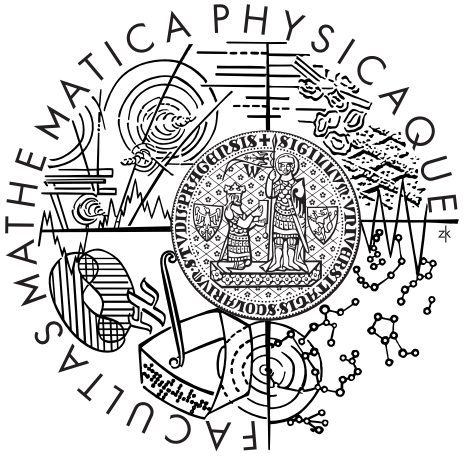
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Bořek Šťastný

Extrémy množiny řešení intervalových lineárních rovnic

Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Milan Hladík, Ph.D.
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2014

Rád bych poděkoval vedoucímu mé práce za velkou ochotu při konzultacích, za cenné rady a podněty. Své rodině a přátelům děkuji za podporu, kterou mi projevovali během celého studia.

Prohlašuji, že jsem tuto bakalářskou 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: Extrémy množiny řešení intervalových lineárních rovnic

Autor: Bořek Šťastný

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: Mgr. Milan Hladík, Ph.D., Katedra aplikované matematiky

Abstrakt: Tato práce se zabývá řešením intervalových soustav rovnic. Popsána je struktura množiny řešení, ze které vyplývá návrh některých algoritmů pro výpočet intervalového obalu množiny řešení. Výpočet intervalového obalu je obecně NP-těžká úloha, přesto existují algoritmy, které často skončí dříve než po exponenciálně mnoha krocích. Jedním z nich je Janssonův algoritmus, který jsme implementovali v prostředí MATLAB za použití intervalové knihovny INTLAB. Metodu jsme optimalizovali a porovnali s existujícími implementacemi. Ukázalo se, že je naše metoda ve srovnání rychlejší pro úlohy, jejichž množina proniká velké množství ortantů. Pokud byl počet navštívených ortantů při výpočtu malý, byla naše implementace v porovnání méně efektivní. Nastíněna je verifikovaná metoda lineárního programování pro dosažení rigorózních výsledků.

Klíčová slova: intervalové soustavy rovnic, intervalový obal, Janssonův algoritmus, MATLAB, INTLAB

Title: Extrema of the solution set of an interval linear system of equations

Author: Bořek Šťastný

Department: Department of Applied Mathematics

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

Abstract: Main topic of this thesis is solving interval linear systems. At first, we describe the structure of the solution set, which is the basis of several algorithms for computing interval hull of the solution set. Although computation of the interval hull is NP-hard problem, there exist algorithms which are not apriori exponential. One such algorithm is Jansson's algorithm which we implemented in MATLAB with utilisation of the interval toolbox INTLAB. We optimised the method and compared it to related implementations. Test results show that our implementation performs better in comparison on interval systems with solution set that is intersecting with many orthants. The opossite holds true when the amount of visited orthants is low. We describe a method of verified linear programming, which is necessary for producing rigorous results.

Keywords: interval linear systems, interval hull, Jansson's algorithm, MATLAB, INTLAB

Obsah

1 Úvod **3**
- 1.1 Cíle práce . . . 4
- 1.2 Související práce . . . 4
- 1.3 Struktura práce . . . 5

2 Základní pojmy **6**
- 2.1 Interval a intervalová aritmetika . . . 6
- 2.2 Intervalové vektory a matice . . . 6
- 2.3 Intervalové lineární soustavy rovnic . . . 7

3 Vlastnosti množiny řešení **8**
- 3.1 Věta Oettliho a Pragera a její důsledky . . . 8
- 3.2 Topologické vlastnosti . . . 10

4 Algoritmy intervalových soustav lineárních rovnic **11**
- 4.1 Využití lineárního programování pro výpočet mezí . . . 11
- 4.2 Algoritmus prohledávání ortantů . . . 11
  - 4.2.1 Popis algoritmu . . . 12
  - 4.2.2 Analýza algoritmu . . . 12
- 4.3 Janssonův algoritmus . . . 13
  - 4.3.1 Popis algoritmu . . . 14
  - 4.3.2 Analýza algoritmu . . . 14
- 4.4 Využití Janssonova algoritmu pro test regularity intervalové matice 15

5 Verifikace v lineárním programování **16**
- 5.1 Základní pojmy a značení lineárního programování . . . 16
- 5.2 Verifikace optimality báze . . . 17
- 5.3 Verifikace pomocí duální úlohy . . . 18

6 Optimalizace Janssonova algoritmu **20**
- 6.1 Odhad počátečního přípustného řešení . . . 20
- 6.2 Vynechání programů na základě mezí . . . 20
- 6.3 Ořezávání množiny přípustných řešení LP . . . 21

7 Testování implementace **22**
- 7.1 Postup testování . . . 22
- 7.2 Výpočet intervalového obalu . . . 23

8 Uživatelská dokumentace **25**
- 8.1 Instalace softwaru . . . 25
  - 8.1.1 Instalace INTLABu . . . 25
  - 8.1.2 Instalace řešiče . . . 25
  - 8.1.3 Ruční instalace MEX funkcí . . . 25
- 8.2 Reprezentace intervalových soustav rovnic . . . 26
- 8.3 Volání řešiče a interpretace výstupu . . . 27
- 8.4 Alternativní implementace . . . 29

9 Programátorská dokumentace **30**
9.1 Implementace Janssonova algoritmu 30
9.2 Datová struktura signatur ortantů 31
9.2.1 Struktura MEX souboru 31
9.2.2 Definované struktury 32
9.3 Verifikované lineární programování 32

10 Závěr **33**

Seznam použité literatury **34**

Seznam použitých zkratek **36**

Seznam tabulek **37**

Seznam obrázků **38**

1. Úvod

Původ slova interval je v latinském slově intervallum, s nímž se poprvé setkáváme okolo roku 1300. Původně šlo o termín označující prostor mezi dvěma palisádami nebo hradbami (inter = mezi, vallum = palisáda). Později se pojem rozšířil do různých oblastí a v současnosti se používá prakticky pouze v přeneseném významu. My se dále budeme zabývat výlučně jeho významem v oblasti matematiky, kde tvoří mimo jiné základ intervalové počtu. Hlavní myšlenkou intervalového počtu je nahrazení přesných hodnot intervaly a zavedení speciální intervalové aritmetiky. Motivace pro tento zvláštní postup vyvstává v mnoha vědních oborech, uvažme tento jednoduchý příklad z fyziky.

