úvodu. Ta popisuje jak pracují aritmetické operace pro dva intervaly, z nichž obě hodnoty bereme nezávisle na sobě. Navíc se nám zde bude hodit malé rozšíření této aritmetiky, a to pro případy, kdy na obou stranách operace stojí jeden a týž interval a hodnoty z něj bereme závisle.

$$\begin{aligned}
\boldsymbol{x} + \boldsymbol{x} &= [\underline{x} + \underline{x}, \overline{x} + \overline{x}], \\
\boldsymbol{x} - \boldsymbol{x} &= [0, 0], \\
\boldsymbol{x} * \boldsymbol{x} &= [\underline{x} * \underline{x}, \overline{x} * \overline{x}], \\
\boldsymbol{x} \ / \ \boldsymbol{x} &= [1, 1], \text{ kde } \boldsymbol{x} \text{ neobsahuje } 0.
\end{aligned}$$

Toto se nám bude hodit zvláště v kroku číslo 4. a 5. algoritmu (GE). I když se na první pohled zdá, že se nejedná o ekvivalentní intervalové úpravy, v těchto krocích je můžeme použít. Můžeme se totiž na intervalový systém dívat jako na množinu všech neintervalových systémů, které obsahuje. Provedeme kroky 4. a 5. na jednotlivých systémech a složíme je nazpět do intervalového systému. Je to vlastně totéž, jako bychom aplikovali intervalové operace, které jsme výše dodefinovali. Nyní uvedeme algoritmus Gaussovy eliminace pro intervalový případ. Bude se jednat o tentýž algoritmus uvedený výše s drobnými změnami.

Algoritmus (Intervalová GE). *Mějme dánu intervalovou lineární soustavu* $\boldsymbol{A}x = \boldsymbol{b}$. *Kde* $\boldsymbol{A}$ *je intervalová matice s rozměry* $n \times n$ *a* $\boldsymbol{b}$ *je intervalový vektor.*

1. *Sestavíme rozšířenou matici soustavy* $[\boldsymbol{A} \,|\, \boldsymbol{b}]$ *a položíme* $k = 1$.
2. *Pokud je* $0 \in \boldsymbol{a}_{ik}$ *pro všechna* $i \geq k$ *skončíme,* $\boldsymbol{A}$ *může být singulární.*
3. *Jinak nalezneme* $0 \notin \boldsymbol{a}_{ik}$, $i \geq k$ *a vyměníme i-tý a k-tý řádek.*
4. *Přenásobíme k-tý řádek* $\frac{1}{\boldsymbol{a}_{kk}}$, *na pozici pivota položíme interval* $[1, 1]$.
5. *Pro každý i-tý řádek* $i > k$ *položme* $\boldsymbol{A}_{i*} = \boldsymbol{A}_{i*} - \boldsymbol{a}_{ik}\boldsymbol{A}_{k*}$, *ve sloupci pod pivotem položme intervaly* $[0, 0]$.

6. Položme $k = k + 1$ a pokud $k \leq n$ jdeme na krok 2.

Algoritmus zpětné substituce bude vypadat stejně, akorát místo čísel počítáme s intervaly. Intervalová verze Gaussovy eliminace však obsahuje jednu nepříjemnost. Díky velkému množství intervalových operací se může stát, že dojde k značnému nadhodnocení obalu množiny řešení. A to tak velkému, že již nebude výsledek použitelný. V nejhorším případě nadhodnocení poroste v závislosti s rozměry matice $\boldsymbol{A}$. Hansen a Smith v roce 1967 ukázali příklad, ve kterém s každou jednotkou rozměru matice soustavy klesá přesnost získaného řešení přibližně o jedno desetinné místo. Toto je jen nejhorší případ, který nutně nemusí nastat vždy. Existují matice (M-matice, H-matice, diagonálně dominantní matice), pro které Gaussova eliminace dává přijatelné výsledky. V ostatních případech je nutné předtím původní soustavu upravit do takového tvaru, aby byla co nejméně náchylná k výše zmíněné chybě.

Jedním ze způsobů, jak snížit roztažení množiny řešení, je podobně jako v neintervalové Gaussově eliminaci použít částečnou pivotizaci. Jako pivotní řádek vybíráme ten, který má největší magnitudu.

5.2 Předpodmínění

Další možností je původní soustavu $\boldsymbol{A}x = \boldsymbol{b}$ převést do takového tvaru, ve kterém dojde k co největšímu omezení vlivů intervalových operací. Česky tento proces budeme nazývat *předpodmínění* (angl. preconditioning). Níže popíšeme metodu předpodmínění, jak ji navrhl Hansen. Nechť A^c je středovou maticí $\boldsymbol{A}$. V praxi nemusí být přímo středovou maticí, stačí aproximace. Ke středové matici spočítáme přibližnou inverzní matici $C \approx (A^c)^{-1}$. Pak spočítáme matici $\boldsymbol{M} = C\boldsymbol{A}$ a vektor $\boldsymbol{m} = C\boldsymbol{b}$. Řešíme nový systém

$$\boldsymbol{M}x = \boldsymbol{m}.$$

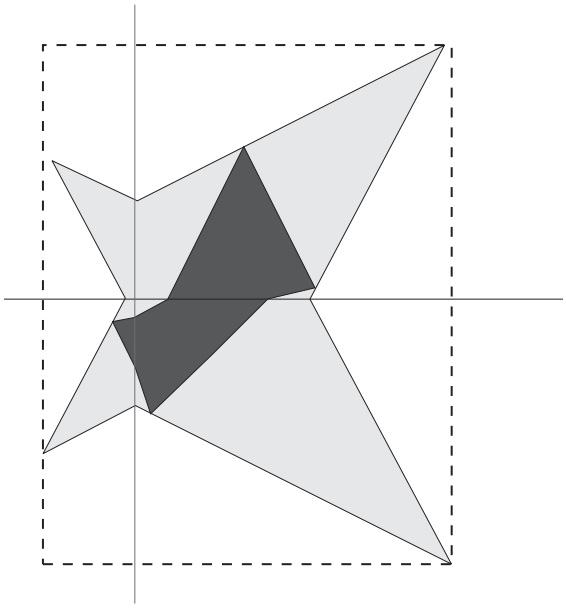
Pokud nejsme schopni spočítat inverzní matici k A^c, protože tato matice je singulární nebo blízká singulární, je pravděpodobné, že systém má neomezenou množinu řešení. Můžeme se ptát, proč soustavu předpodmiňujeme právě maticí C. Jelikož C je matice v ideálním případě inverzní k A^c, dostaneme po přenásobení matice $\boldsymbol{A}$ matici $\boldsymbol{M}$, která má středovou matici velmi blízkou jednotkové matici. Pokud jsou navíc původní intervaly v matici $\boldsymbol{A}$ velmi malé, neliší se celá $\boldsymbol{M}$ příliš od jednotkové matice a intervalové operace s koeficienty mimo diagonálu mají jen malý vliv na nepřesnosti ve výsledné množině řešení.

Počítání matice C zabere nějakou výpočetní dobu navíc. Další malou nevýhodou je, že předpodmínění natahuje množinu řešení. Vezmeme systém z kapitoly o lineárních systémech

$$\begin{pmatrix} [5,\ 10] & [\text{-}20,\ \text{-}5] \\ [10,\ 15] & [5,\ 10] \end{pmatrix} x = \begin{pmatrix} [50,\ 100] \\ [\text{-}50,\ 280] \end{pmatrix}$$

a předpodmíníme ho. Zvětšení množiny řešení nového systému ukazuje obrázek 5.1. Tmavší plocha značí množinu řešení původního systému, světlejší plocha množinu řešení nového systému. Šedé čáry vyznačují jednotlivé osy a obdélník kreslený přerušovanou čárou značí obal nové množiny řešení. Pro velké systémy

je však roztažení množiny řešení zanedbatelné oproti roztažení díky závislostním chybám, ke kterým by došlo bez předpodmínění.



Obrázek 5.1: Zvětšení množiny řešení při předpodmínění

Platí, že množina řešení předpodmíněného systému v sobě vždy zahrnuje řešení původního systému, neboť jsme při předpodmiňování násobili regulární maticí.

Předpodmínění není nutné používat vždy. Pokud je matice $\boldsymbol{A}$ M-matice, platí, že GE přímo produkuje obálku řešení. Naopak v tomto případě by předpodmínění bylo jen na škodu. Obecně Gaussova eliminace funguje dobře na případy, kdy jsou intervaly degenerované nebo velmi úzké. Pokud jsou intervaly velké, může docházet k velkému nárůstu obálky množiny řešení. Podobně jako jsme neintervalovou verzi Gaussovy eliminace převedli na intervalovou verzi, můžeme totéž provést pro Gauss-Jordanovu eliminaci [1]. Pomocí ní budeme lehce schopni spočítat například inverzní intervalovou matici.

5.3 Hansenův přístup pro přeurčené systémy

Předchozí přístup lze víceméně uplatnit s malými úpravami pro přeurčené systémy. Pro přeurčené systémy se často používá metoda nejmenších čtverců. Zde se nám však jedná nikoli o aproximaci množiny řešení, ale přímo o sjednocení množin řešení všech systémů, které intervalový přeurčený systém obsahuje. Přístup pro přeurčené systémy vytvořil profesor Eldon R. Hansen. Je součástí amerického patentu číslo *US 7,296,047 B1*, který sdílí G. Wiliam Walster. Součástí patentu je nejen algoritmus, ale i návrh hardwarové architektury pro výpočet intervalových přeurčených systémů.

Uvedeme zde námi mírně upravenou verzi algoritmu uvedeného v [3]. Mějme přeurčený systém $\boldsymbol{A}x = \boldsymbol{b}$, kde matice $\boldsymbol{A}$ je rozměrů $m \times n$, kde $m > n$. Před vlastní eliminací bude nejprve vhodné, podobně jako v předchozím případě, soustavu vhodně předpodmínit. Začneme tím, že spočítáme matici A^c, která je

[1] Po provedení GJ-eliminace má matice na diagonále samé 1 a mimo diagonálu samé 0.

přibližnou středovou maticí intervalové matice $\boldsymbol{A}$. Z této matice vytvoříme matici B řádu $m \times m$, která vypadá následovně

$$B = \begin{bmatrix} A_1^c & 0 \\ A_2^c & I \end{bmatrix}.$$

Jednotlivé součásti matice B vypadají následovně

A_1^c	horních n řádků matice A^c,
A_2^c	dolních $m-n$ řádků matice A^c,
0	nulová matice řádu $n \times (m-n)$,
I	jednotková matice řádu $(m-n) \times (m-n)$.

Z matice B spočítáme její inverzní matici C. Pokud se nám nepodaří matici spočítat, například proto, že je B singulární, můžeme ho zkusit perturbovat malými hodnotami dokud nebude regulární. Nejedná se nám o přesnou inverzi, stačí aproximace. Dále již postupujeme s předpodmíněnou soustavou

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

Na tuto soustavu aplikujeme algoritmus intervalové Gaussovy eliminace. Tentokrát neeliminujeme až do konce, ale ponecháme poslední sloupec matice $\boldsymbol{A}$ bez eliminace (končíme tedy po $n-1$ krocích). Dostaneme soustavu ve tvaru

$$\begin{bmatrix} \boldsymbol{T} \\ \boldsymbol{W} \end{bmatrix} x = \begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix},$$

kde

$\boldsymbol{T}$	čtvercová horní trojúhelníková matice řádu n,
$\boldsymbol{u}$	intervalový sloupcový vektor s n koeficienty,
$\boldsymbol{v}$	intervalový sloupcový vektor s $(m-n)$ koeficienty,
$\boldsymbol{W}$	je matice rozměrů $(m-n) \times n$ tvaru $[0\ \boldsymbol{z}]$,

kde 0 je nulová matice a $\boldsymbol{z}$ je intervalový sloupcový vektor (poslední sloupec matice $\boldsymbol{W}$). Upravíme ještě $\boldsymbol{z}$ tak, aby obsahoval samé prvky $[1,1]$. Položíme

$$\boldsymbol{v}_i := \frac{\boldsymbol{v}_i}{\boldsymbol{z}_i}$$

pro $i = 1, \ldots, m-n$ a poté položíme za prvky $\boldsymbol{z}$ intervaly $[1,1]$. Díky tomu, že $\boldsymbol{z}_i$ mohou obsahovat prvek 0, můžeme dostat i nekonečný interval $[-\infty, \infty]$. Ukážeme příklad intervalové GE pro přeurčený systém (pro názornost bez předpodmínění). Systém

$$\boldsymbol{A} = \begin{pmatrix} [\ 16.9998,\ 17.0002\ ] & [\ 28.9993,\ 29.0007\ ] & [\ 40.9992,\ 41.0008\ ] \\ [\ 8.9994,\ 9.0006\ ] & [\ 13.9999,\ 14.0001\ ] & [\ 10.9991,\ 11.0009\ ] \\ [\ 15.9991,\ 16.0009\ ] & [\ 25.9999,\ 26.0001\ ] & [\ 3.9993,\ 4.0007\ ] \\ [\ 13.9998,\ 14.0002\ ] & [\ 17.9993,\ 18.0007\ ] & [\ 7.9990,\ 8.0010\ ] \\ [\ 12.9999,\ 13.0001\ ] & [\ 36.9992,\ 37.0008\ ] & [\ 20.9990,\ 21.0010\ ] \end{pmatrix},$$

$$\boldsymbol{b} = \begin{pmatrix} [\ 16.2107,\ 75.7893\ ] \\ [\ 27.9484,\ 60.0516\ ] \\ [\ \text{-}61.0726,\ 135.0726\ ] \\ [\ \text{-}14.6424,\ 102.6424\ ] \\ [\ \text{-}36.5122,\ 80.5122\ ] \end{pmatrix}$$

převedeme intervalovou Gaussovou eliminací pro přeurčený systém do tvaru

$$\begin{bmatrix} \boldsymbol{T} \\ \boldsymbol{W} \end{bmatrix} = \begin{pmatrix} [\ 1.0000,\ 1.0000] & [\ 1.7058,\ 1.7060\ ] & [\ 2.4116,\ 2.4119\ ] \\ [\ 0.0000,\ 0.0000] & [\ 1.0000,\ 1.0000\ ] & [\ \text{-}0.6987,\ \text{-}0.6981\ ] \\ [\ 0.0000,\ 0.0000] & [\ 0.0000,\ 0.0000\ ] & [\ 1.0000,\ 1.0000\ ] \\ [\ 0.0000,\ 0.0000] & [\ 0.0000,\ 0.0000\ ] & [\ 1.0000,\ 1.0000\ ] \\ [\ 0.0000,\ 0.0000] & [\ 0.0000,\ 0.0000\ ] & [\ 1.0000,\ 1.0000\ ] \end{pmatrix},$$

$$\begin{bmatrix} \boldsymbol{u} \\ \boldsymbol{v} \end{bmatrix} = \begin{pmatrix} [\ 0.9535,\ 4.4583\ ] \\ [\ \text{-}6.3738,\ 4.5957\ ] \\ [\ \text{-}3.5444,\ 3.9643\ ] \\ [\ \text{-}3.8950,\ 3.8357\ ] \\ [\ \text{-}4.9540,\ 1.7871\ ] \end{pmatrix}.$$

Nyní již máme celý systém ve vhodném tvaru. Nejprve se budeme zabývat prvky vektoru $\boldsymbol{z}$. Ty, společně s pravým dolním prvkem matice $\boldsymbol{T}$, nám formují, jak bude vypadat řešení proměnné x_n. Dostáváme rovnice

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

Pro zjištění řešení proměnné x_n stačí vyřešit průnik všech těchto intervalů definovaných rovnicemi. Souhrnně zapsáno

$$\boldsymbol{x}_n = \boldsymbol{u}_n \bigcap_{i=1}^{m-n} \boldsymbol{v}_i.$$

V průniku nám vlastně nevadí, že některé pravé strany jsou nekonečné intervaly. Pokud by výsledný průnik byl nekonečný, může to znamenat, že soustava nemá omezené řešení. Může to však i znamenat to, že se nám obálka zvětšila až na nekonečný intervalový vektor díky velkému množství intervalových operací. Pokud intervaly nemají společný průnik, přeurčená soustava nemá řešení. Pro naší soustavu je

$$\boldsymbol{x}_n = [-3.5444, 1.7871].$$

Pokud x_n není prázdný interval, můžeme již klasicky aplikovat zpětnou substituci na soustavu $\boldsymbol{T}x = \boldsymbol{u}$ a najít zbývající komponenty vektoru řešení

$$(\boldsymbol{x}_{n-1}, \boldsymbol{x}_{n-2}, \ldots, \boldsymbol{x}_1).$$

Pro náš zvolený systém je obálka množiny řešení

$$\boldsymbol{x} = \begin{pmatrix} [\ \text{-}13.3267,\ 28.1044\ ] \\ [\ \text{-}8.8501,\ 5.8442\ ] \\ [\ \text{-}3.5444,\ 1.7871\ ] \end{pmatrix}.$$

Nyní již můžeme předvést celý algoritmus výpočtu intervalové obálky čtvercového nebo přeurčeného systému pomocí intervalové Gaussovy eliminace.