Potřebujeme stanovit rychlost pohybu nějakého tělesa. Měření dráhy a času, dvou veličin, z nichž pak rychlost spočítáme, však podléhá omezením daným nedokonalou technikou. Dráhu změříme pomocí metru s přesností např. na centimetr. K měření času použijeme stopky s přesností na setinu sekundy. Pokud chceme znát rychlost s absolutní jistotou, nespokojíme se s prostým dosazením průměrných naměřených hodnot do vzorečku:

$$v = s/t.$$

Stanovíme intervaly, v nichž dráha a čas musí ležet, v případě, že počítáme s možnou chybou měření. Pro dolní hranici intervalu z naměřeného výsledku možnou odchylku odečteme a pro horní hranici ji naopak přičteme:

$$s_0 - \Delta s \leq s \leq s_0 + \Delta s,$$

$$t_0 - \Delta t \leq t \leq t_0 + \Delta t.$$

Tak získáme intervaly, v nichž se nachází skutečné hodnoty dráhy a času. S nimi potom dopočítáme interval $v = [v_l, v_u]$,

$$v_l = (s_0 - \Delta s)/(t_0 + \Delta t),$$
$$v_u = (s_0 + \Delta s)/(t_0 - \Delta t),$$

o kterém již můžeme s jistotou tvrdit, že se v něm skutečná hodnota rychlosti nalézá.

Podobný problém vyvstává při reprezentaci čísel na počítači, která reprezentujeme pouze s konečnou přesností, což vede k zaokrouhlovacím chybám. Jednotlivě nemusí mít tyto chyby na výpočet velký vliv, jejich kumulací během výpočtu však můžeme dostat výsledek velmi vzdálený od toho skutečného. Zapouzdřením nepřesných čísel do intervalů sice také nedostaneme přesnou skutečnou hodnotu, budeme však schopni určit výsledný interval, ve kterém se hledaná hodnota nalézá. To může být velmi důležité pro aplikace, jako je například strojové dokazování.

Během minulých třiceti let role intervalového počtu v numerické analýze kontinuálně rostla, protože nabízí řešení problémů, které je jinak obtížné řešit metodami klasickými. Mezi základní úlohu intervalového počtu patří řešení intervalových soustav rovnic. Intervalová soustava rovnic je definována jako množina lineárních systémů, kde koeficienty matice soustavy a vektoru pravých stran nabývají hodnot mezi určitou horní a dolní mezí.

Odpovídající množina řešení je definována jako množina řešení všech systémů z množiny intervalové soustavy. Ta má obecně nekonvexní a komplikovaný tvar. Často však potřebujeme s výsledkem soustavy rovnic dále počítat. Proto spíše než přesné řešení požadujeme po algoritmu na řešení soustavy rovnic výstup ve formě boxu, který řešení těsně zapouzdřuje. Takovému boxu říkáme intervalový obal. Výpočet intervalového obalu je obecně NP-těžká úloha (Rohn, 1989), (Kreinovich et al., 1998). Existují algoritmy řešení intervalových rovnic, které sice skončí v polynomiálním čase, výsledný box zahrnující množinu řešení však v obecném případě není těsný (Moore et al., 2009). Takovému boxu říkáme intervalová obálka. Pokud nám to velikost úlohy dovoluje, je vždy výhodnější počítat intervalový obal. Může se nám totiž stát, že určíme výsledný interval řešení příliš široký a možná skutečná hodnota sledované veličiny přesáhne kritickou hodnotu.

Algoritmům na řešení intervalových soustav rovnic se věnuje mnoho publikací, viz např. (Neumaier, 1990), (Fiedler et al., 2006). Zajímavým postupem pro získání intervalového obalu množiny řešení je Janssonův algoritmus. Ačkoliv řeší NP-těžkou úlohu, často skončí dříve než po exponenciálně mnoha krocích. Naše práce se bude zabývat implementací a optimalizací právě tohoto algoritmu.

1.1 Cíle práce

Cílem této práce je efektivní a rigorózní implementace Janssonova algoritmu pro nalezení intervalového obalu množiny řešení intervalových soustav rovnic. Program bude napsán jako metoda v prostředí MATLAB za využití intervalové knihovny INTLAB. Základní algoritmus bude vylepšen vhodnými postupy při výpočtu lineárních programů omezujících množinu řešení.

1.2 Související práce

Zde uvádíme některé související práce. V prostředí MATLABu jsou to zejména:

- INTLAB (Rump, 1999) - sada nástrojů pro práci s intervalovými daty, definuje intervalový datový typ a efektivní intervalovou aritmetiku, součástí knihovny je i verifikovaná metoda `verifylss` pro řešení intervalových soustav rovnic poskytující intervalovou obálku řešení;

- VERSOFT (Rohn, 2009) - knihovna funkcí pro verifikované řešení různých intervalových i neintervalových úloh, obsahuje funkci `verintervalhull`, což je jediná nám známá implementace algoritmu pro získání intervalového obalu pro MATLAB.

Mimo MATLAB vznikla řada efektivních řešičů intervalových soustav rovnic. Za všechny uvedeme alespoň (Kramer a Zimmer, 2009), což je řešič napsaný v jazyce C++ za pomoci knihovny pro verifikované výpočty C-XSC. Pokročilou funkcí této implementace je možnost distribuovaného výpočtu a s tím spojený velký výpočetní výkon.

1.3 Struktura práce

Nejprve v následující kapitole zavedeme potřebné pojmy a značení intervalového počtu. V kapitole 4 detailně rozebereme Janssonův algoritmus, jehož návrh vyplývá z vlastností množiny řešení intervalových rovnic uvedených v kapitole 3. Dále v kapitole 5 představíme použité metody verifikace lineárního programování, které zaručují správnost výsledků. V kapitole 6 popíšeme námi implementované optimalizace Janssonova algoritmu a jejich dopad na výpočet. Výslednou implementaci řešiče porovnáme s existujícími metodami v kapitole 7. Poslední kapitoly 8 a 9 slouží jako uživatelská a programátorská příručka. Popíšeme, jak řešič naistalovat do prostředí MATLABu a jak program používat. Vypíchneme důležité detaily implementovaných algoritmů a popíšeme použité datové struktury.

2. Základní pojmy

V této kapitole definujeme základní pojmy a použité značení intervalového počtu.

2.1 Interval a intervalová aritmetika

Definice (Reálný interval)**.** Nechť $\mathbb{R}$ značí těleso reálných čísel. Mějme $\overline{x}, \underline{x} \in \mathbb{R}$ a nechť platí $\underline{x} \leq \overline{x}$. Potom reálným intervalem myslíme množinu

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

Hodnoty $\underline{x}, \overline{x}$ nazýváme dolní, resp. horní mezí intervalu $\boldsymbol{x}$. Pokud platí $\underline{x} = \overline{x}$, potom se intervalu říká degenerovaný. Množinu reálných intervalů značíme $\mathbb{IR}$. Někdy může být užitečné vyjádření intervalu jako:

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

kde definujeme

$$x^c := \frac{1}{2}(\underline{x} + \overline{x}),$$

$$x^\Delta := \frac{1}{2}(\overline{x} - \underline{x})$$

a mluvíme o středu, resp. poloměru intervalu.