Algoritmus (Obálka pomocí GE). *Mějme intervalový lineární systém*

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

kde matice $\boldsymbol{A}$ *má rozměry* $m \times n$ *pro* $m \geq n$.

1. *Provedeme intervalovou Gaussovu eliminaci až do* $k = n - 1$.
2. *Za prvky matice* $\boldsymbol{A}_{n,i}$ *položíme intervaly* $[1, 1]$ *a za* $\boldsymbol{b}_i$ *dosadíme* $\frac{\boldsymbol{b}_i}{\boldsymbol{A}_{n,i}}$ *pro každé* $(i = n, \ldots, m)$.
3. $\boldsymbol{x}_n$ *je průnikem všech* $\boldsymbol{b}_i$ *pro* $(i = n, \ldots, m)$, *pokud je průnik prázdný, končíme – systém nemá řešení.*
4. *Dopočítáme hodnoty* $\boldsymbol{x}_{n-1}, \ldots, \boldsymbol{x}_1$ *intervalovou zpětnou substitucí.*
5. $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ *je výsledná obálka.*

6. Rohnova metoda

Zcela jiný přístup pro počítání vnější obálky nabízí profesor Jiří Rohn. Publikoval ho v roce 1995 v [10]. Základní teorie je postavena na následujícím tvrzení, které kvůli úplnosti uvádíme i s důkazem.

6.1 Základní tvrzení

Tvrzení (Rohn). *Mějme přeurčený intervalový lineární systém $\boldsymbol{A}x = \boldsymbol{b}$ s množinou řešení S. Nechť R je libovolná reálná matice řádu $n \times m$ a nechť x_0 a $d > 0$ jsou libovolné n-prvkové reálné vektory takové, že platí*

$$Gd + g < d,$$

kde

$$G = |I - RA^c| + |R|A^\Delta$$

a

$$g = |R(A^c x_0 - b^c)| + |R|(A^\Delta |x_0| + b^\Delta).$$

Pak platí

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

Důkaz. Nechť $x \in S$, pak platí $Ax = b$ pro nějaké $A \in \boldsymbol{A}$ a $b \in \boldsymbol{b}$. Pak ho můžeme rozepsat jako

$$x = x + R(-Ax + b) = (I - RA)x + Rb.$$

Z toho dostáváme

$$\begin{aligned} x - x_0 &= (I - RA)x + Rb - Ix_0 \\ &= (I - RA)x + Rb - Ix_0 + RAx_0 - RAx_0 \\ &= (I - RA)(x - x_0) + R(b - Ax_0) \\ &= (I - RA^c)(x - x_0) + R(A^c - A)(x - x_0) + R(b^c - A^c x_0) \\ &\quad + R(A^c - A)x_0 + R(b - b^c). \end{aligned}$$

Nyní celou rovnici převedeme do absolutních hodnot

$$\begin{aligned} |x - x_0| &\leq |I - RA^c||x - x_0| + |R|A^\Delta |x - x_0| \\ &\quad + |R(b^c - A^c x_0)| + |R|A^\Delta |x_0| + |R|b^\Delta \\ &= G|x - x_0| + g. \end{aligned}$$

Z této nerovnice vyplývá

$$\begin{aligned} |x - x_0| &\leq G|x - x_0| + g \\ (I - G)|x - x_0| &\leq g. \end{aligned} \tag{6.1}$$

Pro d splňující počáteční podmínku máme

$$\begin{aligned} Gd + g &< d \\ g &< (I - G)d. \end{aligned} \tag{6.2}$$

Z nerovností 6.1 a 6.2 plyne

$$(I - G)|x - x_0| < (I - G)d$$

Jelikož je $g \geq 0$, platí $Gd < d$. Navíc matice G je nezáporná a $d > 0$, platí tedy $\varrho(G) < 1$. Pozorování, ze kterého toto vyplývá lze nalézt i s důkazem v [7] (Důsledek 3.2.3). Díky tomu například z věty 168 viz [11] plyne existence inverzní matice k $(I - G)$ a navíc i $(I - G)^{-1} \geq 0$. Tedy můžeme touto maticí přenásobit obě strany nerovnice aniž by se porušila nerovnost. Získáme

$$|x - x_0| < d.$$

Jinak zapsáno $x \in [x_0 - d, x_0 + d]$. Jelikož bylo x libovolné z S, tak

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

□

Toto tvrzení platí pouze pro případ $m \geq n$, protože pro $m < n$ nenalezneme vhodné d. Pro důkaz možno nahlédnout do [10].

Tvrzení nám poskytuje jednoduchý předpis, jak spočítat intervalovou obálku množiny řešení. Jediné, co ještě neznáme jsou vektory d, x_0 a matice R. Existuje několik možností, jak d získat.

6.2 Nalezení vhodného d

Rohn ve svém článku navrhuje iterativní algoritmus pro získání d. Nerovnici

$$Gd + g < d$$

můžeme totiž přepsat jako rovnici

$$Gd + g + \epsilon = d.$$

Začneme s libovolným malým positivním vektorem ϵ a postupně iterujeme následující algoritmus, dokud d nesplňuje podmínku z tvrzení. Algoritmus vypadá následovně

Algoritmus (Nalezení d). *Mějme G, g z výše zmíněného tvrzení*

1. *Za ϵ položme malý positivní vektor, $d^* = 0$.*
2. *$d = d^*$.*
3. *$d^* = Gd + g + \epsilon$.*
4. *Opakujeme kroky 2. a 3. dokud není $|d^* - d| < \epsilon$.*
5. *d je hledaný vektor.*

Není dopředu jisté, jak dlouho tento algoritmus poběží. Můžeme však s jistotou říci, že daný algoritmus svůj běh vždy skončí. Jak ostatně ukazuje následující tvrzení.

Tvrzení. *Následující dvě podmínky jsou ekvivalentní.*

- $\varrho(G) < 1$.
- *Algoritmus skončí po konečně mnoha krocích pro každé $\epsilon > 0$.*

Důkaz. ($\Rightarrow$) Podívejme se jak vypadají d v jednotlivých krocích.

$$\begin{aligned} d_0 &= 0, \\ d_1 &= g + \epsilon, \\ d_2 &= Gd_1 + g + \epsilon = G(g + \epsilon) + g + \epsilon, \\ &\vdots \\ d_j &= G^{j-1}(g + \epsilon) + G^{j-2}(g + \epsilon) + \ldots + G(g + \epsilon) + g + \epsilon, \\ d_{j+1} &= G^j(g + \epsilon) + G^{j-1}(g + \epsilon) + \ldots + G(g + \epsilon) + g + \epsilon. \end{aligned}$$

Absolutní hodnota rozdílu dvou po sobě jdoucích d vypadá takto

$$|d_{j+1} - d_j| = |G^j(g + \epsilon)| = G^j(g + \epsilon).$$

Jelikož je $\varrho(G) < 1$ platí $G^j \to 0$ a tedy i $G^j(g + \epsilon) \to 0$, kde 0 je nulový vektor. Z definice limity existuje pro každý vektor $\epsilon > 0$ přirozené číslo n_0 takové, že pro každé $n \geq n_0, n \in \mathbb{N}$ platí

$$|d_{j+1} - d_j| < \epsilon.$$

Což je zastavující podmínka algoritmu.

($\Leftarrow$) Nechť algoritmus skončí po konečně mnoha krocích pro libovolný vektor $\epsilon > 0$. Pak skončí i pro nějaký konkrétní $\epsilon > 0$. Pro tento ϵ je splněna ukončující podmínka $|d^* - d| < \epsilon$. Z toho dostáváme

$$d^* = Gd + g + \epsilon < d + \epsilon$$

proto

$$Gd \leq Gd + g < d.$$

A jelikož $d > 0$ a G je nezáporná, platí (stejně jako v důkazu úvodního tvrzení)

$$\varrho(G) < 1.$$

□

Z výše uvedeného důkazu je vidět, že konečnost algoritmu nezávisí na volbě ϵ. Co na volbě ϵ závisí, je doba výpočtu d. Jelikož doba získání d může být hodně dlouhá, je vhodné počet kroků algoritmu shora omezit konstantou. Pokud do daného počtu kroků není vyhovující d nalezeno, přerušíme další výpočet.

Jako další dosud nepublikovanou možnost navrhujeme d spočítat přímo z předpisu tvrzení (Rohn) za použití malého kladného vektoru ϵ.

$$\begin{aligned} d &> Gd + g \\ d &= Gd + g + \epsilon \\ (I - G)d &= g + \epsilon. \end{aligned}$$

Vlastně vyřešíme poslední rovnici. Můžeme použít nějakou metodu pro řešení lineárních systémů, například `verifylss` z Intlabu. Nebo využít pro řešení funkci Matlabu `linsolve`. Nevadí, že může nastat případ, kdy díky zaokrouhlovacím chybám dostaneme mírně nepřesné řešení. Po získání d můžeme dosazením zkontrolovat, zda opravdu platí nerovnost $Gd + g < d$.

6.3 Nalezení R a x_0

Nyní již dokážeme získat d. K výpočtu obálky množiny řešení systému rovnic je však potřebný ještě vektor x_0 a matice R. Rohn ve svých článcích doporučuje brát

$$x_0 \approx Rb_c.$$

Matici R doporučuje počítat jako aproximaci Moore-Penroseovy inverze A_c.

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

Moore-Penroseovu inverzi můžeme takto spočítat, neboť jestliže intervalová matice $\boldsymbol{A}$ splňuje podmínku $Gd+g < d$, pak pro každou $A \in \boldsymbol{A}$ platí, že má lineárně nezávislé sloupce. Důkaz je též možno nalézt v [10].

6.4 Vylepšení obálky

Pokud bychom chtěli obálku ještě více zlepšit, můžeme ji iterativně zpřesňovat. Lze ji vylepšit jinou volbou R a x_0, jelikož ve tvrzení R i x_0 vybíráme libovolné. Rohn popisuje v [10] následující algoritmus.

Algoritmus (Vylepšení uzávěru). *Mějme dánu intervalovou lineární soustavu* $\boldsymbol{A}x = \boldsymbol{b}$.

1. *Opakujme od* $j = 1$ *do* $j =$ *maximální počet kroků.*
2. *Náhodně vybereme* $A \in \boldsymbol{A}, b \in \boldsymbol{b}$.
3. $R = (A^T A)^{-1} A^T$.
4. $x_0 = Rb$.

5. *Spočítejme* $d > 0$ *libovolnou metodou.*

6. *Pro* $j = 1$ *položme* $\boldsymbol{x} = [x_0 - d, x_0 + d]$ *jinak* $\boldsymbol{x} = \boldsymbol{x} \cap [x_0 - d, x_0 + d]$.

7. $S \subseteq \boldsymbol{x}$.

Častou podmínkou zastavení iteračních algoritmů je maximální počet kroků (jako v tomto případě). Druhou častou zastavující podmínkou je míra odlišnosti nového a starého řešení. Pokud jsou obě řešení jen málo vzdálená, respektive zlepšení v posledním kroku bylo menší než je určitá mez, končíme. Tento přístup v tomto algoritmu bohužel nelze užít. Jako protipříklad uvažme tři řešení $(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$ v po sobě jdoucích iteračních krocích a zastavující kritérium ϵ. Nechť platí například, že horní i dolní meze $\boldsymbol{x}_1, \boldsymbol{x}_2$, se liší nejvýš o ϵ. Tedy je splněna zastavující podmínka. Oba odhady $\boldsymbol{x}_1, \boldsymbol{x}_2$ mohou být velmi špatné a $\boldsymbol{x}_3$, které by přišlo po nich, by odhad řešení mnohonásobně zlepšilo. Pak ale algoritmus končí po druhém kroku a na mnohem lepší řešení v kroce třetím se již nedostane. Pokud volíme jednotlivé matice A a b náhodně, je nutné definovat zastavující podmínku pouze pomocí předem daného počtu kroků.

7. Lineární programování

Jiným způsobem, jak k problematice hledání obálky množiny řešení intervalového lineárního systému přistupovat, je lineární programování. Vyjdeme z následující věty, která je mnohými považována za základní charakteristiku slabé řešitelnosti intervalových linárních systémů. Díky její významnosti větu uvádíme i s důkazem. Větu dokázali Oettli a Prager v roce 1964.

7.1 Věta Oettli-Prager

Věta (Oettli-Prager). *Mějme intervalový lineární systém* $\boldsymbol{A}x = \boldsymbol{b}$*, kde*

$$\begin{aligned}\boldsymbol{A} &= [A^c - A^\Delta, A^c + A^\Delta] \in \mathbb{IR}^{m\times n},\\ \boldsymbol{b} &= [b^c - b^\Delta, b^c + b^\Delta] \in \mathbb{IR}^{m}.\end{aligned}$$

Vektor $x \in \mathbb{R}^n$ *je slabým řešením toho systému, právě tehdy, když*

$$|A^c x - b^c| \leq A^\Delta |x| + b^\Delta.$$

Důkaz. ($\Rightarrow$) Pokud je vektor x slabým řešením $\boldsymbol{A}x = \boldsymbol{b}$, platí $Ax = b$ pro nějaké $A \in \boldsymbol{A}$ a $b \in \boldsymbol{b}$. Z toho vyplývá

$$\begin{aligned}|A^c x - b^c| &= |(A^c x - b^c) + (b - Ax)|\\ &= |(A^c - A)x + b - b^c|\\ &= |(A^c - A)x| + |b - b^c| \leq A^\Delta |x| + b^\Delta.\end{aligned}$$

($\Leftarrow$) Nechť platí $|A^c x - b^c| \leq A^\Delta |x| + b^\Delta$ pro nějaké x. Položme $y \in \mathbb{R}^m$

$$y_i = \begin{cases} \frac{(A^c x - b^c)_i}{(A^\Delta |x| + b^\Delta)_i}, & \text{jestliže } (A^\Delta |x| + b^\Delta)_i > 0\\ 1, & \text{jestliže } (A^\Delta |x| + b^\Delta)_i = 0 \end{cases} \quad (i = 1, \ldots, m).$$

Pak platí, že $|y| \leq e$ a zároveň

$$A^c x - b^c = D_y (A^\Delta |x| + b^\Delta).$$

Pokud položíme $z = \operatorname{sgn}(x)$, pak evidentně $|x| = D_z x$. Z předchozího vzorce odstraněním absolutní hodnoty tímto způsobem a převedením prvků obsahujících x na stejné strany dostáváme

$$(A^c - D_y A^\Delta D_z)x = b^c + D_y b^\Delta.$$

Jelikož $|y| \leq e$ a $z \in \{\pm 1\}^n$, platí

$$\begin{aligned}|D_y A^\Delta D_z| &\leq A^\Delta,\\ |D_y b^\Delta| &\leq b^\Delta.\end{aligned}$$

Z toho vyplývá, že $A^c - D_y A^\Delta D_z \in \boldsymbol{A}$ a $b_c + D_y b^\Delta \in \boldsymbol{b}$. Tedy x je slabé řešení $\boldsymbol{A}x = \boldsymbol{b}$. □

Množinu řešení systému $\boldsymbol{A}x = \boldsymbol{b}$ můžeme tedy popsat

$$S = \{x;\ |A^c x - b^c| \leq A^\Delta |x| + b^\Delta\}.$$

7.2 Prohledávání ortantů

Důležitým pojmem, se kterým budeme pracovat je ortant.

Definice (Ortant). *Ortant se signaturou $z \in \{\pm 1\}^n$ je množina $Z \subseteq \mathbb{R}^n$ taková, že*

$$Z = \{D_z x \geq 0;\ x \in \mathbb{R}^n\}.$$

Jinak řečeno, všechny vektory spadající do daného ortantu musí mít znaménka jednotlivých složek stejná jako odpovídající složky v signatuře ortantu.
Množina řešení lineárního intervalového systému nemusí být nutně konvexní, ale je konvexní v každém ortantu (i když může být prázdná). Můžeme tedy s pomocí výše uvedené věty a úlohy lineárního programování spočítat obálku řešení v každém ortantu zvlášť. Výhodou je, že dostaneme přesný obal množiny řešení našeho systému. Kapitolu o lineárním programování jsme zařadili jednak z důvodu ucelenosti, ale také kvůli tomu, že při porovnávání výsledných obálek jednotlivých metod se nám bude hodit znalost obalu. V n-dimenzionálním prostoru máme celkem 2^n ortantů. Řešit úlohu lineárního programování v každém z nich může být náročným výpočetním problémem již pro relativně nízká čísla (20-30). Pokud chceme zjišťovat obal prozkoumáním řešení v jednotlivých ortantech, není nutné prozkoumat všechny. Mějme nějaký intervalový vektor jako odhad obálky množiny řešení. K určení, které ortanty prohledávat nám dopomůže znaménkový vektor tohoto odhadu. Pro intervalový vektor $\boldsymbol{x}$ definujeme jeho znaménkový vektor $\operatorname{sgn}(\boldsymbol{x})$ lehce odlišně než pro neintervalový vektor.

Definice (sgn pro intervalový vektor). *Pro intervalový vektor*

$$\boldsymbol{x} = ([\underline{x}_1, \overline{x}_1], [\underline{x}_2, \overline{x}_2], \ldots, [\underline{x}_n, \overline{x}_n])$$

definujeme jeho znaménkový vektor $\operatorname{sgn}(\boldsymbol{x})$ *následovně*

$$(\operatorname{sgn}(\boldsymbol{x}))_i = \begin{cases} 1, & \textit{jestliže } \underline{x}_i > 0, \\ -1, & \textit{jestliže } \overline{x}_i < 0, \\ 0, & \textit{jestliže } 0 \in x_i. \end{cases}$$

Pokud již známe nějakou obálku množiny řešení, její znaménkový vektor nám udává, které části osy musíme procházet. Pokud je na některé pozici 0, musíme prohledat obě části příslušné osy. Nejprve nadefinujme množinu ortantů, které budeme muset prohledat.

Definice. *Mějme předběžnou obálku řešení $\boldsymbol{x}$. Množinu signatur ortantů, které budeme muset prohledat označme*

$$Orth(\boldsymbol{x}) = \{z;\ D_z \operatorname{sgn}(\boldsymbol{x}) \geq 0, z \in \{\pm 1\}^n\}.$$

7.3 Úloha pro lineární programování

Nyní se dostáváme k jádru naší kapitoly, které shrneme v podobě následujícího pozorování. V trochu jiném zápisu ho nalezneme v [10].

Pozorování. *Mějme intervalový lineární systém* $\boldsymbol{A}x = \boldsymbol{b}$*, kde*

$$\begin{aligned}\boldsymbol{A} &= [A^c - A^\Delta, A^c + A^\Delta] \in \mathbb{IR}^{m\times n},\\ \boldsymbol{b} &= [b^c - b^\Delta, b^c + b^\Delta] \in \mathbb{IR}^{m}.\end{aligned}$$

Pak přesný obal množiny řešení v ortantu daném signaturou z *získáme vyřešením následujících úloh lineárního programování*

$$\begin{aligned}\underline{x}_i^z &= \min x_i,\\ \overline{x}_i^z &= \max x_i,\end{aligned}$$

pro $(i = 1, 2, \ldots, n)$ *na stále stejné množině popsané soustavou*

$$\begin{aligned}&D_z x \geq 0,\\ &b^c - b^\Delta \leq (A^c + A^\Delta D_z)x,\\ &(A^c - A^\Delta D_z)x \leq b^c + b^\Delta\}.\end{aligned}$$

Důkaz. Množina řešení v každém ortantu tvoří konvexní polyedr, můžeme tedy použít úlohu lineárního programování. Účelové funkce pro horní a dolní meze v daném směru jsou $\min e_i^T x$ a $\max e_i^T x$ tedy vlastně $\min x_i$ a $\max x_i$. $D_z x$ nám určuje ortant, který prohledáváme, neboť pro každé x z daného ortantu platí $D_z x \geq 0$. Podle věty Oettli-Prager platí

$$|A^c x - b^c| \leq A^\Delta |x| + b^\Delta.$$

Jelikož se nacházíme v ortantu z, tak pro každé řešení v něm platí

$$|x| = D_z x.$$

Můžeme tedy pro daný ortant přepsat větu Oettli-Prager na

$$|A^c x - b^c| \leq A^\Delta D_z x + b^\Delta.$$

Rozepsáním absolutní hodnoty dostáváme dvě soustavy, které musí řešení x splňovat

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

Převedením prvků obsahujících x na stejnou stranu získáme rovnice ze znění pozorování. Abychom získali obal v daném ortantu, musíme úlohu vyřešit ve všech směrech x_i.

□

Mějme tedy intervalový odhad $\boldsymbol{x}$ množiny řešení. Dále mějme spočítané optimální obaly v každém ortantu podle $\operatorname{sgn}(\boldsymbol{x})$. Snadno z nich již určíme obal celé množiny řešení $\boldsymbol{h} = [\underline{h}, \overline{h}]$ podle předpisu

$$\begin{aligned}\underline{h}_i &= \min\{\underline{x}_i^z \mid z \in Orth(\boldsymbol{x})\},\\ \overline{h}_i &= \max\{\overline{x}_i^z \mid z \in Orth(\boldsymbol{x})\}, \quad \text{pro } (i = 1, 2, \ldots, n).\end{aligned}$$

Někdy se může stát, že množství prohledávaných ortantů bude příliš velké. Zda je vhodné použít tento postup, záleží na mohutnosti množiny $Orth(\boldsymbol{x})$. Pro každý ortant totiž potřebujeme vyřešit $2n$ úloh lineárního programování pro určení optimálního obalu. Pokud bychom prohledávali všechny ortanty řešíme jich

$$2^n \times (2n).$$

V praxi nebývá lineární programování vždy používáno, neboť jeho verifikovaná verze je značně časově náročná. My zde nebudeme vytvářet vlastní nástroj pro řešení úlohy lineárního programování. Použijeme existující spolehlivé nástroje například Rohnův Versoft – konkrétně funkci `verlinprog`.

7.4 Generování signatur ortantů

Při prohledávání ortantů potřebujeme šikovně generovat jednotlivé signatury ortantů. V ideálním případě tak, že v každém kroku algoritmu vygenerujeme novou signaturu. Tedy jsme schopni všechny signatury vygenerovat v 2^n krocích. K tomu nám poslouží následující algoritmus. Důkaz správnosti je možné dohledat v [1].

Algoritmus (Generování $\{\pm 1\}^n$). *Mějme dáno $n \in \mathbb{N}$.*

1. $z = 0 \in \mathbb{R}^n$, $y = e \in \mathbb{R}^n$, $Y = \{y\}$.
2. *Opakujme kroky 3. až 6. dokud $z \neq e$.*
3. $k = \min\{i\,;\ z_i = 0\}$.
4. *Položme $z_i = 0$ pro všechna $(i = 1, \ldots, k-1)$.*
5. $z_k = 1$, $y_k = -y_k$.
6. $Y = Y \cup \{y\}$.
7. *Y je množina $\{\pm 1\}^n$.*

Algoritmus využijeme, i když negenerujeme všechny signatury ortantů. Jestliže máme znaménkový vektor $\operatorname{sgn}(\boldsymbol{x})$ obálky množiny řešení, vygenerujeme nejprve množinu $\{\pm 1\}^k$, kde k je počet nul ve znaménkovém vektoru. Jsou v ní obsaženy všechny možnosti, jak dosadit prvky $-1, 1$ na pozice, na kterých je 0 v $\operatorname{sgn}(\boldsymbol{x})$. Poté jednotlivé prvky z $\{\pm 1\}^k$ doplníme na příslušné pozice fixními hodnotami $-1, 1$ ze znaménkového vektoru. Výsledkem je množina ortantů, které budeme prohledávat podle $\operatorname{sgn}(\boldsymbol{x})$.

8. Iterační metody

Iterační metody jsou často velmi efektivní a v praxi používané. Pro čtvercové systémy existuje celá řada iteračních metod. Některé z nich však v obdélníkovém případě nelze užít (například pokud používáme rozklady čtvercových matic). Zde si představíme iterační metody, které, poté co je mírně upravíme, lze přenést i na přeurčené systémy. Použijeme Jacobiho metodu a Gauss–Seidelovu metodu. I když obě využívají podobnou ideu, každá ma své výhody, a proto si je představíme zvlášť v každé podkapitole. Iterační metody pro čtvercové systémy lze užít pro řešení přeurčených systémů, pokud tyto systémy převedeme na čtvercové. Převodům se budeme věnovat v kapitole následující. U iteračních metod na začátku předpokládáme, že máme již nějaký odhad obálky, ve které se řešení soustavy nachází. Počáteční odhad budeme značit

$$\boldsymbol{x}^{(0)} = (\boldsymbol{x}_1^{(0)}, \boldsymbol{x}_2^{(0)}, \ldots, \boldsymbol{x}_n^{(0)}).$$

8.1 Jacobiho metoda

Mějme intervalovou lineární soustavu $\boldsymbol{A}x = \boldsymbol{b}$, kde intervalová matice $\boldsymbol{A}$ má rozměry $m \times n$. Nejprve metodu vysvětlíme na intervalovém čtvercovém případě. Rovnice systému, kterou určuje i-tý řádek vypadá takto

$$\boldsymbol{A}_{i1}x_1 + \boldsymbol{A}_{i2}x_2 + \ldots + \boldsymbol{A}_{ii}x_i + \ldots + \boldsymbol{A}_{in}x_n = \boldsymbol{b}_i.$$

Proměnou x_i z této rovnice vyjádříme jako

$$x_i = \frac{1}{\boldsymbol{A}_{ii}}\Big[\boldsymbol{b}_i - (\boldsymbol{A}_{i1}x_1 + \ldots + \boldsymbol{A}_{i(i-1)}x_{i-1} + \boldsymbol{A}_{i(i+1)}x_{i+1} + \ldots + \boldsymbol{A}_{in}x_n)\Big].$$

To nám již dává formuli, pro iterativní zlepšování jedné proměnné.

$$\boldsymbol{x}_i^* = \frac{1}{\boldsymbol{A}_{ii}}(\boldsymbol{b}_i - \sum_{j \neq i} \boldsymbol{A}_{ij}\boldsymbol{x}_j^{(k)}) \quad \text{pro } i = (1, \ldots, n). \tag{8.1}$$

Pro vypočítání $\boldsymbol{x}_i^{(k+1)}$ stačí provést průnik se starým řešením.

$$\boldsymbol{x}_i^{(k+1)} = \boldsymbol{x}_i^{(k)} \cap \boldsymbol{x}_i^*.$$

Pokud dostaneme prázdný průnik, znamená to, že systém nemá řešení. Pokud bychom odhad množiny řešení zaručenými metodami nepočítali, ale zvolili ho řekněme náhodně, může prázdný průnik také znamenat, že jsme počáteční odhad určili špatně.

Z předpisu vyplývá, že výhoda Jacobiho metody spočívá v tom, že jednotlivé $\boldsymbol{x}_i^{(k+1)}$ lze počítat na základě starých hodnot paralelně. Dále z předpisu vyplývá jeden malý problém nacházející se ve zlomku $\frac{1}{\mathbf{A}_{ii}}$. Může totiž nastat případ $0 \in \boldsymbol{A}_{ii}$. To lze vyřešit několika způsoby. Buď budeme mít požadavek, aby koeficienty na diagonále matice neobsahovaly 0, nebo se můžeme snažit přerovnat řádky tak, aby na diagonále byly intervaly neobsahující 0. Další možností je užít

rozšířenou intervalovou aritmetiku (o té jsme se zmínili již v kapitole o intervalové aritmetice). Pak není nutné mít žádné požadavky na koeficienty matice.

Až dosud jsme popisovali intervalovou verzi Jacobiho metody pro čtvercové systémy. U přeurčených systémů máme m rovnic a n proměnných a platí $m > n$. Pro čtvercové systémy jsme si představili předpis, ve kterém i-tá proměnná byla generována z i-tého řádku. Tak tomu bylo kvůli výhodnému předpokladu funkčnosti algoritmu – stačí, aby diagonála obsahovala prvky bez 0. Tak tomu ale obecně být nemusí, proměnné můžeme k řádkům přiřadit různě. Přiřazení můžeme dokonce volit náhodně, či je různě cyklicky permutovat. Podobně se můžeme zachovat u přeurčených systémů. Z m rovnic jich můžeme v daném kroku vybrat n a ty použít pro zlepšení příslušných složek řešení. Jelikož řádky volíme náhodně, nestačí nám požadovat, aby prvky na diagonále neobsahovaly 0. Museli bychom požadovat $0 \notin \boldsymbol{A}_{ij}$ pro všechna i, j. Což je dosti omezující podmínka. My to budeme řešit tak, že žádné podmínky na koeficienty uvažovat nebudeme. Pokud koeficient $\boldsymbol{A}_{ij}$ bude obsahovat 0, jednoduše pomocí i-tého řádku nebudeme vyjadřovat proměnnou x_j a vyjádříme ji pomocí jiného. Pokud budou 0 rozmístěny v koeficientech matice nešikovně, může se stát, že o řešení soustavy nebudeme touto metodou schopni nic říct. Pro názornost zde uvádíme celý algoritmus.

Algoritmus (Jacobiho metoda). *Mějme intervalový lineární systém $\boldsymbol{A}x = \boldsymbol{b}$. Matice $\boldsymbol{A}$ je řádu $m \times n$ taková, že $m \geq n$.*

1. *Spočítejme odhad množiny řešení $\boldsymbol{x}^{(0)}$.*
2. *Opakujme kroky 3. až 5. dokud není konec.*
3. *Vyberme n z m rovnic a přiřaďme je k navzájem různým proměnným.*
4. *Spočítejme $\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_n^*$ podle vzorce 8.1.*
5. *Položme $\boldsymbol{x}_i^{(k+1)} = \boldsymbol{x}_i^* \cap \boldsymbol{x}_i^{(k)}$ pro $(i = 1, \ldots, n)$. Pokud je některý z průniků prázdný, končíme – systém nemá řešení.*
6. *$\boldsymbol{x}^{(k+1)}$ obsahuje množinu řešení.*

Ještě je potřeba zamyslet se u kroku 2. nad částí „dokud není konec“. U čtvercového systému se pro Jacobiho metodu používá ukončující podmínka

$$d < \epsilon,$$

kde d je vzdálenost mezi $\boldsymbol{x}^{(k+1)}$ a $\boldsymbol{x}^{(k)}$ a ϵ je malý kladný vektor. Tím, že vybíráme vždy n z m řádků náhodně, může se nám stát, že dvě po sobě jdoucí obálky řešení se zlepší jen o málo (nebo dokonce vůbec nezlepší), a tudíž se již nedostane na mnohem výraznější vylepšení pomocí výběru jiných n řádků. Spolehlivou zastavující podmínkou je pevně daný počet iteračních kroků. Problém je však, jak počet kroků dopředu odhadnout. Na druhou stranu iterační metody neběží příliš mnoho kroků (i pro velké systémy jsou to řádově desítky).