Definice (Intervalová aritmetika)**.** Nechť $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{IR}$. Intervalová operace $\circ$ je definovaná jako

$$\boldsymbol{x} \circ \boldsymbol{y} = \{x \circ y; x \in \boldsymbol{x}, y \in \boldsymbol{y}\}.$$

Pro základní aritmetické operace $+, -, *, /$ můžeme zápis vyjádřit:

- sčítání: $\boldsymbol{x} + \boldsymbol{y} = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$;
- odčítání: $\boldsymbol{x} - \boldsymbol{y} = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$;
- násobení: $\boldsymbol{x} * \boldsymbol{y} = [\min(S), \max(S)]$ kde $S = \{\underline{xy}, \underline{x}\overline{y}, \overline{x}\underline{y}, \overline{xy}\}$;
- dělení: $\boldsymbol{x}/\boldsymbol{y} = \boldsymbol{x} * \frac{1}{\boldsymbol{y}}$, pokud $0 \notin \boldsymbol{y}$ a $\frac{1}{\boldsymbol{y}} = [\frac{1}{\overline{y}}, \frac{1}{\underline{y}}]$.

2.2 Intervalové vektory a matice

Klasické reálné vektory značíme netučnými malými písmeny (např. $x, y, z \in \mathbb{R}^n$), reálné matice netučnými písmeny velkými (např $A, B, C \in \mathbb{R}^{m \times n}$). Pro odlišení budeme značit intervalové vektory a matice písmeny tučnými ($\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{IR}^n$, resp. $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C} \in \mathbb{IR}^{m \times n}$).

Definice (Intervalová matice)**.** Nechť $\underline{A}, \overline{A} \in \mathbb{R}^{m \times n}$ takové, že $\underline{A} \leq \overline{A}$. Potom množině:

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

říkáme reálná intervalová matice a maticím $\underline{A}, \overline{A}$ říkáme její horní a dolní mez. Množinu všech reálných intervalových matic $m \times n$ značíme $\mathbb{IR}^{m\times n}$. Středovou matici A^c a matici poloměru A^Δ definujeme jako:

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

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

Pomocí středové matice a matice poloměru můžeme intervalovou matici ekvivalentně zapsat jako:

$$\boldsymbol{A} := [A^c - A^\Delta, A^c + A^\Delta] = \{A; |A^c - A| \leq A^\Delta\},$$

kde relaci uspořádání $\leq$ definujeme po složkách.

Speciálním případem reálné neintervalové matice je jednotková matice, kterou značíme $I_n \in \mathbb{R}^n$. Intervalové vektory chápeme jako intervalové matice o rozměrech $1 \times n$ nebo $m \times 1$. Speciální připady klasických neintervalových vektorů, jež budeme využívat, jsou vektor samých jedniček a vektor samých nul:

$$\mathbb{1} \in \mathbb{R}^m,$$

$$0 \in \mathbb{R}^m.$$

Definice. Řekneme, že intervalová matice $\boldsymbol{A} \in \mathbb{IR}^{n\times n}$ je regulární, pokud jsou regulární všechny matice $A \in \boldsymbol{A}$.

Definice. Řekneme, že intervalová matice $\boldsymbol{A} \in \mathbb{IR}^{n\times n}$ je singulární, pokud je singulární alespoň jedna matice $A \in \boldsymbol{A}$.

2.3 Intervalové lineární soustavy rovnic

Definice (Intervalové rovnice)**.** Soustavu intervalových lineárních rovnic definujeme jako množinu:

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

a budeme ji značit jako:

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

Definice (Množina řešení)**.** Množinu řešení odpovídající soustavě $\boldsymbol{A}x = \boldsymbol{b}$ definujeme jako množinu:

$$\Sigma := \Sigma(\boldsymbol{A}, \boldsymbol{b}) = \{x; Ax = b, A \in \boldsymbol{A}, b \in \boldsymbol{b}\}.$$

Definice (Intervalový obal)**.** Intervalový obal množiny řešení Σ značíme $\diamond\Sigma = [\underline{\diamond\Sigma}, \overline{\diamond\Sigma}]$ a definujeme jako:

$$\underline{\diamond\Sigma}_i := \min_{x\in\Sigma} x_i,$$

$$\overline{\diamond\Sigma}_i := \max_{x\in\Sigma} x_i.$$

Definice (Intervalová obálka)**.** Vnější intervalová obálka množiny řešení Σ je intervalový vektor $\boldsymbol{x}$ splňující:

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

Vnitřní intervalová obálka množiny řešení Σ je intervalový vektor $\boldsymbol{y}$ splňující:

$$\boldsymbol{y} \subseteq \Sigma.$$

3. Vlastnosti množiny řešení

V této kapitole nastíníme důležité vlastnosti množiny řešení soustavy lineárních intervalových rovnic, ze kterých přímo vyplývá návrh algoritmů pro nalezení jejího intervalového obalu.

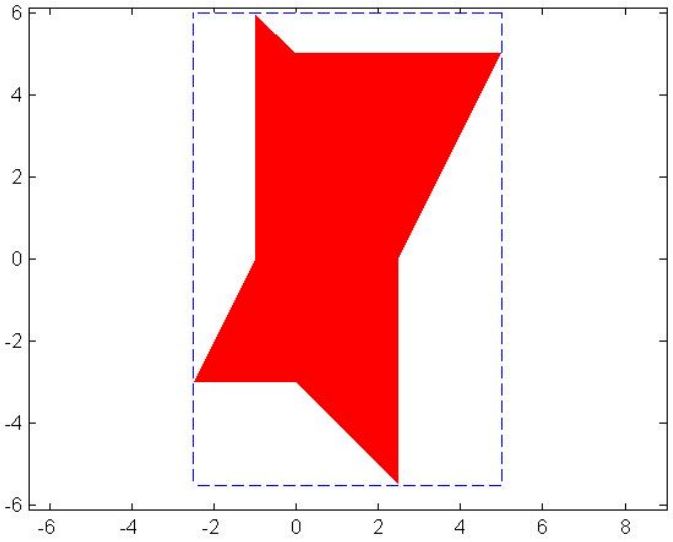
3.1 Věta Oettliho a Pragera a její důsledky

Uvažujme následující příklad:

Příklad 3.1.

$$\begin{pmatrix} [2,4] & [-1,0] \\ [0,1] & [1,2] \end{pmatrix} x = \begin{pmatrix} [-2,5] \\ [-3,5] \end{pmatrix}.$$

Množinu řešení je možné vidět na následujícím obrázku, přerušovaně je vyznačený intervalový obal.



Obrázek 3.1: Množina řešení příkladu 3.1

Následující věta Oettliho a Pragera popisuje množinu řešení a její důkaz je možné najít v (Oettli a Prager, 1964) nebo (Rohn, 1989). Věta nepředpokládá regularitu matice $\boldsymbol{A}$.

Věta 3.1 (Oettli-Prager)**.** *Buď $\boldsymbol{A} \in \mathbb{IR}^{m\times n}, \boldsymbol{b} \in \mathbb{IR}^m$. Množinu řešení $\Sigma(\boldsymbol{A}, \boldsymbol{b})$ lze vyjádřit:*

$$\Sigma := \{x \in \mathbb{R}^n; |A^c x - b^c| \leq A^\Delta |x| + b^\Delta\}.$$

Z věty plyne následující důležitý výsledek, který ukázal například (Rohn, 1989). Nejprve definujeme pojem ortantu.