8.2 Gauss-Seidlova metoda

Intervalovou verzi Gauss-Seidelovy metody poprvé zmínili Alefeld a Herzberger v roce 1970. Gauss-Seidelova metoda je určitým vylepšení Jacobiho metody, kdy s nově spočítanými koeficienty vektoru obálky počítáme rovnou a nečekáme až na další iterační krok. Nevýhodou Gauss-Seidelovy metody ale je, že ji nelze paralelizovat. Iterační předpis je podobný jako v Jacobiho metodě. Hodnotu $\boldsymbol{x}_i$ z rovnice v i-tém řádku v kroku $(k+1)$ vyjádříme

$$\begin{aligned} \boldsymbol{x}_i^* &= \frac{1}{\boldsymbol{A}_{ii}}\Big[\boldsymbol{b}_i - (\boldsymbol{A}_{i1}\boldsymbol{x}_1^{(k+1)} + \ldots + \boldsymbol{A}_{i(i-1)}\boldsymbol{x}_{i-1}^{(k+1)} + \\ &\quad + \boldsymbol{A}_{i(i+1)}\boldsymbol{x}_{i+1}^{(k)} + \ldots + \boldsymbol{A}_{in}\boldsymbol{x}_n^{(k)})\Big]. \end{aligned} \tag{8.2}$$

Tedy slovně řečeno, když vyjadřujeme proměnnou $\boldsymbol{x}_i^{(k+1)}$, využijeme k tomu již nově spočítaných hodnot $\boldsymbol{x}_1^{(k+1)}, \boldsymbol{x}_2^{(k+1)}, \ldots, \boldsymbol{x}_{i-1}^{(k+1)}$ a zbylých starých hodnot proměných $\boldsymbol{x}_{i+1}^{(k)}, \boldsymbol{x}_{i+2}^{(k)}, \ldots, \boldsymbol{x}_n^{(k)}$. Pro vypočítání $\boldsymbol{x}_i^{(k+1)}$ stačí opět provést průnik se starým řešením.

$$\boldsymbol{x}_i^{(k+1)} = \boldsymbol{x}_i^{(k)} \cap \boldsymbol{x}_i^*.$$

Algoritmus (Gauss-Seidelova metoda). *Mějme systém $\boldsymbol{A}x = \boldsymbol{b}$, kde $\boldsymbol{A}$ je řádu $m \times n$ taková, že $m \geq n$.*

1. *Spočítejme odhad množiny řešení $\boldsymbol{x}^{(0)}$.*
2. *Opakujme kroky 3. až 5. dokud není konec.*
3. *Vyberme n z m rovnic a přiřaďme je k jednotlivým proměnným.*
4. *Spočítejme $\boldsymbol{x}_i^*$ podle vzorce 8.2 a $\boldsymbol{x}_i^{(k+1)} = \boldsymbol{x}_i^* \cap \boldsymbol{x}_i^{(k)}$ pro $(i = 1, \ldots, n)$. Pokud je některý z průniků prázdný, končíme – systém nemá řešení.*
5. *$\boldsymbol{x}^{(k+1)}$ obsahuje množinu řešení.*

U Gauss-Seidelovy metody můžeme v 3. kroku řádky a proměnné svázat různými způsoby. Buď můžeme použít stejný způsob jako u Jacobiho metody. Tedy, že i-té proměnné přiřadíme řádek, který jsme jiné proměnné ještě nepřiřadili a který nemá na i-té pozici koeficient obsahující 0. Tím, že nové hodnoty proměnných počítáme rovnou, můžeme přiřadit i zbylé řádky redundantně k nějakým proměnným. Toto zobrazení buď můžeme ponechat fixní přes všechny kroky, nebo ho můžeme v každém iteračním kroku obměňovat. Pokud máme v matici hodně koeficientů obsahujících 0, můžeme podle těch neobsahujících 0 vyjadřovat každou proměnnou v každém řádku.

Pokud použijeme výběr proměnných a řádků jako v Jacobiho metodě, platí pro ukončující podmínky Gauss-Seidelovy metody stejné důsledky. Pokud vyjadřujeme všechny proměnné podle všech řádků neobsahujících na příslušné pozici 0, nebo je každému řádku pevně přiřazena nějaká proměnná, můžeme použít zastavující podmínku s ϵ.

Může se stát, že obě metody uvedené v této kapitole nebudou konvergovat. Pak je nutné použít jiný způsob řešení. Pro čtvercové systémy jsou iterační metody mnohdy efektivní. Gauss-Seidelova metoda konverguje v mnoha případech

rychleji než Jacobiho metoda. U přeurčených systémů však nastává potíž, pokud nepoužijeme předpodmínění, často se nám stane, že předběžná obálka již nepůjde vylepšit, neboť při vyjadřování proměnných díky intervalovým operacím dostaneme mnohem větší obálku než byla předběžná. Tím pádem ta zůstane až do konce naším nejlepším odhadem. Metody lze ale vylepšit s pomocí předpodmínění.

8.3 Předpodmínění

Ve čtvercovém případě se soustava často před vlastním výpočtem předpodmíní. Využívá se k tomu přibližná inverzní matice k středové matici $\boldsymbol{A}$ ze stejných důvodů, jako jsme uvedli v kapitole o Gaussově eliminaci. Přeurčené systémy budeme předpodmiňovat maticí C stejnou jako pro přeurčené systémy v kapitole o Gaussově eliminaci. Po předpodmínění dostaneme často soustavu, jejíž matice na levé straně bude vypadat podobně jako následující matice

$$\begin{pmatrix} \sim 1 & \sim 0 & \dots & \sim 0 \\ \sim 0 & \sim 1 & \dots & \sim 0 \\ \vdots & \vdots & \ddots & \vdots \\ \sim 0 & \sim 0 & \dots & \sim 1 \\ \sim 0 & \sim 0 & \dots & \sim 0 \\ \vdots & \vdots & \ddots & \vdots \\ \sim 0 & \sim 0 & \dots & \sim 0 \end{pmatrix},$$

kde ~ 1 značí interval blízký $[1, 1]$ obsahující 1 a ~ 0 značí interval blízký $[0, 0]$ obsahující 0. Zavedli jsme podmínku, že při vyjadřování i-té proměnné pomocí nějakého řádku, nesmí tento řádek mít koeficient obsahující 0 na pozici i. Jediná možnost, jak v tomto případě přiřadit řádky k proměnným, je řádek i k i-té proměnné. Tím vlastně provedeme Jacobiho metodu pro čtvercový případ na horní čtverec našeho přeurčeného systému. Také je zde možnost spodní nulové řádky systému obsahující 0 vynechat a na horní čtverec uplatnit některou iterační metodu pro čtvercové systémy. Vynecháním může ale dojít ke ztrátě informací. Například, že systém není řešitelný. Pro malé poloměry intervalů dává metoda i tak solidní výsledky.

8.4 Určení počátečního odhadu obálky

Iterační metody využívají při svém výpočtu již získaný odhad obálky množiny řešení zadaného systému. Existuje několik metod, jak předběžnou obálku spočítat. První a nejjednoduší je pro intervalový lineární systém $\boldsymbol{A}x = \boldsymbol{b}$ vyřešit středovou soustavu

$$A^c x = b^c$$

a výsledný vektor x obalit nějakým intervalem. Otázkou je jaký poloměr intervalu zvolit. Jako vstup iteračních metod stačí pro poloměry koeficientů menší než 0.1 prakticky použít poloměr v řádu desítek až stovek.

Další metodou je využít předpodmínění. Pokud matice předpodmíněného systému s malými poloměry vypadá tak, jak matice zobrazená v předchozí sekci, na

pravé straně se v horních n koeficientech nachází přibližné řešení systému. Typicky je o něco užší, můžeme si však vzít jeho střed a obalit ho nějakým větším intervalem. Tato metoda je sice výpočetně náročnější, ale většinou v iteračních metodách stejně předpodmiňujeme a odhad tak získáme navíc.

Třetí metodou je využít odhadu používaného pro Krawczykovu metodu. Z již předpodmíněného systému vezmeme horní čtvercový systém. V jeho množině řešení se určitě nachází řešení celého původního systému, protože odebráním řádků můžeme množinu řešení jen zvětšit. Pro tento čtvercový systém použijeme inicializační krok pro Krawczykovu metodu. Krawczykovu metodu samotnou zde nepopisujeme. Pro její popis nutno nahlédnout do [6]. Zabývat se budeme pouze inicializačním krokem. Dá se ukázat, že pokud systém splňuje určité podmínky, množina řešení v počáteční obálce zaručeně leží. Mějme nyní takto získaný čtvercový systém $\boldsymbol{A}x = \boldsymbol{b}$, kde matice $\boldsymbol{A}$ je rozměrů $n \times n$. Definujme

$$\boldsymbol{E} = I_n - \boldsymbol{A}.$$

Pokud platí $\|\boldsymbol{E}\| < 1$, můžeme počáteční odhad určit jako

$$\boldsymbol{x}_i^{(0)} = \frac{[-1,1]\,\|\boldsymbol{b}\|}{1 - \|\boldsymbol{E}\|} \quad \text{pro } (i = 1, \ldots, n).$$

Pro intervalovou maticovou normu

$$\|\boldsymbol{A}\| = \max_i \sum_j |\boldsymbol{A}_{ij}|.$$

9. Převedení na čtvercový případ

V této kapitole se podíváme na to, jak lze přeurčený intervalový lineární systém převést na čtvercový systém. Pro čtvercové případy existuje mnohem variabilnější škála algoritmů na jejich řešení. Převedením na čtvercový případ tak získáme přístup k dalším metodám – Krawczykowa iterační metoda, metody využívající maticové rozklady apod.

9.1 Metoda nejmenších čtverců

Také pro přeurčené intervalové lineární systémy se dá použít metoda nejmenších čtverců. Mějme systém intervalový lineární $\boldsymbol{A}x = \boldsymbol{b}$ a jeho množinu řešení

$$S = \{x;\ Ax = b, A \in \boldsymbol{A}, b \in \boldsymbol{b}\}.$$

Pak platí, že $S \subseteq S^*$, kde

$$S^* = \{x;\ A^TAx = A^Tb, A \in \boldsymbol{A}, b \in \boldsymbol{b}\}.$$

Vlastně aplikujeme metodu nejmenších čtverců na každý systém $Ax = b$ obsažený v $\boldsymbol{A}x = \boldsymbol{b}$. Platí $S \subseteq S^*$, protože množina S^* může být větší, neboť metoda nejmenších čtverců nám vydá řešení, i když daný systém řešení nemá. Dále pak platí

$$S \subseteq S^* \subseteq S^{**},$$

kde

$$S^{**} = \{x;\ Ax = b, A \in \boldsymbol{A}^T\boldsymbol{A}, b \in \boldsymbol{A}^T\boldsymbol{b}\}.$$

Zde řešíme systém $\boldsymbol{A}^T\boldsymbol{A}x = \boldsymbol{A}^T\boldsymbol{b}$. Jestliže měla původní matice $\boldsymbol{A}$ rozměry $m \times n$, má matice $\boldsymbol{A}^T\boldsymbol{A}$ rozměry $n \times n$. Dostáváme tedy čtvercový systém. Tato metoda má však několik nevýhod. Narozdíl od původní soustavy má totiž vždy řešení, což pro nás může být nevýhodné. Další nevýhoda je, že matici násobíme maticí a díky velkému množství operací násobení intervalů může dojít k velkému nárůstu intervalových koeficientů ve výsledné matici, a tudíž i k zvětšení množiny řešení. Nehledě k tomu, že celá soustava má kvadratické podmínkové číslo, to znamená, že nepřesnosti v počítání se na výsledku projeví až kvadraticky. Díky zmíněným nevýhodám je metoda prakticky ztěží použitelná.

9.2 Předpodmínění

Řekli jsme si, že při řešení soustavy $\boldsymbol{A}^T\boldsymbol{A}x = \boldsymbol{A}^T\boldsymbol{b}$ dochází k velkému nárůstu řešení. Máme však možnost původní soustavu $\boldsymbol{A}x = \boldsymbol{b}$ předpodmínit jako v případě Gaussovy eliminace. Použijeme stejnou matici C. Dostáváme tak soustavu

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

Tato soustava již dává lepší výsledky, nicméně pořád v praxi nepoužitelné, jak ukážeme v části o srovnání jednotlivých metod.

9.3 Převedení na nadčtverec

Naštěstí metoda nejmenších čtverců se dá ekvivalentně vyjádřit jinak. Platí totiž následující pozorování.

Pozorování (Převedení na nadčtverec). *Mějme intervalový lineární systém* $\boldsymbol{A}x = \boldsymbol{b}$*, kde* $\boldsymbol{A}$ *má rozměry* $m \times n$*, pro* $m \geq n$*. Platí*

$$S \subseteq S^* \subseteq \boldsymbol{x},$$

kde $\boldsymbol{x}$ *je intervalový vektor, který je částí řešení intervalové soustavy*

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

Důkaz. Pro neintervalový systém můžeme soustavu $A^T Ax = A^T b$ přepsat ekvivalentně jako soustavu

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

neboť po roznásobení dostaneme 2 soustavy

$$\begin{aligned} y + Ax &= b, \\ A^T y &= 0. \end{aligned}$$

Z první rovnice vyjádříme $y = b - Ax$ a dosadíme do druhé. Dostaneme

$$A^T(b - Ax) = 0.$$

Po roznásobení a jednoduché úpravě dostaneme $A^T Ax = A^T b$. Pro druhý směr můžeme kroky zopakovat pozpátku. Jelikož si můžeme představit intervalový systém jako množinu systémů, můžeme tuto úpravu provést pro každý systém zvlášť a dohromady zapsat jako intervalovou soustavu ve znění pozorování. Platí $S^* \subseteq \boldsymbol{x}$, protože, když při řešení intervalové soustavy bereme koeficienty z intervalové matice nezávisle na sobě, může dojít k určitému nadhodnocení.

□

Řešíme čtvercový intervalový systém, kde matice nalevo je řádu $(m+n) \times (m+n)$. Pokud je m výrazně větší než n dostaneme systém mnohem větší než je původní. Například pro systém 50×3 dostaneme nový systém 53×53. Systém je opět hůře podmíněn než původní systém, avšak nemá tendenci rešení tolik natahovat, díky tomu, že zde nepoužíváme násobení maticemi. Opět je zde však ten nedostatek, že metoda vydá řešení, i když žádné řešení neexistuje. Metoda toolboxu Intlab `verifylss` umožňuje řešit i přeurčené intervalové lineární systémy. Pokud nahlédneme do zdrojových kódů, zjitíme, že toto je právě metoda, kterou Intlab používá. Po převedení na tento čtvercový systém, je spuštěna upravená Krawczykova metoda, která je detailněji popsána v části o Intlabu.

9.4 Využití podčtverců

Zatím jsme $\boldsymbol{A}x = \boldsymbol{b}$ převáděli na nadčtverec. Jako vlastní dosud nepublikovanou metodu navrhujeme postup, kdy řešení budeme skládat jako průnik řešení jednotlivých čtvercových podsystémů. V každém kroku vybereme náhodně řádky matice $\boldsymbol{A}$, tak aby tvořili čtvercovou matici. Dále doplníme vektor pravých stran prvky vektoru $\boldsymbol{b}$, které odpovídají příslušným řádkům. Celý algoritmus vypadá takto.

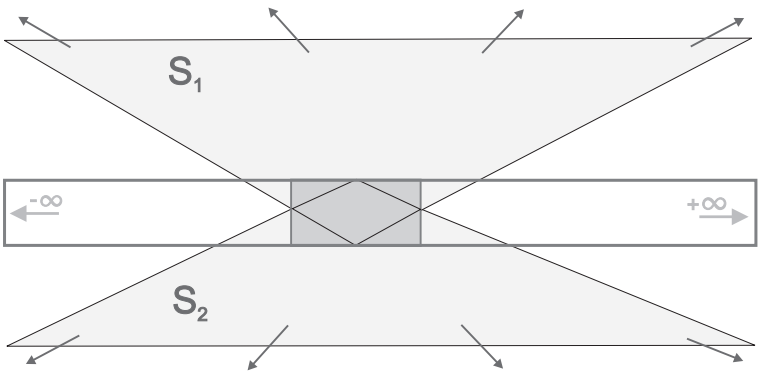
Algoritmus (Náhodné podčtverce). *Nechť $\boldsymbol{A}x = \boldsymbol{b}$ je intervalový lineární systém, kde matice $\boldsymbol{A}$ je řádu $m \times n$ pro $m > n$.*

1. *$\boldsymbol{x}$ je n-prvkový intervalový vektor s koeficienty $[-\infty, +\infty]$.*
2. *Opakujme kroky 3. až 5. dokud není konec.*
3. *Vyberme náhodně n-prvkovou množinu indexů $K \subset \{1, 2, \ldots, m\}$.*
4. *$\boldsymbol{C} = \{\boldsymbol{A}_{i*} \mid i \in K\}$, $\boldsymbol{d} = \{\boldsymbol{b}_{i*} \mid i \in K\}$.*
5. *Vyřešme $n \times n$ čtvercovou soustavu $\boldsymbol{C}y = \boldsymbol{d}$, nechť $\boldsymbol{y}$ je její řešení.*
6. *$\boldsymbol{x} = \boldsymbol{x} \cap \boldsymbol{y}$.*
7. *Pokud je průnik prázdný končíme, systém nemá řešení.*

Pro množinu řešení S vždycky platí

$$S \subseteq \boldsymbol{x}.$$

Čím více průniků čtvercových podsystémů prozkoumáme, tím lepší odhad na výsledek získáme. Aby měl původní přeurčený systém řešení, musí mít všechny řešení čtvercových podsystémů neprázdný průnik. Může se stát, že výsledek bude značně nadhodnocen. Může totiž teoreticky nastat případ, který ukazuje obrázek 9.1. Písmeny S_1 a S_2 jsou označeny množiny řešení dvou různých podsystémů. Obdélník vyplněný šedou barvou naznačuje ideální obal průniku těchto dvou množin. Obdélník se šedým obvodem označuje velmi širokou obálku, kterou můžeme dostat, pokud pronikneme obaly S_1 a S_2.



Obrázek 9.1: Selhání metody podčtverců

Počet všech čtvercových podsystémů je $\binom{m}{n}$. Z toho vyplývá, že se metoda hodí pro hodně malé n, nebo pro případy, kdy m a n jsou si blízké. Určitou nevýhodou

také je, že pokud vybíráme čtvercové podsystémy náhodně, nevíme, kdy metodu ukončit. Dokud neprozkoumáme všech $\binom{m}{n}$ podsystémů, nemáme jistotu, zda jsme dostali nejlepší možnou obálku množiny řešení, kterou nám metoda může nabídnout, či zda systém vůbec řešení má. Máme tedy na výběr. Buď prozkoumáme samostatně všechny možné průniky intervalových obálek všech podsystémů, nebo můžeme metodu spustit zároveň s jinou metodou a podsystémy vybírat náhodně jako prostředek možného zlepšení intervalové obálky. Použitím metody v konkrétních případech se budeme zabývat v kapitole o testování.

10. Porovnání metod

V této předposlední a nejdelší kapitole porovnáme z různých hledisek algoritmy, které jsme v minulých kapitolách představili. Abychom nemuseli porovnávat všechny najednou, porovnáme je nejdříve v rámci skupin, které jsou dány kapitolou, kam dotyčné algoritmy patří. Po vybrání nejlepších a nebo zajímavě se chovajících algoritmů je porovnáme mezi sebou. Testovat budeme z různých hledisek. Zajímat nás bude především rychlost algoritmů a šířky výsledných intervalových obálek. Také nás bude zajímat, jak se algoritmy chovají pro neřešitelné systémy.

10.1 Testovací soustava

Všechny testy byly prováděny na přenosném počítači Dell Inspiron 6400.

- Procesor Intel T2400, Core Duo, 1.83 MHz
- Paměť 2.50 GB, 987 MHz

Testováno pro Matlab R2009a.

10.2 Metodika testování

Nejprve popíšeme metodiku testování. Budeme testovat metody jednak na systémech řešitelných, abychom zjistili, jak těsnou obálku jsou schopny vypočítat. Následně na systémech, které řešení nemají, abychom odhalili případné zvláštní chování algoritmů, ze kterého se dá neřešitelnost systému přímo poznat. Řešitelné systémy generujeme náhodně, a to tak, že nejprve vygenerujeme matici A reálných čísel z daného intervalu $[-x, x]$, typicky pro $x = 25$. Stejně vygenerujeme vektor řešení x a spočítáme pravou stranu soustavy jako $b = Ax$. Poté z A a b vytvoříme intervalovou matici $\boldsymbol{A}$ a intervalový vektor $\boldsymbol{b}$, tím, že obalíme jejich jednotlivé koeficienty náhodnými malými intervaly podle zadaného poloměru. Pokud zvolíme například poloměr 10^{-4} znamená to, že pro každý koeficient matice A je příslušný koeficient matice $\boldsymbol{A}$ interval

$$[a_{ij} - \epsilon, a_{ij} + \epsilon], \text{ kde } \epsilon \leq 10^{-4}.$$

Vygenerovaná matice A má pro rozměry $m \times n$ hodnost n s pravděpodobností blížící se 1. Díky dopočítání pravé strany máme jistotu, že soustava bude mít řešení, a to minimálně jedno blízké x (díky možným zaokrouhlovacím chybám). Pokud zvolíme poloměry intervalů malé, bude mít soustava řešení omezené. Většina metod totiž není schopna rozpoznat neomezenou množinu řešení (Rohnova metoda, iterační metody, metoda nejmenších čtverců), proto nás zajímá hlavně omezená množina řešení. Systémy bez řešení generujeme tak, že vygenerujeme matici A a vektor pravé strany b (tentokrát negenerujeme x) a převedeme je stejným způsobem na intervalové $\boldsymbol{A}, \boldsymbol{b}$. Opět pro malé poloměry intervalů nebude mít systém řešení s pravděpodobností blížící se 1.

Budeme testovat na systémech do řádu zhruba 1000. Počet testů bude záviset na velikosti systému a na rychlosti algoritmu. Pro malé matice a rychlé algoritmy to budou řádově tisíce až stovky. Pro pomalé algoritmy a rozsáhlé matice stovky až desítky. Jak již bylo řečeno, pokud to bude mít smysl, budeme testovat pro systémy řešitelné i neřešitelné zvlášť. Poloměry intervalů budeme nejčastěji brát $\leq 10^{-1}, \leq 10^{-4}$. Jiné poloměry budeme používat, pokud budeme chtít ukázat mezní chování algoritmu. Některé algoritmy vyžadují jako vstup malý kladný vektor ϵ. Ten budeme brát s koeficienty $0.01, 10^{-5}$ (v textu ho pro zkrácení budeme značit $\epsilon = 0.01$ nebo $\epsilon = 10^{-5}$). Jiné koeficienty opět použijeme pokud budeme chtít ukázat mezní chování.

Při srovnání šířek výsledných intervalových obálek budeme porovnávat vůči nějaké existující metodě. Nejčastější to bude metoda Intlabu `verifylss`. Obálku metody vůči níž porovnáváme budeme značit x^{test}. Šířku intervalů budeme měřit relativně po jednotlivých složkách a pak ji ještě oproti všem složkám zprůměrujeme, tedy pro nějaký získaný obal x délky n počítáme poměr jako

$$\frac{1}{n}\sum_{i=1}^{n}\frac{\mathrm{w}(x_i)}{\mathrm{w}(x_i^{test})}.$$

Čím nižší číslo tím lépe. Pro obálku ekvivalentní s x^{test} bude tento poměr rovný 1.

Pro sledování doby běhu algoritmu budeme používat vestavěné funkce Matlabu – `tic`, `toc`. Funkce `tic` nastaví časovač a funkce `toc` vrátí aktuální hodnotu časovače.

10.3 Jednotlivé skupiny algoritmů

10.3.1 Rohnova metoda

V kapitole Rohnův obal je uvedena pouze jedna metoda, nicméně výsledný obal určujeme pomocí vektoru d. Ten můžeme najít dvěma způsoby, buď přímo, nebo iteračně. Nás zde bude zajímat, která metoda je výhodnější. V přímé metodě používáme pro výpočet d Matlabovskou funkci `linsolve`.

Nejprve se podíváme na to, v kolika z testovaných případů je přímá metoda rychlejší než iterační. Výsledky ukazuje tabulka 10.1. Z ní je vidět, že pro malé matice je přímá metoda rychlejší. Čím menší poloměr intervalů, tím klesají rozměry soustav, pro které je přímá metoda rychlejší. Naopak podle dalších testů, čím menší ϵ, tím tyto rozměry stoupají. Zajímavou anomálii tvoří poloměry 0.1, kde čím menší ϵ, tím je přímá metoda rychlejší. Rozdíly v rychlostech však nejsou pro malé matice nijak dramatické, což znázorňuje následující tabulka 10.2. Tabulka ukazuje, na jakém desetinném místě se hodnoty časů od sebe liší (např. 3 znamená tisíciny). Pokud se v nějakém políčku vyskytne více hodnot, znamená to, že se liší přibližně stejný počet na daných řádech. Pokud tedy chceme porovnat metody z hlediska časového, nejprve se podíváme do tabulky 10.1, abychom zjistili, která metoda je rychlejší, a poté do tabulky 10.2, abychom zjistili o kolik. Pro malé matice jsou metody téměř stejně rychlé. Rozdíly mezi metodami jsou zhruba v řádu stotisícin, což je pod hranicí přesnosti metod sledujících čas. Rozdíly nastanou až u velkých matic (řád větší jak 100), kdy iterační metoda postupně převládne. Pro poloměry intervalů okolo 0.1 pro matice rozměrů 200×170 často nastane případ,

kdy neplatí podmínka $\varrho(G) < 1$. Důsledkem je, že se iterační metoda nezastaví (respektive i zastaví, ale vydá téměř nekonečné kladné d) a přímá metoda vrací záporný vektor d. V tabulce jsou označeny max. U menších poloměrů již tyto problémy nenastávají.

Iterační metoda vykoná i pro velké matice řádově desítky kroků. Průměrné počty kroků ukazuje tabulka 10.3. Více kroků proběhne, pokud jsou poloměry intervalů větší. Čím menší ϵ tím více kroků je zapotřebí.

Nyní nás budou zajímat přímo vektory d. Jelikož určují horní a dolní mez výsledné obálky, platí čím menší d tím lépe. Při testování nám ve všech případech vyšlo, že d nalezené iterační metodou je menší. Výjimku tvoří extrémně nízké hodnoty ϵ a r. Tabulka 10.4 ukazuje v jakých řádech se liší maximální rozdíly v koeficientech. Celkově se tedy iterační metoda chová výhodněji a budeme dále její použití preferovat.

Demonstrovali jsme, jak lze obálku získanou Rohnovou metodou iterativně vylepšit. Také jsme si ukázali, že bohužel nelze užít postačující podmínka ϵ. V následujícím testu jsme otestovali pro stejné rozměry a parametry soustav Rohnovu iterační metodu pro 10, 100 a 1000 kroků. Při porovnání s Rohnovou neiterační metodou nemá smysl testovat časy běhů, neboť jsou metody přibližně $10\times$, $100\times$ a $1000\times$ pomalejší. Budeme tedy testovat jen šířku výsledné obálky. Pro různé parametry systémů se metoda chová velmi podobně. Tabulka 10.5 ukazuje poměry pro systémy s $r \leq 10^{-4}$ a $\epsilon = 10^{-5}$. Zatímco pro malé systémy je zlepšení obálky rapidní, pro větší matice i hodně iteračních kroků přinese jen malé zlepšení. Pro malé systémy má cenu vykonat kolem 100 kroků. Pro velké systémy stačí desítky – další kroky by již byly zbytečné.

10.3.2 Gaussova eliminace

Na obálce získané Gaussovou eliminací se často projeví velké množství intervalových operací, které je nutné během výpočtu vykonat. Zvláště, když nepoužijeme předpodmínění. Díky tomu, že se během zpětné substituce počítá řešení od x_n k x_1 s pomocí stále delšího vektoru x, dostáváme pro koeficienty s indexem blízkým 1 již značně nepřesné hodnoty. Například pro matici 15×13 o poloměru $\leq 10^{-3}$ jsme pomocí Gaussovy eliminace bez předpodmínění dostali obálku, kterou ukazuje tabulka 10.6 (v prvním sloupci jsou středy intervalů a ve druhém poloměry intervalů jednotlivých koeficientů).

Pro větší rozměry matic se nám často stane, že algoritmus Gaussovy eliminace skončí, neboť v průběhu eliminace již nelze nanalézt žádný pivot neobsahující prvek 0. Může se také stát, že nenalezneme konečný průnik pro x_n. Tabulka 10.7 ukazuje na kolika systémech GE bez předpodmínění skončí v průběhu eliminace (E) a na kolika v průběhu zpětné substituce (S). Systémy, kterými GE úspěšně projde a získáme intervalový vektor označíme (V). Je vidět, že i pro velmi malé matice dochází k takovému natažení intervalů, že nelze v algoritmu dále pokračovat. Situace se výrazně zlepší pokud použijeme předpodmínění (znázorněno v tabulce 10.8). Pro matice s velkými poloměry je GE použitelná do řádu zhruba 40. S malými poloměry GE doběhne i pro velké matice.

Již víme, že pokud při zpětné substituci pro přeurčený systém při počítání průniků proměnné x_n dostaneme prázdný průnik, systém nemá řešení. Podíváme se tedy, jak často je GE schopna rozpoznat systém bez řešení. Nejprve opět zkusíme

matice	$\epsilon = 0.01$ $r \leq 0.1$	$\epsilon = 0.01$ $r \leq 10^{-4}$	$\epsilon = 10^{-5}$ $r \leq 0.1$	$\epsilon = 10^{-5}$ $r \leq 10^{-4}$
5×3	98.9	99.2	99.2	98.6
15×10	97.2	96.4	99.0	96.9
25×21	94.1	2.2	98.0	1.9
35×23	6.2	1.5	97.7	1.0
45×31	2.3	1.6	97.6	1.3
50×35	2.5	1.3	96.5	1.3
100×87	1.6	0	98.4	0

Tabulka 10.1: Rohn – v kolika % případů je přímá metoda rychlejší

matice	$\epsilon = 0.01$ $r \leq 0.1$	$\epsilon = 0.01$ $r \leq 10^{-4}$	$\epsilon = 10^{-5}$ $r \leq 0.1$	$\epsilon = 10^{-5}$ $r \leq 10^{-4}$
5×3	5	5	5	5
15×10	5/6	5	5	5
25×21	5/6/7	5	5	5
35×23	6	5	5	5
50×35	5	5	5	5
73×55	4	4	5	4
100×87	4	4	4	4
200×170	max	3	max	3
500×487	-	2	-	2
1000×967	-	1	-	1

Tabulka 10.2: Rohn – řády rozdílů časů přímé a iterační metody

matice	$\epsilon = 0.01$ $r \leq 0.1$	$\epsilon = 0.01$ $r \leq 10^{-4}$	$\epsilon = 10^{-5}$ $r \leq 0.1$	$\epsilon = 10^{-5}$ $r \leq 10^{-4}$
5×3	2	2	3.39	2
15×10	2.32	2	4.85	2
25×21	3.47	2	7.12	2
35×23	3	2	6.1	2
50×35	4	2	7.98	2
73×55	5.42	2	11.2	2
100×87	14.78	2	30.97	2
200×170	max	2	max	2
500×487	-	2	-	3
1000×967	-	2	-	4

Tabulka 10.3: Rohn – průměrný počet kroků iterační metody

matice	$\epsilon = 0.01$ $r \leq 0.1$	$\epsilon = 0.01$ $r \leq 10^{-4}$	$\epsilon = 10^{-5}$ $r \leq 0.1$	$\epsilon = 10^{-5}$ $r \leq 10^{-4}$
5×3	3,4	7	6	7
15×10	2,3	7	6	7
25×21	2,3	6	5	7
35×23	2/3	6	5/6	7
50×35	3	6	6	7
73×55	2	6	5	7
100×87	2	5	5	6
200×170	max	5	max	6
500×487	-	4	-	6

Tabulka 10.4: Rohn – řády absolutních maximálních rozdílů mezi d

systém	10	100	1000
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

Tabulka 10.5: Rohn – poměry obálek Rohnovy iterační metody (10, 100 a 1000 kroků)

proměnná	střed	délka
x_1	0.1479	227.6698
x_2	15.2091	172.5929
x_3	11.1031	68.4653
x_4	9.7809	64.1056
x_5	-8.8168	27.2234
x_6	25.8164	25.8398
x_7	-19.0444	30.4596
x_8	-22.0799	11.0313
x_9	1.9649	12.1172
x_{10}	-19.1817	11.6841
x_{11}	-20.9670	1.9153
x_{12}	-4.6988	3.5407
x_{13}	3.1223	4.3894

Tabulka 10.6: GE – nárůst výsledné obálky bez předpodmínění

	$r \leq 10^{-1}$			$r \leq 10^{-4}$		
systém	(V)	(E)	(S)	(V)	(E)	(S)
5×3	100			100		
15×10	35	51	17	100		
25×21			100	9	33	58
35×23			100	1	3	96
50×35			100			100
73×55			100			100
100×87			100			100
200×170			100			100
500×470			100			100

Tabulka 10.7: GE bez předpodmínění – procento předčasných ukončení

	$r \leq 10^{-1}$			$r \leq 10^{-4}$		
systém	(V)	(E)	(S)	(V)	(E)	(S)
5×3	99	1		100		
15×10	94		6	100		
25×21	73	2	25	100		
35×23	76	1	23	100		
50×35	49	2	49	100		
73×55	15		85	100		
100×87			100	100		
200×170			100	100		
500×470			100	100		

Tabulka 10.8: GE s předpodmíněním – procento předčasných ukončení

	$r \leq 10^{-1}$				$r \leq 10^{-4}$			
systém	(V)	(E)	(S)	(N)	(V)	(E)	(S)	(N)
5×3	1			99				100
15×10	20	58	22					100
25×21			100		8	34	57	1
35×23			100			8	92	
50×35			100				100	
73×55			100				100	
100×87			100				100	
200×170			100				100	
500×470			100				100	

Tabulka 10.9: GE bez předpodmínění – detekce neřešitelnosti

GE bez předpodmínění. Výsledky ukazuje tabulka 10.9. Detekované neřešitelné systémy označujeme (N). Pro větší systémy dojde díky intervalovým operacím k nárůstům intervalových koeficientů, takže již průnik pro x_n není neprázdný.

I když použijeme předpodmínění, situace se nezlepší. Při jeho použití dojde vždy k určitému nárůstu množiny řešení, takže i neřešitelný systém, začne mít nějaké řešení. Pro detekci neřešitelnosti malých systémů (řád kolem 10) s malými poloměry se hodí GE bez předpodmínění.