Definice (Ortant)**.** Buď $s \in \{-1, +1\}^n$. Buď $D_s = diag(s)$ diagonální matice vektoru s. Ortant určený vektorem s je množina:

$$R^n(s) := \{x \in \mathbb{R}^n; D_s x \geq 0\}.$$

Vektoru s říkáme signatura ortantu $R^n(s)$.

Věta 3.2. *Průnik množiny řešení Σ s každým ortantem je konvexní polyedr.*

Důkaz. Popis konvexního polyedru v určeném ortantu získáme následovně. Buď x libovolné řešení z množiny Σ v daném ortantu. Buď $s = sgn(x)$ jeho znaménkový vektor a $D_s = diag(s)$. Pro všechna řešení $x \in R^n(s)$ platí:

$$|x| = D_s x$$

a z Věty 3.1 potom pro ně dostáváme:

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

Průnik daného ortantu a množiny řešení je tedy dán nerovnicemi:

$$(A^c - A^\Delta D_s)x \leq b^c + b^\Delta,$$

$$(A^c + A^\Delta D_s)x \geq b^c - b^\Delta,$$

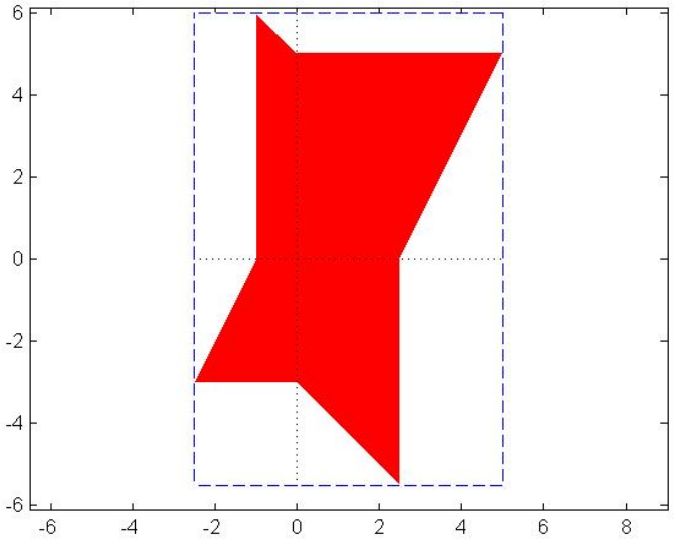
$$D_s x \geq 0.$$

□

Poznámka. Průnik množiny řešení Σ s ortantem $R^n(s)$ budeme značit $\Sigma(s)$.

Důsledek 3.3. *Množina řešení Σ je sjednocením nejvýše 2^n konvexních polyedrů $\Sigma(s)$.*

Pro ilustraci předchozí věty jsou na následujícím obrázku zobrazujícím množinu řešení z Příkladu 3.1 vyznačeny osy souřadnic.



Obrázek 3.2: Množina řešení příkladu 3.1 s vyznačenými osami souřadnic

3.2 Topologické vlastnosti

Věta 3.4. *Množina řešení Σ je uzavřená.*

Důkaz. Plyne z Věty 3.1. □

Věta 3.5. *Nechť je množina řešení Σ neprázdná. Potom platí právě jedno z následujících tvrzení:*

1. *Množina řešení Σ je omezená.*
2. *Každá souvislá komponenta Σ je neomezená.*

Důkaz. Viz (Jansson, 1997). □

Věta 3.6. *Nechť je množina řešení Σ neprázdná. Pokud je souvislá komponenta $\hat{\Sigma}$ množiny řešení Σ omezená, potom platí následující:*

1. *Σ je kompaktní.*
2. *$\boldsymbol{A}$ je regulární.*
3. *Σ je souvislá a $\hat{\Sigma} = \Sigma$.*

Důkaz. Viz (Jansson, 1997). □

Věta 3.7. *Nechť $\boldsymbol{A}$ obsahuje alespoň jednu regulární matici. Potom následují tvrzení jsou ekvivalentní:*

1. $\boldsymbol{A}$ je regulární.
2. Množina řešení Σ je omezená.

Důkaz. Viz (Rohn, 1994). □

4. Algoritmy intervalových soustav lineárních rovnic

Nyní představíme některé algoritmy pro řešení intervalových soustav lineárních rovnic. Nejdříve popíšeme základní algoritmus pro počítání intervalového obalu množiny řešení a poté jeho efektivnější verzi - Janssonův algoritmus.

4.1 Využití lineárního programování pro výpočet mezí

Následující věta ukazuje, jak spočítat přesné meze $\Sigma(s)$.

Věta 4.1. *Buď* $s \in \{-1, +1\}^n$

1. *Hodnota* $s_k . \min_{x \in \Sigma(s)} s_k x_k$ *je přesnou dolní, resp. horní mezí k-té složky* $(\Sigma(s))_k$ *v případě, že* $s_k = +1$*, resp.* $s_k = -1$*.*
2. *Hodnota* $s_k . \max_{x \in \Sigma(s)} s_k x_k$ *je přesnou horní, resp. dolní mezí k-té složky* $(\Sigma(s))_k$ *v případě, že* $s_k = +1$*, resp.* $s_k = -1$*.*
3. $\Sigma(s)$ *je neomezená právě tehdy, když existuje* $k \in \{1 \dots n\}$ *takové, že hodnota* $s_k . \max_{x \in \Sigma(s)} s_k x_k$ *je neomezená.*

Důkaz. Viz (Jansson, 1997). □

Pokud máme přesné meze $\Sigma(s)$, je výpočet intervalového obalu $\diamond\Sigma = [\underline{\diamond\Sigma}, \overline{\diamond\Sigma}]$ již přímočarý. Označme $h(s) = [\underline{h(s)}, \overline{h(s)}]$ dolní a horní meze $\Sigma(s)$. Máme:

$$\underline{\diamond\Sigma}_k = \min\{\underline{h(s)}_k, s \in \{-1, +1\}^n\}, \tag{4.1}$$

$$\overline{\diamond\Sigma}_k = \max\{\overline{h(s)}_k, s \in \{-1, +1\}^n\}. \tag{4.2}$$

4.2 Algoritmus prohledávání ortantů

Abychom mohli projít všechny ortanty a získat příslušné meze $h(s)$, budeme potřebovat efektivně generovat všechny prvky množiny $\{-1, 1\}^n$. Označme ji Y_n. Důkaz správnosti a konečnosti následujícího algoritmu lze nalézt v (Rohn, 2003).

Algorithm 1 Algoritmus generování Y_n

Ensure: $Y = Y_n$

1: $z := 0 \in \mathbb{R}^n$; $y \in Y_n$; $Y := \{y\}$;
2: **while** $z \neq \mathbb{1}$ **do**
3: $k := \min\{i; z_i = 0\}$;
4: **for** i := 1 ... k - 1 **do**
5: $z_i := 0$;
6: **end for**
7: $z_k := 1; y_k := -y_k$;
8: $Y = Y \cup y$;
9: **end while**
10: **return** Y

4.2.1 Popis algoritmu

Nyní již máme všechny prostředky pro algoritmus založený na prohledávání ortantů a omezení množiny řešení pomocí LP (lineárního programování).