10.3.3 Iterační metody

Pokud nepředpodmiňujeme, existuje několik možností, jak v obou iteračních metodách přiřazovat proměnné k jednotlivým řádkům. Pokud předpodmíníme, redukuje se problém na iterační metodu pro čtvercový případ. Bohužel metody bez použití předpodmínění nám dávají příliš velkou obálku množiny řešení. Je to způsobeno velkým množstvím intervalových operací při vyjadřování proměnných. Pokud do iterační metody dosadíme příliš přesný odhad řešení, může se stát, že se ho již iteračně nepodaří vylepšit. Pokud naopak dosadíme odhad řešení příliš nadhodnocený, může se stát, že metody nezkonvergují do úzké obálky. Budeme testovat následující metody (v závorce uvádíme zkratky, pod kterými je budeme kvůli zkrácení v textu prezentovat).

- Jacobiho metoda s náhodným výběrem řádků ($Jrand$)
- Jacobiho metoda s předpodmíněním ($Jpre$)
- Gauss-Seidlova metoda s náhodným výběrem řádků ($GSrand$)
- Gauss-Seidlova metoda s vyjadřováním podle všech řádků ($GSall$)
- Gauss-Seidlova metoda s předpodmíněním ($GSpre$)
- Lineární programování ($Linprog$)

Jako počáteční řešení budeme používat odhady pomocí středové soustavy (pro velikosti poloměrů, kterými se zabýváme stačí poloměr předběžné obálky rovný 100) i pomocí Krawczykovy metody. Metody s příponou *rand* používají jako zastavující podmínku pevně daný počet kroků.

Před vlastním testováním zmíníme jeden problém při předpodmiňování, na který jsme narazili. Pokud mají matice poloměry koeficientů blízké 10^{-1} nebo 10^{-2}, může se stát, že předpodmíněná matice obsahuje prvky 0 mimo diagonálu, ale i na diagonále, tudíž iterační metody nelze užít. Pro matice s nižšími poloměry již tyto problémy nenastávají, proto musíme iterační metody testovat pro tyto matice. Dalším problémem je, že pro matice s poloměrem intervalů blízkých 0.1 nám pro Krawczykův odhad neplatí podmínka $\|\boldsymbol{E}\| < 1$, metodu tedy nemůžeme pro odhad užít.

Abychom byli schopni počet kroků *rand* metod pro testování odhadnout, nejprve se podíváme kolik kroků iterační metody s předpodmíněním vykonají. Iterační metody nevykonají mnoho kroků. Tabulka 10.10 ukazuje průměrné počty kroků pro různá zastavující ϵ, a to 0.01 a **eps**[1]. Odhad bereme pomocí Krawczykovy metody. Je vidět, že čím menší ϵ, tím více iteračních kroků metoda vykoná.

[1]Podle nápovědy v Matlabu je $\mathbf{eps} = 2^{-52}$.

Použití **eps** je pro praktické účely zbytečně nízké. Ukazuje spíš limitní počet kroků pro velmi malé ϵ. Velikost poloměrů intervalů systému volíme $\leq 10^{-4}$. Z tabulky je vidět, že obě metody provedou přibližně stejně kroků pro malé ϵ. Pro větší ϵ *GSpre* provede výrazně méně kroků.

Nyní již přistupme k testování šířek získaných obálek. Srovnávat budeme iterační metody s obalem získaným metodou `verifylss`. Iterační metody bez předpodmínění *Jrand* a *GSrand* pouštíme 20 iteračních kroků. U metody *GSall* používáme, stejně jako u *pre* metod, zastavující podmínku ϵ. Tabulka 10.11 ukazuje výsledky pro náhodné systémy s poloměrem $\leq 10^{-3}$ a pro $\epsilon = 0.01$. Používáme odhad pomocí středové soustavy.

Z této tabulky plyne několik závěrů. Metody s příponou *rand* dávají stejné řešení. Jelikož číselné hodnoty jsou ve většině případů totožné, uvažujeme z toho, že se prvotní odhad množiny řešení nepodařilo vylepšit. Metody nepoužívající předpodmínění nemá tedy smysl používat díky obrovské šířce výsledné obálky. Podle poměrů šířek obálek metody s příponou *pre* konvergují ke stejnému řešení. Na poloměrech koeficientů soustavy ani zvolených ϵ příliš nezáleží, poměry se i tak pohybují v řádech jednotek až desítek.

Z těchto metod tedy vítězí *Jpre* a *GSpre*. Závěrečným porovnáním bude průměrná doba běhu těchto dvou algoritmů. Zobrazena je v tabulce 10.12. Testováno na systému s poloměry $\leq 10^{-3}$. Pro malé systémy je rychlejší metoda *Jpre* o setiny vteřin, avšak pro velké systémy je *GSpre* rychlejší až o vteřiny (časový zlom je v tabulce oddělen vodorovnou čárou). Čím menší ϵ, tím déle trvá výpočet. Pokud ještě snížíme poloměry, časové rozdíly mezi metodami se začnou rozplývat, dokonce *Jpre* začne být nepatrně rychlejší. Pro další porovnávání budeme testovat metodu *GSpre*.

V úvodu k iteračním metodám bylo řečeno, že pokud je průnik dvou po sobě jdoucích obálek prázdný, systém nemá řešení. Bohužel při testování jsme zjistili, že tento případ nastává zřídka, a ještě k tomu pro malé matice. V případě nepředpodmíněných systémů bude neprázdný průnik způsoben zřejmě relativně velkými koeficienty. V případě předpodmíněných nárůstem množiny řešení vlivem předpodmínění. Nebude tedy spolehlivým ukazatelem neřešitelnosti systému.

10.3.4 Převedení na čtvercový případ

V kapitole převedení na čtvercový případ jsme si představili dva způsoby, jak lze řešit metodu nejmenších čtverců převedením na čtvercový případ. Prvním způsobem jsme dostali matici menší, ale bylo vyžadováno násobení dvou intervalových matic. Aplikací druhé metody jsme dostali matici větší. Při té nemá smysl předpodmiňovat, neboť pokud pro řešení nadčtverce využíváme iterační metody, předpodmínění proběhne před samotnou iterační metodou a předchozí předpodmínění by jen zbytečně roztáhlo množinu řešení. Obě metody vracejí řešení, i když soustava žádné řešení nemá. V prvním případě nicméně ani předpodmínění problém nezachrání, neboť dostaneme řešení zbytečně roztáhlé, jak ukazuje následující příklad. Pro systém $\boldsymbol{A}x = \boldsymbol{b}$

	$\epsilon = 0.01$		$\epsilon =$ **eps**	
systém	*Jpre*	*GSpre*	*Jpre*	*GSpre*
15×10	2.07	2.05	6.10	5.26
25×21	2.28	2.18	6.98	6.18
35×23	2.36	2.27	7.31	6.30
50×35	2.46	2.37	7.94	6.73
73×55	2.94	2.76	8.40	7.20
100×87	3.10	3.00	11.9	9.00

Tabulka 10.10: Iterační metody – průměrný počet kroků

matice	*Jrand*	*Jpre*	*GSRand*	*GSall*	*GSpre*
5×3	9.4×10^4	1.6960	9.4×10^4	5×10^4	1.6959
15×10	4.11×10^4	10.7055	4.11×10^4	4.11×10^4	10.7031
25×21	1.44×10^4	33.2188	1.44×10^4	1.44×10^4	33.2105
35×23	1.75×10^4	5.915	1.75×10^4	1.75×10^4	5.9149
50×35	1.15×10^4	9.7655	1.15×10^4	1.15×10^4	9.7639
73×55	6.58×10^3	4.63	6.58×10^3	6.58×10^3	4.63
100×87	2.83×10^3	7.99	2.83×10^3	2.83×10^3	7.99

Tabulka 10.11: Iterační metody – poměry získaných obálek

	$\epsilon = 0.01$		$\epsilon = 10^{-5}$	
systém	*Jpre*	*GSpre*	*Jpre*	*GSpre*
5×3	0.0213	0.0219	0.023	0.0242
15×10	0.0518	0.059	0.0659	0.0741
25×21	0.1029	0.1174	0.1424	0.1543
35×23	0.1224	0.1349	0.1609	0.1706
50×35	0.1954	0.2059	0.267	0.2789
73×55	0.3441	0.3610	0.4471	0.478
100×87	0.8877	0.8537	1.2584	1.1098
200×170	1.7962	1.7024	2.15	2.0447
500×470	12.2193	10.2190	13.8454	11.4155

Tabulka 10.12: Iterační metody s předpodmíněním – průměrné časy

$$\boldsymbol{A} = \begin{pmatrix} [\ 16.9998,\ 17.0002\ ] & [\ 28.9993,\ 29.0007\ ] & [\ 40.9992,\ 41.0008\ ] \\ [\ 8.9994,\ 9.0006\ ] & [\ 13.9999,\ 14.0001\ ] & [\ 10.9991,\ 11.0009\ ] \\ [\ 15.9991,\ 16.0009\ ] & [\ 25.9999,\ 26.0001\ ] & [\ 3.9993,\ 4.0007\ ] \\ [\ 13.9998,\ 14.0002\ ] & [\ 17.9993,\ 18.0007\ ] & [\ 7.9990,\ 8.0010\ ] \\ [\ 12.9999,\ 13.0001\ ] & [\ 36.9992,\ 37.0008\ ] & [\ 20.9990,\ 21.0010\ ] \end{pmatrix},$$

$$\boldsymbol{b} = \begin{pmatrix} [\ 16.2107,\ 75.7893\ ] \\ [\ 27.9484,\ 60.0516\ ] \\ [\ -61.0726,\ 135.0726\ ] \\ [\ -14.6424,\ 102.6424\ ] \\ [\ -36.5122,\ 80.5122\ ] \end{pmatrix}.$$

dostáváme použitím metody nejmenších čtverců s násobením matic a předpodmíněním následující řešení

$$\boldsymbol{x}_1 = \begin{pmatrix} [\ -67.4786,\ 145.2822\ ] \\ [\ -87.6947,\ 42.3447\ ] \\ [\ -36.6750,\ 38.7357\ ] \end{pmatrix}.$$

Při převedení na nadčtverec dostaneme

$$\boldsymbol{x}_2 = \begin{pmatrix} [\ -9.0921,\ 17.9360] \\ [\ -6.8996,\ 4.7132] \\ [\ -3.6517,\ 4.0093] \end{pmatrix}.$$

Při porovnání časových složitostí druhá metoda je rychlejší (testováno pro matice do řádu 100×87) a pro tyto matice se časy lišili v průměru v tisícinách sekundy.

Nyní přejděme k metodě podčtverců. Pro malé rozměry se vyplatí testovat průniky všech možných podčtverců. Nutno podotknout, že jich je $\binom{m}{n}$. Všechny kombinace generujeme pomocí funkce Matlabu `nchoosek` [2]. V tabulce 10.13 jsou uvedeny přibližné časy pro malé matice. Čtvercové podmatice řešíme s pomocí metody Intlabu `verifylss`. Metodu se vyplatí používat, pokud m i n jsou si blízké i pro relativně velká m. Jestliže jsou rozměry matice velké, vybíráme předem daný počet čtvercových podsystémů, z jejichž řešení pronikáme výslednou obálku. Opět zde nelze použít zastavující podmínka s kladným vektorem ϵ. Důvod je stejný jako u Rohnovy iterační metody, kde vybíráme náhodné matice z intervalové matice nebo u iteračních metod, kde vybíráme náhodné řádky z matice. Otestovali jsme metodu podčtverců s počtem opakování 10, 50 a 100. Výsledky ukazuje tabulka 10.14. Pro malé matice $m \times n$ nedojde k příliš velkému zlepšení obálky s přibývajícím počtem iteračních kroků, neboť 10, 50 i 100 jsou relativně blízké kombinačnímu číslu $\binom{m}{n}$. Zlepšení nastane až s většími maticemi. Poměry obálek porovnáváme opět s metodou `verifylss`. Ukazuje se, že poměrně rozumná volba je, když pro systém testujeme m čtvercových podsystémů.

Pokud nalezneme podsystém, který nemá průnik s dosavadním řešením, pak celý původní systém nemá řešení. Testovali jsme pro systémy s poloměry

$$\leq 10^{-1}, \leq 10^{-2}, \leq 10^{-3}, \leq 10^{-4}.$$

[2] V nápovědě k Matlabu je uvedeno, že má smysl používat tuto funkci pro $n \max 15$.

Prohledávali jsme vždy 50 náhodných čtvercových podsystémů. Kromě případů pro neřešitelné systémy s poloměry $\leq 10^{-1}$ pro větší matice, se podařilo detekovat 100 % případů. Průměrné počty kroků potřebné k odhalení neřešitelnosti jsou uvedeny v tabulce 10.15. Výjimky jsou označeny symbolem $\times$, který znamená, že během provedených 50 kroků se nepodařilo neexistenci řešení detekovat. Čím menší poloměry matic, tím stačí k určení neřešitelnosti menší počet kroků.

10.3.5 Lineární programování

Pokud pro určení přesného obalu řešení systému $\boldsymbol{A}x = \boldsymbol{b}$, kde matice $\boldsymbol{A}$ má rozměry $m \times n$, prohledáváme všechny ortanty ($2n \times 2^n$ úloh lineárního programování), je časová náročnost značná. Jak ostatně ukazuje tabulka přibližných časů 10.16. V případě, že prohledáváme jen některé ortanty určené znaménkovým vektorem předem vypočítané obálky (zde metodou `verifylss`) dostáváme přijatelnější časové výsledky (10.17). Opět jsou zde přibližné časy. Pro řešení úlohy lineárního programování používáme Rohnovu metodu `verlinprog`. Na konci této kapitoly až budeme srovnávat jednotlivé šíře výsledných obálek všech algoritmů, budeme pro malé matice porovnávat s lineárním programováním. Pro větší matice budeme z časových důvodů porovnávat s metodou `verifylss`.

10.4 Celkové porovnání

Pro celkové porovnání všech metod jsme vybrali zástupce všech skupin. Testovat budeme následující algoritmy. Pro snadné označení v textu je opět označíme zkratkami.

- Gauss-Seidlova metoda s předpodmíněním ($GSpre$).
- Gaussova eliminace ($Gauss$).
- Rohnova metoda ($Rohn$).
- Metoda nejmenších čtverců (Lsq).
- Metoda podčtverců ($Subsq$).
- Metoda Intlabu verifylss ($Verifylss$).

10.4.1 Rychlost algoritmů

První, co nás bude zajímat je doba běhu jednotlivých algoritmů. Metoda $GSpre$ používá jako zastavovací vektor $\epsilon = 0.001$ a odhad řešení získaný pomocí středové soustavy. Čas počítáme i s odhadem řešení. Metoda podčtverců bude náhodně vybírat m podčtverců pro soustavy o rozměrech $m \times n$. Gaussovu eliminaci budeme užívat s předpodmíněním. Rohnovu metodu budeme užívat s iteračním hledáním d a s ukončujícím vektorem rovněž $\epsilon = 0.001$. Pro řešení metody nadčtverce (Lsq) budeme používat Krawczykovu iterační metodu (odkaz na ni je možné najít v kapitole o převedení na čtvercový případ). Průměrné časy ukazuje tabulka 10.18. Testovány jsou na soustavách s poloměry intervalů $\leq 10^{-4}$. Výrazně nejhůř dopadla Gaussova eliminace. Naproti tomu nejlépe dopadla Rohnova metoda a za

matice	čas(sekundy)
5×3	0.1
9×5	0.6
11×6	2.15
13×7	7.67
15×13	0.62
15×7	28.16
23×19	42.43
40×39	0.30
40×38	4.47

Tabulka 10.13: Metoda podčtverců – časy pokud vyjadřujeme všechny podčtverce

	$r \leq 10^{-2}$			$r \leq 10^{-4}$		
systém	10	50	100	10	50	100
5×3	0.8263	0.8208	0.8208	0.8685	0.8623	0.8623
15×10	1.1910	1.0088	0.9474	1.1804	0.9681	0.9331
25×21	1.2950	1.0267	1.0061	1.2745	1.0337	1.0070
35×23	1.8898	1.2903	1.2079	1.8181	1.3010	1.1928
50×35	2.1116	1.4941	1.3975	2.0147	1.4294	1.3407
73×55	2.2965	1.7698	1.5979	2.0864	1.6126	1.4845
100×87	2.2878	1.6822	1.5669	1.9955	1.5266	1.4084

Tabulka 10.14: Metoda podčtverců – zlepšení obálky v závislosti na počtu prohledaných podsystémů

	$r \leq 10^{-1}$	$r \leq 10^{-2}$	$r \leq 10^{-3}$	$r \leq 10^{-4}$
5×3	2.15	2.06	2.08	2.11
15×10	2.3	2.03	2	2
25×21	3.02	2.08	2.02	2
35×23	3.09	2.06	2	2
50×35	6.93	2.1	2.01	2
73×55	$\times$	2.2	2	2
100×87	$\times$	2.3	2	2
200×170	$\times$	5.3	2.1	2

Tabulka 10.15: Metoda podčtverců – průměrné počty kroků potřebné k detekci neřešitelnosti

matice	čas
5×3	6 sec
9×5	43 sec
13×7	5 min
15×9	28 min

Tabulka 10.16: Lineární programování – časy při prohledání všech ortantů

ní v těsném závěsu metoda `verifylss`. Pokud ještě zmenšíme poloměry intervalů (na $\leq 10^{-5}$) nebo pokud zmenšíme ϵ na 10^{-5} pořadí rychlostí zůstává zachováno.