Algorithm 2 Algoritmus prohledávání ortantů

Require: $\boldsymbol{A} \in \mathbb{IR}^{m \times n}, \boldsymbol{b} \in \mathbb{IR}^m$

1: vygeneruj Y_n pomocí Algoritmu 1
2: **for all** $s \in Y_n$ **do**
3: **for** k:=1 ... n **do**
4: vyřeš LP $\min_{x \in \Sigma(s)} s_k x_k$ a $\max_{x \in \Sigma(s)} s_k x_k$
5: **if** $\max_{x \in \Sigma(s)} s_k x_k$ je neomezené **then** konec, $\boldsymbol{A}$ je singulární
6: **end if**
7: ulož optima do příslušných složek $[\underline{h(s)}, \overline{h(s)}]$ podle Věty 4.1
8: **end for**
9: **end for**
10: ze vztahů (4.1) a (4.2) vypočítej intervalový obal $[\underline{\diamond\Sigma}, \overline{\diamond\Sigma}]$
11: **return** $\diamond\Sigma$

4.2.2 Analýza algoritmu

Konečnost algoritmu plyne z konečnosti algoritmu pro generování signatur ortantů a z toho, že v každém ortantu řešíme konečný počet lineárních programů.

Pokud skončí algoritmus v kroku 5, potom z Vět 4.1, 3.5 a 3.7 plyne, že $\boldsymbol{A}$ je singulární. Pokud algoritmus skončí v kroku 11, potom našel omezenou souvislou komponentu. Z Věty 3.6 plyne, že je to jediná komponenta a že $\boldsymbol{A}$ je regulární. Z Věty 4.1 potom plyne, že vypočítaná hodnota $\diamond\Sigma$ je opravdu intervalový obal množiny řešení úlohy.

Pokud je $\boldsymbol{A}$ regulární, potom v průběhu algoritmu navštívíme všech 2^n ortantů. V každém ortantu řešíme $2n$ lineárních programů, který je každý řešitelný v polynomiáním čase. Proto je výsledná časová složitost $\mathcal{O}(2^n)$.

Tak jak je algoritmus zapsaný, je jeho prostorová složitost exponenciální vzhledem ke vstupu. Není však potřeba generovat celou množinu Y_n a ukládat ji do paměti, můžeme nově vygenerovanou signaturu ihned použít k popsání lineárních

programů v daném orthnatu a zahodit. Také nemusíme shromažďovat všechny meze $[\underline{h(s)}, \overline{h(s)}]$, postačí nám, když budeme udržovat intervalový obal množiny řešení v prošlých ortantech $\diamond\Sigma'$, který v každém dalším ortantu případně rozšíříme. Prostorová složitost takto modifikovaného algoritmu pak závisí na zvolené metodě lineárního programování.

4.3 Janssonův algoritmus

Hlavní neefektivita předchozího algoritmu spočívá v tom, že prochází všechny ortanty bez ohledu na to, jestli se tam nějaká komponenta množiny řešení nachází nebo ne. Tuto neefektivitu odstraňuje následující algoritmus, který popsal C. Jansson ve svém článku (Jansson, 1997). Nejprve definujeme grafovou strukturu tak, abychom se mohli lépe bavit o pojmu sousední ortant.

Definice (Reprezentační graf)**.** V závislosti na množině řešení Σ definujeme graf $G = (V, E)$ s množinou vrcholů:

$$V := \{s \in \{-1, +1\}^n; \Sigma(s) \neq \varnothing\}$$

a množinou hran:

$$E := \{(s, t); s, t \in V, \exists! k : s_k \neq t_k, \Sigma(s) \cap \Sigma(t) \neq \varnothing\}$$

a říkáme, že graf G reprezentuje množinu řešení Σ. Vrcholy spojené hranou nazýváme sousední a množinu sousedních vrcholů vrcholu s značíme $N(s)$.

Následující věta charakterizuje vztah množiny řešení Σ s jejím reprezentačním grafem G.

Věta 4.2. *(a) Každá neprázdná souvislá komponenta $\hat{\Sigma}$ množiny řešení Σ může být reprezentována jako:*

$$\hat{\Sigma} = \bigcup\{\Sigma(s); s \in U\},$$

kde U je množina vrcholů komponenty souvislosti grafu G.

(b) Pokud je Σ souvislá a omezená, potom je graf G souvislý a

$$\Sigma = \bigcup\{\Sigma(s); s \in V\}.$$

Důkaz. Viz (Jansson, 1997). □

Věta 4.3. *Následující tvrzení jsou pro $s \in V$ ekvivalentní:*

(a) $t \in N(s)$,

(b) $\exists k \in \{1 \ldots n\}$*:* $s_k = -t_k$, $s_i = t_i$ *pro* $i \in \{1 \ldots n\}, i \neq k$ *a* $\min_{x \in \Sigma(s)} s_k x_k = 0$.

Důkaz. Viz (Jansson, 1997). □

Informaci, které sousední ortanty musíme navštívit, získáme díky Větě 4.3 téměř zadarmo. Lineární programy určující sousední vrcholy již v ortantu řešíme kvůli výpočtu mezí množiny řešení.

4.3.1 Popis algoritmu

Algoritmus je grafové prohledávání, které začíná tak, že spočítá libovolné řešení $x \in \Sigma$. Jeho znaménkový vektor $s = sgn(x)$ určí první vrchol reprezentující ortant, ve kterém jsou počítány meze množiny řešení. Tím se určí množina sousedů $N(s)$ a stejně se pak postupuje v každém následujícím sousedním vrcholu. Algoritmus si musí pamatovat již navštívené vrcholy tak, aby se do nich nevracel.

Algorithm 3 Janssonův algoritmus

Require: $\boldsymbol{A} \in \mathbb{IR}^{m\times n}, \boldsymbol{b} \in \mathbb{IR}^m$

1: vyřeš $A^c x = b^c$;
2: $s := sgn(x)$;
3: $L := \{s\}$ (L je seznam vrcholů k navštívení);
4: $U := \varnothing$ (U je seznam navštívených vrcholů);
5: **while** $L \neq \varnothing$ **do**
6: odeber prvek s ze seznamu L;
7: $U := U \cup s$;
8: **for** k:=1 ... n **do**
9: vyřeš LP $\max_{x\in\Sigma(s)} s_k x_k$
10: **if** $\max_{x\in\Sigma(s)} s_k x_k$ je neomezené **then** konec, $\boldsymbol{A}$ je singulární
11: **end if**
12: ulož optima do přislušných složek $[\underline{h(s)}, \overline{h(s)}]$ podle Věty 4.1
13: **end for**
14: **for** k:=1 ... n **do**
15: vyřeš LP $\min_{x\in\Sigma(s)} s_k x_k$
16: **if** $\min_{x\in\Sigma(s)} s_k x_k$=0 **then** ulož vrchol t do N(s) podle Věty 4.3
17: **end if**
18: ulož optima do přislušných složek $[\underline{h(s)}, \overline{h(s)}]$ podle Věty 4.1
19: **end for**$L := L \cup \{N(s) - U\}$;
20: **end while**
21: ze vztahů (4.1) a (4.2) vypočítej intervalový obal $[\underline{\diamond\Sigma}, \overline{\diamond\Sigma}]$
22: **return** $\diamond\Sigma$

4.3.2 Analýza algoritmu

Množství signatur ortantů je konečné. Tedy i množina vrcholů grafu je konečná, a jelikož se díky kroku $L := L \cup \{N(s) - U\}$ nevracíme do již navštívených vrcholů, je počet průchodů skrz while smyčku konečný. V té počítáme konečné množství lineárních programů. Tedy algoritmus je konečný.

Pokud skončí algoritmus v kroku 10, potom z Vět 4.1, 3.5 a 3.7 plyne, že $\boldsymbol{A}$ je singulární. Pokud algoritmus skončí a $L \neq \varnothing$, potom algoritmus uloží do proměnné U všechny vrcholy, které jsou ve stejné komponentě souvislosti grafu G jako startovní vrchol s. Ta odpovídá z Věty 4.2(a) souvislé komponentě $\hat{\Sigma}$. Jelikož ta je omezená, plyne z Vět 3.6 a 4.2(b), že $\hat{\Sigma} = \Sigma$ a $\boldsymbol{A}$ je regulární. Z věty 4.1 potom opět plyne, že vypočítaná hodnota $\diamond\Sigma$ je opravdu intervalový obal množiny řešení úlohy.

V každém ortantu řeší algoritmus $2n$ úloh lineárního programování. Ty jsou řešitelné v polynomiálním čase. V případě, že je počet ortantů s neprázdným prů-

nikem s množinou řešení polynomiálně omezený, potom algoritmus skončí v polynomiálním čase. V nejhorším případě je opět složitost $\mathcal{O}(2^n)$ a to i pro velmi jednoduché příklady. Stačí zvážit $A = I_n$ a $b = [-1, 1]^n$. V praxi může být časová úspora značná, zvláště když režie spojená s efektivním procházením ortantů je oproti času stráveným počítáním lineárních programů zanedbatelná.

Jelikož je nutné si pamatovat již navštívené vrcholy, je i prostorová složitost algoritmu $\mathcal{O}(2^n)$.

4.4 Využití Janssonova algoritmu pro test regularity intervalové matice

Kontrola regularity intervalové matice je NP-těžký problém (Poljak a Rohn, 1993). Postup využití Janssonova algoritmu pro testování regularity matice zmiňují (Jansson a Rohn, 1999). Pokud totiž metoda najde omezenou množinu řešení, je potom dle Věty 3.6 intervalová matice soustavy regulární. Jansonův algoritmus nám tedy také poskytuje univerzální metodu pro rozhodování requlárity intervalové matice, která může skončit dříve než po exponenciálně mnoha krocích.

5. Verifikace v lineárním programování

Doposud jsme předpokládali, že počítáme s přesnou aritmetikou. V tom případě je výpočet intervalového obalu množiny řešení pomocí Janssonova algoritmu správný. Při implementaci algoritmu na počítačích musíme zohlednit používanou floating-point aritmetiku a z ní plynoucí zaokrouhlovací chyby. Veškeré meze množiny řešení počítané Jannsonovým algoritmem získáváme pomocí lineárního programování, konkrétně jsou to hodnoty účelové funkce v optimálním řešení. Budeme požadovat verifikované meze, tj. interval spočítaný na počítači za použití floating point aritmetiky, ve kterém se skutečná mez nachází. Hodnotu intervalového obalu v dané složce potom zvolíme z krajních hodnot tohoto intervalu tak, aby výsledný spočítaný intervalový obal obsahoval ten skutečný, který bychom získali při přesném výpočtu. Metodě lineárního programování, která spočítá verifikované meze, říkáme verifikovaná metoda lineárního programování. Nejprve stručně zavedeme základní pojmy spojené s lineárním programováním. Poté představíme dvě verifikované metody založené na odlišných vlastnostech lineárního programování.

5.1 Základní pojmy a značení lineárního programování

Definice (Standardní úloha)**.** Úlohou lineárního programování ve standardním tvaru rozumíme

$$\max\{c^T x; x \in X\}, X := \{x \in \mathbb{R}^n; Ax = b, x \geq 0\},$$

kde A $= \mathbb{R}^{m\times n}, b \in R^m, c \in R^n, m < n$. Je dobře známé, že lze libovolnou úlohu lineárního programování převést na její standardní tvar. V různých literaturách je alternativně standardní forma uváděna také jako minimalizační úloha. Pokud nebude uvedeno jinak, budeme předpokládat úlohu maximalizační. Funkci $f = c^T x$ nazýváme účelovou funkcí. Množina X je konvexní polyedr množiny přípustných řešení. Řešení $x^\star$ je optimální, pokud platí $c^T x^\star \geq c^T x$ pro všechna přípustná řešení x.

Definice (Báze)**.** Množinu indexů $B = \{\beta_1 \dots \beta_m\} \subset \{1 \dots n\}$ nazveme bází lineárního programu, pokud systém

$$\sum_{\beta\in B} a_{i\beta} x_{i\beta} = b_i, i \in \{1 \dots m\}$$

má jednoznačné řešení $x_B = (x_{\beta_1} \dots x_{\beta_m})$. Vektoru x_B říkáme vektor bazických proměnných a vektor $x_N = (x_{\gamma_1} \dots x_{\gamma_{n-m}})$, kde $N = \{1 \dots n\} \setminus B$, značí vektor nebazických proměnných. Podobně rozdělujeme matici $A = \{a_1 \dots a_n\}$ na $A_B = (a_{\beta_1} \dots a_{\beta_m})$ a $A_N = (a_{\gamma_1} \dots a_{\gamma_{n-m}})$ a stejně tak vektor c na c_B a c_N. Pokud máme $x_N = 0$ a zároveň platí podmínka $x_B \geq 0$, nazýváme vektor $x(B) = (x_B, x_N))$ přípustné bazické řešení.

Hlavní věta lineárního programování říká, že pokud existuje optimální řešení lineárního programu, potom má nejvýše m nenulových složek. Viz např. (Dantzig, 1963). Jsou to právě přípustná bazická řešení. Bázi určující optimální řešení nazýváme optimální.

5.2 Verifikace optimality báze

Existuje mnoho efektivních metod, jak spočítat optimální řešení lineárního programu. Asi nejznámější z nich je Simplexová metoda (Dantzig, 1963). Jejich implementace na počítačích většinou zanedbává zaokrouhlovací chyby a výstupem je aproximace skutečného řešení. Obvykle je tato aproximace velmi dobrá a použitelná pro většinu aplikací. Bylo však ukázáno (Kendall, 1963), že existují systémy, které jsou citlivé i na velmi malé změny koeficientů a výsledná aproximace řešení se kvůli zaokrouhlovaní může velmi lišit od očekávané hodnoty. Existují však postupy, jak ověřit správnost řešení a určit verifikovaný interval řešení. Metodu v této kapitole publikoval C. Jansson v článku (Jansson, 1988) jako součást jeho verifikované metody lineárního programování s tolerancemi. Předpokládá využití existujících verifikovaných metod pro řešení klasických soustav rovnic.