10.4.2 Přesnost algoritmů

Šířky obálek budeme testovat nadvakrát. Pro malé systémy budeme porovnávat s optimálním obalem získaným pomocí lineárního programování. Pro systém s poloměry $\leq 10^{-4}$ a $\epsilon = 0.001$ dostáváme tabulku 10.19. Je vidět, že Rohnova metoda nedává moc dobré výsledky, to je způsobeno velkým ϵ vzhledem k poloměru koeficientů systému. Pokud položíme $\epsilon = 10^{-5}$ dostaneme následující výsledky v tabulce 10.20. V obou dvou případech používáme metodu podčtverců s prohledáním všech podsystémů. Z tabulky vidíme, že pokud prohledáme všechny podsystémy dostaneme obal velmi blízký ideální obálce. Metody `verifylss` a *Lsq* dávají stejnou obálku, což je způsobeno tím, že i metoda `verifylss` využívá metodu nejmenších čtverců. Porovnáním časů zjistíme, že `verifylss` je implementována efektivněji. Překvapivým zjištěním je, že metoda *GSpre* a *Gauss* dávají stejnou šíři obálky. Přičítáme to tomu, že u obou metod využíváme stejné předpodmínění.

Pro malé systémy dávají lineární programování a metoda podčtverců téměř stejné obálky. Podíváme se na porovnání časů. Protože kdyby metoda podčtverců byla rychlejší než lineární programování, bylo by vhodné ji v konkrétních případech použít. Průměrné časy ve vteřinách pro různé poloměry ukazuje tabulka 10.21. Je vidět, že pro malé systémy nebo pokud jsou si maticové rozměry blízké je rychlejší metoda podčtverců. Pro větší poloměry je lineární programování výrazně pomalejší, což přičítáme vnitřní implementaci `verlinprog`.

Pro stejné systémy a $\epsilon = 10^{-5}$ budeme velké rozměry matic z časových důvodů porovnávat s metodou `verifylss`. Při metodě podčtverců vybíráme pro systémy velikosti $m \times n$ již jen m podsystémů. Dostáváme tabulku 10.22. Nejlépe vychází *Rohn* a *Lsq*, které jsou srovnatelné s `verifylss`.

10.5 Neřešitelnost systému

To, že systém nemá řešení můžeme poznat několika způsoby. Mějme intervalový lineární systém $\boldsymbol{A}x = \boldsymbol{b}$, kde matice $\boldsymbol{A}$ má rozměry $m \times n$. Pokud má pro každý podsystém $Ax = b$ matice $[\, A \,|\, b \,]$ hodnost rovnou $n+1$, pak každá matice A musí mít hodnost n a podle Frobeniovy věty systém $Ax = b$ nemá řešení. To, že má intervalová matice hodnost rovnou počtu svých sloupců zjistíme podle následující postačující podmínky. Odkaz je možno najít v [9].

Postačující podmínka (hodnost rovna počtu sloupců). *Mějme intervalovou matici $\boldsymbol{A} = [A^c - A^\Delta, A^c + A^\Delta]$ o rozměrech $m \times n$. Pak platí, že všechny matice $A \in \boldsymbol{A}$ mají hodnost rovnou n, pokud A^c má hodnost n a zároveň*

$$\varrho(|(A^c)^+|A^\Delta) < 1,$$

kde $(A^c)^+$ je Moore-Penroseova inverze A^c.

Jelikož má A^c lineárně nezávislé sloupce díky hodnosti n, můžeme spočítat

$$(A^c)^+ = \big((A^c)^T(A^c)\big)^{-1}(A^c)^T.$$

matice	čas
5×3	1 sec
15×10	3.5 sec
25×21	13 sec
35×23	19 sec
45×31	43 sec
55×35	1 min
73×55	9 min

Tabulka 10.17: Lineární programování – časy při využití znaménkového vektoru

matice	*GSpre*	*Gauss*	*Rohn*	*Lsq*	*Subsq*	*Verifylss*
5×3	0.022	0.0239	0.00096	0.0124	0.0217	0.0046
15×13	0.0731	0.1365	0.0018	0.0144	0.0666	0.005
35×23	0.1257	0.5657	0.0026	0.0226	0.1625	0.0065
50×35	0.1927	1.1618	0.0048	0.0373	0.2513	0.0103
100×87	0.4961	4.9578	0.034	0.1894	1.0431	0.0386
200×170	1.3616	19.7027	0.1678	1.2556	6.3006	0.2428

Tabulka 10.18: Průměrné časy všech metod ve vteřinách

matice	*GSpre*	*Gauss*	*Rohn*	*Lsq*	*Subsq*	*Verifylss*
5×3	1.9054	1.9054	20.0494	1.1932	1	1.1932
9×5	4.1892	4.1892	11.4236	1.1780	1	1.1780
13×7	3.5314	3.5214	9.5117	1.2021	1	1.2021
15×9	5.4511	5.4511	6.9811	1.1543	1.0001	1.1543

Tabulka 10.19: Poměry šířek obálek všech metod – malé systémy ($\epsilon = 0.001$)

matice	*GSpre*	*Gauss*	*Rohn*	*Lsq*	*Subsq*	*Verifylss*
5×3	9.3552	9.3555	1.6560	1.3740	1	1.3740
9×5	6.5017	6.5020	1.3201	1.2047	1	1.2047
13×7	19.1848	19.1955	1.2982	1.2039	1	1.2039
15×9	14.7869	14.7907	1.2161	1.1607	1.0001	1.1607

Tabulka 10.20: Poměry šířek obálek všech metod – malé systémy ($\epsilon = 10^{-5}$)

	$r \leq 10^{-1}$		$r \leq 10^{-4}$	
systém	*Linprog*	*Subsq*	*Linprog*	*Subsq*
5×3	1.078	0.055	0.88	0.056
9×5	1.67	0.61	1.6367	0.55
13×7	3.52	9.01	2.59	7.39
15×13	20.79	0.72	6.79	0.48

Tabulka 10.21: Srovnání časů lineárního prog. a metody podčverců pro malé systémy

Systém $\boldsymbol{A}x = \boldsymbol{b}$, kde matice $\boldsymbol{A}$ má rozměry $m \times n$ je neřešitelný, pokud intervalová matice $[\boldsymbol{A}|\boldsymbol{b}]$ má hodnost rovnou $n + 1$. Pokud však podmínka neplatí, nejsme schopni o neřešitelnosti nic říci.

Neřešitelnost systému můžeme poznat i některými algoritmy, které jsme si dosud představili. Předně je to lineární programování. Pokud nenalezne řešení v žádném ortantu, systém nemá řešení. Dále pro malé systémy jsme schopni neřešitelnost poznat Gaussovou eliminací bez předpodmínění. Pro systémy s poloměry koeficientů menšími jak 10^{-2} jsme schopni neřešitelnost spolehlivě poznat metodou podčtverců.

10.6 Ideální algoritmus

Záleží k jakému systému počítáme obálku množiny řešení. Pokud máme malý systém, vyplatí se využít metodu podčtverců a spočítat obálku velmi blízkou obalu. Tato metoda neklade žádné podmínky na poloměry intervalů systému. Navíc máme možnost detekovat případnou neřešitelnost systému. Pro větší systémy též bez omezení můžeme využít metodu nejmenších čtverců. Pokud se však zřekneme detekce neřešitelnosti. Srovnatelné výsledky dává Rohnova metoda, která je ještě o něco rychlejší. Je zde však problém, že pro poloměry blízké 0.1 se občas nepodaří nalézt hledaný vektor d. Iterační metody dávají velmi dobré výsledky pro čtvercové systémy, pro přeurčené se však příliš nehodí. Stejně tak Gaussova eliminace dává pro velké systémy příliš široké obálky. Metodou podčtverců také můžeme určit počáteční obálku systému. Stačí vybrat náhodně několik málo čtvercových podsystémů a jejich průnik vezmeme za odhad obálky.

Výsledky, které algoritmy se hodí pro konkrétní případy, shrnujeme pro přehlednost v následující tabulce. Tabulka ukazuje shrnutí poznatků z testování pro přeurčené lineární systémy do řádu zhruba 1000. V prvním sloupci jsou uvedeny vlastnosti systému. Slovo „libovolný“ znamená, že systém je libovolné velikosti. Druhý sloupeček zobrazuje případné další požadavky. Pokud je někde uvedeno „+ čas“ znamená to, že nás zajímá i doba výpočtu. V pravém sloupci jsou uvedeny metody, které se pro danou kombinaci systému a našich požadavků hodí.

systém $m \times n$	co chceme	metoda
libovolný	obal	*Linprog*
$\max(m,n) < 20$, blízké m,n	téměř obal + čas	*Subsq*
$\max(m,n) < 100, m - n < 3$	co nejužší obálka	*Subsq*
libovolný, $m - n \geq 3$	co nejužší obálka	$Verifylss$
libovolný, $r \leq 10^{-2}$	co nejužší obálka + čas	*Rohn*
libovolný	detekce neřešitelnosti	*Linprog*
libovolný, $r \leq 10^{-1}$	detekce neřešitelnosti + čas	*Subsq*

matice	*GSpre*	*Rohn*	*Lsq*	*Subsq*	*Gauss*
35×23	11.1655	1.0177	1.000	1.4045	11.1664
50×35	11.8722	1.0108	1.000	1.5445	11.8736
100×87	13.513	1.0014	0.9999	1.4222	13.515
200×170	16.3620	0.999	0.9998	1.8641	16.3639

Tabulka 10.22: Poměry šířek obálek všech metod – velké systémy

11. Intlab

11.1 Úvod

Veškeré algoritmy vytvořené a použité v této práci jsou napsány v jazyce Matlab. Intervalové počítání je řešeno s pomocí toolboxu Intlab. Tato kapitola je věnována právě Intlabu, jednak z důvodu snadného pochopení metod použitých v algoritmech a jednak kvůli tomu, že v českém jazyce (pokud je nám známo) neexistuje žádný základní úvod k Intlabu. Následující text může v budoucnu sloužit jako základní úvodní materiál. Zde se zabýváme především částí Intval, která obstarává základní funkčnost intervalového počítání. Detailnější popis této i ostatních částí lze nalézt v [6], [14] nebo přímo v demo materiálech Intlabu. V této kapitole předpokládáme pouze základní znalost jazyka Matlab.

11.2 Popis Intlabu

Intlab je volně dostupný soubor nástrojů pro Matlab. Roku 1998 ho vytvořil profesor Siegfried M. Rump. Hlavním cílem jsou rychlé výpočty a jednoduché použití při vytváření kódu. Další snahou je, aby získané výsledky byly verifikované, tedy abychom měli jistotu, že hledané řešení se v našem výsledku opravdu nachází. Je založen na systému BLAS[1]. Předpokladem je, že počítačová aritmetika splňuje standard IEEE-754. Systém je kompletně napsaný v jazyce Matlab s výjimkou jedné procedury `setround(x)`[2] sloužící pro nastavování směru zaokrouhlení. Aktuální nastavení směru zaokrouhlování lze zjistit příkazem

```
getround()
```

Standard IEEE-754 a funkce `setround` jsou jediným limitem přenositelnosti mezi různými stroji a operačními systémy.

11.3 Součásti Intlabu

Intval - Základní část, definující typ interval. Definuje operace a funkce na intervalech. Převážně touto částí se v této kapitole budeme zabývat.

Long - Umožňuje práci s čísly s nastavitelně velkou přesností.

Polynom - Toolbox pro práci s polynomy. Umožňuje generovat náhodné polynomy, vyčíslovat jejich hodnoty, hledat jejich kořeny. Umožňuje i práci s intervalovými polynomy. Obsahuje předdefinované typy některých známých polynomů.

Gradient, Hessian, Slope, Taylor - Toolboxy pro automatickou diferenciaci různých řádů a Taylorovy řady.

[1] Basic Linear Algebra Subroutines.

[2] parametr x: -1 k nejbližšímu dolnímu, 0 k nejbližšímu, 1 k nebližšímu hornímu.

11.4 Zapojení Intlabu

Pro zprovoznění Intlabu je nutné učinit dva kroky. Zaprvé je potřeba v kořenovém adresáři Intlabu změnit v souboru *startup.m* řádek

```
cd 'c:\intlab'
```

kde je nutné do uvozovek napsat plnou cestu adresáře, kde je Intlab uložen. Dále je nutné přidat do seznamu cest Matlabu adresář, kde je uložena knihovna Intlab. To provedeme v hlavním menu posloupností akcí

```
File->Set Path->Add Folder
```

Nevadí pokud k souboru, kde jsou uloženy cesty Matlabu nemáme přístupová práva. Matlab se nás automaticky zeptá, zda má vytvořit lokální kopii tohoto souboru v našem domovském adresáři. Tím je zaručeno, že při každém spuštění Matlabu dojde k inicializaci knihovny Intlab a načtení všech modulů potřebných pro její běh. Pokud nechceme, aby se Intlab spouštěl při každém spuštění Matlabu, neprovedeme výše uvedené změny. Je možné ho jednorázově inicializovat příkazem `startintlab` v příkazové řádce Matlabu. Předtím je nutné změnit adresářovou cestu na adresář, kde je uložen Intlab, nebo tento adresář mít v cestách Matlabu.

11.5 Intval

11.5.1 Definice intervalu

Intlab z výpočetních důvodů uchovává reálné intervaly ve tvaru infimum, supremum. A komplexní intervaly ve tvaru střed, radius. Pokud krajní meze intervalů nejsou reprezentovatelné, Intlab provede zaokrouhlení (dolní mez na nejbližší dolní reprezentovatelné číslo, horní mez na nejbližší horní reprezentovatelné číslo). Ke změně zaokrouhlování se použije právě výše zmíněná funkce `setround`. Intlab umožňuje každý interval definovat třemi způsoby:

1. `x = intval(7)`
2. `x = infsup(2.3+3i,2.5-0.2i)`
3. `x = midrad(2.5,0.05)`

Případ 1. slouží pro zadání intervalové hodnoty. Řádek 2. udává interval pomocí infima a suprema. A pomocí 3. zadáme reálný interval pomocí středu a poloměru. U prvního případu je však nutné dávat pozor. Ideální případ je, že dané číslo 7 bude obaleno malým intervalem. V případě reprezentovatelnosti čísla tento interval bude nulového poloměru, zde $[7, 7]$. U čísel, které však nejsou strojově reprezentovatelná se může stát, že v takto malém intervalu se původně zadané číslo nebude nacházet. Je proto lepší chápat příkaz `x = intval(7)` jako konverzi čísla typu `double` na typ `interval`. Původního záměru dosáhneme použitím příkazu

```
x = intval('7')
```

tímto příkazem je provedena konverze řetězce na interval, tentokrát se zachováním a správnou konverzí mezí. Pokud do řetězce dosadíme více intervalových hodnot, Intlab ho převede na vektor intervalů. Tento vektor můžeme přetvarovat do matice $m \times n$ použitím funkce `reshape(x, m, n)`. Všechny tyto příkazy pracují také s vektory a maticemi. Příslušné matice si musí rozměrově odpovídat. U midrad je možné navíc zadat pro všechny prvky matice jednotný poloměr.

```
>> i = intval([1.1 2.3; 3.1 4.3])
intval i =
[    1.1000,    1.1001] [    2.2999,    2.3000]
[    3.1000,    3.1001] [    4.2999,    4.3000]
>> i = infsup( [1 2; 3 4], [ 1.1 2.3; 3.1 4.3 ] )
intval i =
[    1.0000,    1.1001] [    2.0000,    2.3000]
[    3.0000,    3.1001] [    4.0000,    4.3000]
>> i = midrad( [1 2; 3 4], [ 0.1 0.3; 0.1 0.3 ] )
intval i =
[    0.8999,    1.1001] [    1.6999,    2.3001]
[    2.8999,    3.1001] [    3.6999,    4.3001]
>> i = midrad( [1 2; 3 4],  0.1 )
intval i =
[    0.8999,    1.1001] [    1.8999,    2.1001]
[    2.8999,    3.1001] [    3.8999,    4.1001]
```