Popis metody Mějme nyní $B = \{\beta_1 \dots \beta_m\} \subset \{1 \dots n\}$ odhad optimální báze. Pomocí libovolné verifikované metody pro řešení neintervalových soustav rovnic vyřešíme soustavy:

$$\begin{aligned} A_B x_B &= b, \\ A_B^T y &= c_B. \end{aligned}$$

Tím získáme verifikované intervalové vektory $\boldsymbol{x_B}$ a $\boldsymbol{y}$, ve kterých se skutečné hodnoty nalézají. Zároveň získáme ověření, zda je matice A regulární. V tom případě definujeme:

$$\begin{aligned} \boldsymbol{d_N} &:= A_N \boldsymbol{y} - c_N, \\ \boldsymbol{x(B)} &:= (\boldsymbol{x_B}, x_N),\ x_N := 0, \\ \boldsymbol{f} &:= c_B \boldsymbol{x_B}. \end{aligned}$$

Spočítaný interval $\boldsymbol{f}$ a intervalový vektor $\boldsymbol{x(B)}$ potom obsahují skutečné hodnoty f a $x(B)$. Následující věta poskytne jednoduše ověřitelné kriterium optimality řešení.

Věta 5.1. *Pokud platí $\boldsymbol{x_B} > 0$ a zároveň $\boldsymbol{d_N} > 0$, potom existuje jediné optimální řešení a je obsažené v $\boldsymbol{x(B)}$.*

Důkaz. Viz (Jansson, 1988). □

Metodu jsme implementovali jako pomocnou funkci `verSimplex`. Pro verifikované řešení soustav rovnic používáme funkci `verifylss` z balíku INTLAB. Naše experimenty ukázaly, že ve většině případů tato metoda uspěje a pro malé a střední soustavy je i dostatečně rychlá. V případě že optimální řešení není jediné, kritérium z Věty 5.1 selže a musíme zvolit další postup. Existují způsoby, jak efektivně rozpoznat další báze, ve kterých se nachází další optimální řešení lineárního programu (Jansson, 1988). Těch však může být nakonec velmi mnoho a verifikace tak bude výpočetně náročná, proto v tomto případě používáme metodu založenou na dualitě úloh lineárního programování.

5.3 Verifikace pomocí duální úlohy

Metodu v tomto oddílu můžeme použít, pokud nepožadujeme verifikované řešení $\boldsymbol{x_B}$, ale stačí nám horní odhad na účelovou funkci, jako tomu je v našem případě. Ten jsme schopni spočítat na počítači za pomocí řízeného zaokrouhlování. Pro tento účel lze v prostředí MATLAB vyžít metody setround z balíku INTLAB. Následující postup je představen v (Neumaier a Shcherbina, 2004) a prezentuje metodu příbuznou metodě z (Jansson, 2004).

Definice (Duální úloha)**.** Duální úloha k úloze lineárního programování ve standardním tvaru je úloha:

$$\min\{b^T y; y \in Y\}, Y := \{y \in \mathbb{R}^m; A^T y \geq c\}.$$

Popis metody Mějme nyní y aproximaci optimálního řešení duální úlohy. Označme:

$$r = A^T y - c. \tag{5.1}$$

Máme $c = A^T x - r$ a tedy:

$$c^T x = (A^T y - r)^T x = y^T A x - r^T x = b^T y - r^T x. \tag{5.2}$$

Použitím vhodného zaokrouhlování jsme schopni spočítat interval $\boldsymbol{r}$, ve kterém se nachází výsledek rovnice (5.1). Dosazením jeho dolní meze $\underline{r}$ do (5.2) dostáváme:

$$c^T x \leq b^T y - \underline{r}^T x. \tag{5.3}$$

Označme nyní $\underline{r}_{neg} = \min(\underline{r}, 0)$ a předpokládejme, že máme verifikované horní meze $\overline{x}$ všech proměnných vektoru x, to jest platí $x \leq \overline{x}$. Pravou stranu nerovnice (5.3) pak dále odhadneme:

$$c^T x \leq b^T y - \underline{r}^T x \leq b^T y - \underline{r}_{neg} \overline{x}. \tag{5.4}$$

Tento odhad jsme již schopni za pomocí zaokrouhlování nahoru spočítat verifikovaně i s použitím floating-point aritmetiky. Vektor $\overline{x}$ můžeme v našem případě získat například pomocí výpočtu intervalové obálky množiny řešení ještě před samotným výpočtem intervalového obalu. Jak uvidíme později v Kapitole 7, výpočet intervalové obálky je typicky výrazně rychlejší než výpočet intervalového obalu, a proto je cena za tento předvýpočet zanedbatelná. Tuto metodu jsme neimplementovali, ale využili jsme knihovny VSDP (Verified SemiDefinite Programming) (Harter et al., 2012), která je volně dostupná pro soukromé a akademické

účely. Závěrem je třeba poznamenat, že verifikaci pomocí duální úlohy bychom mohli používat v celém průběhu výpočtu. Jak ale uvidíme v další kapitole, je pro nás výrazně výhodnější počítat aproximaci primárního řešení a následně hledat verifikované řešení za pomocí metody, která na tento výpočet přímo navazuje.

6. Optimalizace Janssonova algoritmu

Z popisu algoritmu plyne, že každý ortant, který má neprázdný průnik s množinou řešení, navštívíme právě jednou. Je to právě z důvodu, jak definujeme reprezentační graf a použitím pravidla pro získání ortantů sousedních. Doba výpočtu je pak závislá na počtu ortantů a na době výpočtu v jednotlivých ortantech. Cena výpočtu v každém ortantu je poměrně vysoká - je nutné spočítat $2n$ lineárních programů. Je však možné využít jejich vzájemné souvislosti a jejich výpočet zefektivnit. V této kapitole nastíníme použité postupy uplatněné v naší implementaci.

6.1 Odhad počátečního přípustného řešení

Všechny lineární programy $\min_{x\in\Sigma(s)} s_k x_k$ a $\max_{x\in\Sigma(s)} s_k x_k$ v daném ortantu sdílí množinu přípustných řešení. Jejich optimální řešení jsou zároveň přípustná řešení programů ostatních. Dále je intuitivní předpokládat, že optima skupin (min/max) problémů leží "relativně blízko sebe", myšleno počtem kroků simplexového algoritmu (Jansson, 1997). Pokud máme například úlohu, ve které musíme navštívit všechny ortanty, je potom v každém ortantu společným optimem minimalizačních úloh počátek os souřadnic. V obecném případě je tato heuristika vhodným nástrojem, jak odhadnout směr, kterým se bude výpočet lineárního programu ubírat. V tabulce je časová úspora v procentech oproti algoritmu bez této optimalizace.