Mezi jednotlivými formáty zobrazení lze přepínat příkazy

```
format midrad, format infsup, format _
```

```
i = midrad( 1, 0.01);
>> format infsup
>> i
intval i =
[    0.9899,    1.0101]
>> format midrad
>> i
intval i =
<    1.0000,   0.0101>
>> format _
>> i
intval i =
    1.00__
```

To, jak se intervaly budou standardně zobrazovat, lze nastavit příkazy

```
intvalinit('displaymidrad')
intvalinit('displayinfsup')
intvalinit('display_')
```

11.5.2 Základní intervalové funkce

Jak je u Matlabu zvykem, funkce pracují často jak pro jednotlivé intervaly, tak i pro vektory a matice. Pro vektory a matice jsou funkce aplikovány na jednotlivé koeficienty zvlášť. Funkce a operace pracují tak, jak jsme je definovali v kapitole o intervalové aritmetice, jejich definice tedy nebudeme znovu uvádět.

mid(a) - vrací střed zadaného intervalu

rad(a) - vrací poloměr zadaného intervalu

inf(a) - vrací dolní mez intervalu

sup(a) - vrací horní mez intervalu

mag(a) - vrací magnitudu intervalu

mig(a) - vrací mignitudu intervalu

11.5.3 Základní funkce a operace

Základní aritmetické operace `(+,-,*,/,^)` fungují i pro intervaly. Pozor je potřeba dát u maticového násobení, kdy Intlab standardně používá rychlé násobení (zapíná se příkazem `intvalinit('fastivmult')`), které může nadhodnotit ideální výsledek až o 1.5 poloměru. Pro důležité výpočty existuje přesnější varianta, která ale trvá déle (`intvalinit('sharpivmult')`). Intval definuje elementární verifikované funkce pro intervaly. Opět pracující s intervaly, vektory i maticemi intervalů. Obsahují:

- goniometrické funkce (sin, cos, tan, cot, sec, cosec)
- cyklometrické funkce (asin, acos, atan, acot)
- hyperbolické funkce (sinh, cosh, tanh, coth)
- hyperbolometrické funkce (asinh, acosh, atanh, acoth)
- logaritmy (log, log2, log10)
- exponenciály (exp)

11.5.4 Množinové operace

in(a,b) - vrací 1 pokud se prvek a nachází v intervalu b, a může být interval i číslo

in0(a, b) - vrací 1 pokud a leží ve vnitřku intervalu b

intersect(a,b) - spočítá průnik intervalů, pokud nemají společný průnik vrací NaN

emptyintersect(a,b) - pokud chceme testovat prázdnost průniku voláme tuto funkci, vrací 2 proměnné, první je logická proměnná, která nabývá hodnoty 1 při prázdném průniku, jinak 0, druhá je průnik (NaN při prázdném)

hull(a,b) - pro sjednocení intervalů a, b se nepoužívá funkce union, ale hull, která vytvoří obal intervalů a, b

eq(a,b) - vrací 1 pokud se a, b rovnají, jinak 0

ge(a,b) - implementuje relaci $\geq$, vrací 1 pokud platí, jinak 0

le(a,b) - implementuje relaci $\leq$, vrací 1 pokud platí, jinak 0

Intlab obsahuje další množství funkcí, které zde z prostorových důvodů neuvádíme. Jejich detailní popis je možné nalézt v demo materiálech Intlabu nebo přímo v m-souborech příslušejících jednotlivým funkcím. V každém z nich se nachází nejen detailní popis funkce, ale i seznam provedených změn v implementaci.

11.5.5 Maticové funkce

Jak již bylo řečeno vstupem funkcí mohou být ve většině případů i matice. Intlab dokonce podporuje operace pro řídké matice (sparse matrices). Důležitou operací je inverze intervalové matice. Inverzní matici k čtvercové A spočítáme pomocí `inv(A)`.

11.5.6 Systémy rovnic

Intlab obsahuje vestavěné funkce pro verifikovaný výpočet obálky řešení lineárních i nelineárních systémů. Pro řešení intervalových lineárních systémů slouží funkce

```
verifylss(A, b)
```

kde A je bodová nebo intervalová matice a b je bodový nebo intervalový vektor. V Matlabu je možné řešení rovnic zapsat zkráceně ve formě `A\b`. Jestliže je alespoň nějaká složka A nebo b interval, volá se funkce `verifylss`. Výše zmíněná funkce využívá pro čtvercové systémy upravené metody Krawczykova operátoru. Pokud tato metoda nenajde řešení po 7 krocích, přejde se na metodu Hansen-Bliek-Rohn-Ning-Kearfott. Ta je zhruba stejně přesná. Dává lepší výsledky pro extrémně špatně podmíněné systémy. Ale je o něco pomalejší. Je možné první část s Krawczykovým operátorem přeskočit a přejít rovnou na druhou část příkazem

```
intvalinit('LssSecondStage')
```

Pokud je matice A řídká matice, solver automaticky používá variantu pro řídké matice. Pro přeurčené systémy se využije převod na nadčtverec. Obecně b může být i matice, pak metoda vyřeší soustavu pro každý sloupec matice b a vrací matici vektorů řešení. Takto se počítá v Intlabu například inverzní intervalová matice k matici A, tedy jako `verifylss(A, speye(n))`. Pro řešení nelineárních systémů slouží funkce `verifynlss`.

11.5.7 Kreslení

Intlab obsahuje i nejrůznější funkce pro kreslení intervalových hodnot ve 2D a 3D.

plot(X,Y, c) - X, Y jsou reálné intervalové vektory, vykreslí jednotlivé boxy $[X_i, Y_i]$ vybarvené barvou c

plotintval(X) - vykreslí reálný nebo komplexní interval

plotlinsol(A,b) - vykreslí řešení reálného intervalového systému o 2 a 3 proměnných (tedy v 2D a 3D)

11.6 Aplikace intlabu

Při rozhodování, kterou intervalovou knihovnu použít padla volba na Intlab právě kvůli jeho aplikační šíři. Intlab nalezl praktické i teoretické uplatnění v mnoha oborech. Vybrali jsme jich několik tak, aby byla vidět veliká variabilita uplatnění. Velkou část referencí (přes 300) lze nalézt v seznamu profesora Rumpa na adrese

`www.ti3.tu-harburg.de/rump/intlab/INTLABref.pdf`

- Senzorická lokalizace a navigace robotů
- Evoluční design letadel
- Numerické metody ve vědeckém počítání
- Lineární optimalizační problémy s nepřesnými daty
- Intervalové metody pro řešení nelineárních rovnic
- Modelování nejistoty v populační biologii a ekologii
- Intervalové Monte Carlo metody
- Intervalová Fuzzy logika
- Kalibrace kamer a 3D rekonstrukce obrazu

11.7 Jiné knihovny

Existuje celá řada knihoven pro různé programovací jazyky. O hlavních z nich bychom se měli pro úplnost a srovnání zmínit.

11.7.1 Boost Interval Arithmetic Library

Boost je obsáhlá knihovna doplňků pro programovací jazyk C++. Obsahuje i třídy pro práci s intervaly. Třídy jsou definovány jako šablonové. Je tedy možné definovat vlastní intervalové typy nad existujícími datovými typy například float nebo double. Pro složitější datové typy (i uživatelem definové) umožňuje knihovna definovat zaokrouhlování a další podmínky pro práci s intervaly. Interval se zadává pomocí rozsahu mezí, nebo pro jeden parametr se provádí konverze na typ interval. Obsahuje základní aritmetické operátory `(+, -, *, /)`, algebraické funkce (sqrt, square, pow, abs), klasické funkce (exp, log, sin, cos, asin, acos), porovnávací operátory `(<, <=, >, >=, ==, !=)`. Dále intervalově specifické funkce (min, max, width, empty). Malou nevýhodou je, že použití knihovny závisí na architektuře, operačním systému a překladači. Může se stát, že pro jejich určitou kombinaci nemusí definovaná intervalová aritmetika generovat plausibilní výsledky.

11.7.2 Mathematica

Všechny základní funkce a operace programu Mathematica pracují i s objekty typu Interval (sin, exp, log, řešení rovnic). Intervaly je možné zadávat pomocí horní a dolní meze nebo jako sjednocení intervalů. Mathematica umožňuje aritmetické i množinové operace na intervalech. Čísla strojově nereprezentovatelná nebo čísla libovolné přesnosti lze převést na intervaly. Navíc intervaly můžeme dostat také jako výsledky různých operací například limit. Vše je doplněno o možnost intervaly přehledně vizualizovat.

11.7.3 XSC jazyky

XCS[3] jazyky představují rozšíření programovacích jazyků pro vědecké účely. Mimo standardních typů definují typy pro verifikované počítání (reálný i komplexní interval), aritmetické operátory i základní funkce či dynamické datové struktury. Součástí jsou knihovny umožňující provádět verifikované výpočty klasických problémů - řešení systémů lineárních a nelineárních rovnic, výpočet maticové inverze, vlastních čísel a vektorů, vyhodnocení aritmetických výrazů a polynomů, diferenciálních a integrálních rovnic a optimalizačních problémů. Dostupné jsou knihovny pro jazyky C++, Pascal a Fortran.

11.7.4 FI_LIB a FILIB++

FI_LIB[4] je knihovna napsaná v ANSI-C pro verifikované výpočty pomocí intervalových metod. Všechny výpočty jsou prováděny s přesností IEEE-double format. Rutiny nepoužívají změnu zaokrouhlovacích módů (jako například Intlab funkci `setround`). FILIB++ je rozšířením knihovny FI_LIB. Celá knihovna je napsaná formou šablonových tříd.

11.7.5 Versoft

Profesor Jiří Rohn vytvořil toolbox Versoft, který poskytuje verifikované řešení pro nejrůznější problémy numerické lineární algebry, které počítají s exaktními nebo intervalovými daty. Je napsaný v Matlabu za pomocí Intlabu. Obsahuje nejrůznější funkce pro matice, maticové dekompozice, lineární systémy a optimalizační problémy. Názvy funkcí (až na jednu výjimku) začínají na `VER`, pak následuje název většinou korespodující funkce Matlabu. Například
`VERRANK, VEREIG, VERDET` .

11.7.6 Ostatní

Zde jsme vybrali jen ty nejrozšířenější knihovny a software srovnatelné s Intlabem. Mezi další může patřit například software Maple též podporující intervalovou aritmetiku. Dále je nutno zmínit software COSY, který využívá intervalových metod pro globální optimalizaci a řešení obyčejných diferenciálních rovnic. Tento systém se používá pro predikci srážek Země s blízkými kosmickými objekty. Autory jsou

[3] Language e**X**tensions for **S**cientific **C**omputation.
[4] **F**ast **I**nteval **LIB**rary.

Martin Berz a Kyoko Makino, kteří v roce 2008 získali Mooreovu cenu. Existují i jiné intervalové balíky pro ostatní jazyky – Fortran, Java, Python.

11.8 Lime 1.0

Lime 1.0 je námi vytvořená knihovna pro Matlab. Využívá Intlab a částečně Versoft. Obsahuje veškeré algoritmy zmíněné v této práci. Obsahuje i některé algoritmy navíc, které z předchozích volně vyplynuly. Zdrojové kódy i s html dokumentací je možné nalézt na přiloženém CD. V dokumentaci jsou popsány pouze základní algoritmy bez pomocných metod. Pro popis ostatních algoritmů, nebo pro větší detaily je možné nahlédnout přímo do zdrojových kódů, které obsahují komentáře a poznámky k implementaci.

12. Závěr

V této práci jsme uvedli detailní přehled většiny metod použitelných pro nalezení nebo alespoň odhad intervalového obalu přeurčených systémů. Existují ještě další metody, kterými jsme se zde nezabývali. Například jsou to evoluční algoritmy nebo pravděpodobnostní metody. Jsou známy některé vědecké výsledky využívající tyto metody, avšak díky odlišným principům těchto metod, jsme je do naší práce nezařazovali.

V počátečních kapitolách jsme představili základy intervalové analýzy – intervalové aritmetiky, intervalové lineární algebry a intervalových lineárních systémů.

Tématem dalších kapitol byly jednotlivé metody. Předvedli jsme Gaussovu eliminaci, iterační metody, Rohnovu metodu, různé způsoby převedení přeurčeného systému na čtvercový případ a získání optimálního obalu s pomocí lineárního programování. Tato část byla přehledová, kde jsme shrnuli výsledky o jednotlivých způsobech řešení přeurčených intervalových lineárních systémů. Mimo to se nám podařilo získat i několik vlastních výsledků. Například převedení intervalových metod i pro přeurčené systémy, alternativní způsoby počítání některých součástí metod (Rohnova metoda, Iterační metody), či vlastní metoda podčverců, která pro malé systémy počítá obálku velmi blízkou obalu a navíc umožňuje detekci neřešitelnosti systému.

Všechny metody, o nichž jsme se v práci zmínili, byly implementovány v jednom toolboxu pro systém Matlab a jsou obsaženy na přiloženém CD. Toolbox obsahuje převážně metody pro řešení přeurčených intervalových systémů, avšak jsou v něm navíc obsaženy i některé metody intervalové lineární algebry, které z náplně práce přirozeně vyplynuly. V budoucnu by bylo vhodné balík rozšířit i o další metody a poskytnout tak ucelený nástroj pro intervalovou lineární algebru, případně zahrnout i řešení intervalových nelineárních systémů.

Ke konci práce jsme jednotlivé metody porovnali, jak z hlediska délky běhu, tak z hlediska přesnosti získaných obálek. Pokud algoritmus obsahoval některé kroky, které bylo možno řešit různými způsoby, porovnali jsme je taktéž. Testovali jsme zvlášť pro systémy řešitelné i neřešitelné. U řešitelných nás zajímala co nejužší obálka množiny řešení. U neřešitelných to, zda je metoda schopna neřešitelnost nějakým způsobem rozpoznat. Na konci porovnání jsme rozebrali případy, kdy je vhodné použít konkrétní metody.

Samostatnou kapitolu tvoří základní úvod do systému Intlab. Při pátrání se nám nepodařilo nalézt úvod v českém jazyce. Tato kapitola slouží pro snadnější orientaci ve zdrojových kódech. A také může být později využita jako výuková pomůcka pro případné zájemce o úvod do Intlabu na Matematicko-fyzikální fakultě.

Přeurčené systémy tvoří ucelenou a důležitou část intervalové lineární algebry. Tuto práci by bylo vhodné postupem času rozšířit směrem k řešení intervalových systémů obecně, ať lineárních, nebo i nelineárních. Možná by mohlo být zajímavé porovnat metody obsažené v práci i s evolučními algoritmy a pravděpodobnostními metodami. Také by bylo vhodné nalézt i jiné způsoby předpodmínění, které by umožňovaly širší použití iteračních metod.

Literatura

[1] M. Fiedler, J. Nedoma, J. Ramik, J. Rohn, and K. Zimmermann. *Linear optimization problems with inexact data*. Springer, 2006.

[2] E.R. Hansen and G. W. Walster. *Global optimization using interval analysis*. Marcel Dekker, New York, second edition, 2004.

[3] E.R. Hansen and G.W. Walster. Solving overdetermined systems of interval linear equations. *Reliable computing*, 12(3):239–243, 2006.

[4] G. Heindl, V. Kreinovich, and A.V. Lakeyev. Solving linear interval systems is np-hard even if we exclude overflow and underflow. *Reliable Computing*, 4(4):383–388, 1998.

[5] R.B. Kearfott. Interval computations: Introduction, uses, and resources. *Euromath Bulletin*, 2(1):95–112, 1996.

[6] R.E. Moore, R.B. Kearfott, and M.J. Cloud. *Introduction to interval analysis*. Society for Industrial Mathematics, 2009.

[7] A. Neumaier. *Interval methods for systems of equations*, volume 37. Cambridge University Press, 1990.

[8] G. Rex and J. Rohn. Sufficient conditions for regularity and singularity of interval matrices. *SIAM Journal on Matrix Analysis and Applications*, 20(2):437–445, 1998.

[9] J. Rohn. A handbook of results on interval linear problems. *Internet text available at http://www. cs. cas. cz/rohn/handbook*.

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

[11] J. Rohn. *Lineární algebra a optimalizace*. Karolinum, 2004.

[12] J. Rohn. Forty necessary and sufficient conditions for regularity of interval matrices: A survey. *Electronic Journal of Linear Algebra*, 18(500-512):2, 2009.

[13] J. Rohn. VERSOFT: Verification software in MATLAB / INTLAB, version 10, 2009. `http://uivtx.cs.cas.cz/~rohn/matlab/`.

[14] S.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/`.

Seznam tabulek