velikost soustavy	úspora
5x5	3
10x10	6
15x15	10
20x20	13

Tabulka 6.1: Optimalizace odhadem přípustného řešení

6.2 Vynechání programů na základě mezí

Předchozí myšlenku můžeme ještě rozšířit. Pro jednoduchost předpokládejme, že pracujeme v kladném ortantu tj. v ortantu se signaturou $\mathbb{1}$. Pokud optima lineárních programů obsahují nulovou složku x_k, můžeme pak lineární program $\min_{x\in\Sigma(s)} s_k x_k$ vynechat a hodnotě intervalového obalu v dané složce a směru přiřadit nulu, jelikož jsme již během předchozího výpočtu našli přípustné řešení, které je pro tento problém optimální. Tento princip je možné použít pro libovolnou jednoduchou mez, v našem algoritmu se takové vyskytují jako omezující podmínky určující ortant. Takto můžeme postupovat v libovolném ortantu, je však potřeba dát pozor, který směr (min/max) můžeme takto vynechat, což závisí na signatuře ortantu. Efekt této optimalizace typicky roste s počtem sousedních ortantů a největší efekt má při plném počtu ortantů s neprázdným průnikem s množinou řešení. Úspora v takovém případě je v tabulce v procentech.

velikost soustavy	úspora
2x2	30
4x4	25
6x6	16
8x8	12

Tabulka 6.2: Optimalizace vynecháním programů na základě mezí

6.3 Ořezávání množiny přípustných řešení LP

Dosavadní optimalizace využívaly souvislosti lineárních programů v jednom ortantu. Vypočítané meze však můžeme dát do souvislosti s mezemi v ostatních ortantech. Po výpočtu mezí v každém ortantu dostaneme intervalový obal již prošlých ortantů, který je podmnožinou výsledného hledaného intervalového obalu celé množiny řešení. Myšlenku uvedeme na konkrétním příkladu. Uvažme opět kladný ortant se signaturou $\mathbb{1}$. Při počítání mezí v kladném směru složky (maximalizační LP) můžeme použít hodnotu z již spočítaného intervalového obalu jako dolní mez na hodnotu účelové funkce. Díky tomu můžeme tímto odhadem oříznout množinu přípustných řešení pro tento výpočet a v extrémním případě dostat množinu prázdnou. Dolní odhad na intervalový obal množiny řešení můžeme také získat přibližným rychlým algoritmem pro výpočet vnitřní obálky ještě před začátkem algoritmu. Opět uvádíme úsporu v procentech v následující tabulce.

velikost soustavy	úspora
2x2	2
4x4	36
6x6	55
8x8	61

Tabulka 6.3: Optimalizace ořezáváním množiny přípustných řešení LP

7. Testování implementace

V této kapitole představíme naměřené výsledky časové náročnosti naší implementace. Abychom je mohli lépe interpretovat, budeme pro srovnání zároveň na stejných vstupních datech měřit i další metody. V první řadě se nabízí srovnání s metodou `verintervalhull` z balíku VERSOFT J. Rohna, jelikož tato metoda také počítá verifikovaný intervalový obal množiny řešení. Pro zajímavost také uvedeme srovnání s často používanou metodou INTLABu `verifylss` pro spočítání verifikované obálky.

7.1 Postup testování

Veškeré testy jsme provedli na notebooku Samsung Ativ7:

Procesor: Inter Core i5-3337u, 1.8 Ghz, x64,
RAM: 4 GB,
OS: Windows 8, 64 bit.

Metody jsme testovali následujícím způsobem. Nejprve jsme vygenerovali náhodnou soustavu požadované velikosti $A^c x = b^c$ s koeficienty z intervalu $[-100, 100]$. Poté jsme vygenerovali matici A^Δ a vektor b^Δ. Koeficienty těchto matic tvořila náhodná čísla v rozmezí 0 až R, kde R jsme volili ve třech variantách, abychom získali odhad výkonu algoritmu na různě těsných intervalech. Tuto soustavu jsme poté řešili každou testovanou metodou. Výsledky, kdy intervalová soustava obsahovala singulární matici (a množina řešení byla neomezená), jsme do následujících výsledků nezahrnovali, jelikož toto zjistit trvá typicky mnohem kratší dobu. Měřili jsme pomocí funkcí `tic` a `toc` z prostředí Matlab. Toto měření jsme opakovali podle časové náročnosti soustavy. Pro malé soustavy byl počet opakování v rozmezí stovek až tisíců testů, pro větší soustavy byl kvůli časové náročnosti omezen na stovky testů. Minimální počet pro největší testované soustavy bylo 100 opakování. Dále někde uvádíme pro orientaci výsledky větších soustav, kde by takové testování bylo nepraktické a výsledky jsou založeny pouze na jednotkách pokusů. Takovéto výsledky označujeme symbolem " $\star$ ".

Mimo měření doby výpočtu jsme také porovnávali šířku spočítaných intervalů. Podle předpokladů se tento výsledek mezi metodami `ilsjanssonhull` a `verintervalhull` téměř neliší a rozdíl je způsobený pouze rozdílnými metodami verifikace a z ní plynoucími nadhodnoceními v posledních cifrách. Z tohoto důvodu uvádíme pouze zajímavější výsledky poměru obálky spočítané metodou `verifylss` a metodou `ilsjanssonhull`. Tím dostaneme představu, o kolik může metoda počítající obálku nadhodnotit oproti přesnému výsledku metody počítající intervalový obal. Poměr obálek počítáme jako průměrnou hodnotu z podílů šířek intervalů ve všech složkách vektoru řešení.

7.2 Výpočet intervalového obalu

V prvním měření jsme volili $R := 0{,}00001$. Výsledky v sekundách jsou v následující tabulce. Sloupec obálka značí uvedený poměr obálky metod `verifylss` a `ilsjanssonhull`.

velikost matice	ilsjanssonhull	verintrevalhull	verifylss	obálka
4x4	0,08842	0,00598	0,00317	1,00008
7x7	0,18733	0,09829	0,00320	1,00009
10x10	0,34035	0,14473	0,00329	1,00018
13x13	0,64541	0,21135	0,00335	1,00026
16x16	0,99109	0,27020	0,00345	1,00035
19x19	1,42487	0,32883	0,00346	1,00036
22x22	2,01261	0,40262	0,00339	1,00033

Tabulka 7.1: Výsledky testování pro $R := 0{,}00001$

Je vidět, že na tomto typu úlohy je metoda `verifylss` počítající intervalovou obálku množiny řádově rychlejší než obě metody počítající intervalový obal. Vypočítaná obálka je pak v průměru relativně velmi těsná. Pro výpočet intervalového obalu vychází lépe metoda VERSOFTu. Jelikož je počet navštívených ortantů v průměru velmi malý (a často byl navštíven pouze jeden), přičítáme tento výsledek efektivnější metodě výpočtu mezí v ortantu, než pomocí lineárního programování, jako v případě naší implementace Janssonova algoritmu. V další tabulce najdeme výsledky, pokud zvýšíme R na 0,1.

velikost matice	ilsjanssonhull	verintrevalhull	verifylss	obálka
4x4	0,10087	0,07055	0,00373	1,19380
7x7	0,33062	0,22732	0,00459	1,68149
10x10	0,93518	0,85385	0,00513	2,00256
13x13	3,25804	3,19073	0,00586	2,21660

Tabulka 7.2: Výsledky testování pro $R := 0{,}1$

I v tomto případě je metoda `verifylss` výrazně rychlejší, ale vypočítaná obálka již není těsná, přičemž s narůstající velikostí soustavy dává metoda horší výsledky. Průměrný navštívený počet ortantů narůstá a metody pro výpočet intervalového obalu jsou v tomto případě mnohem vyrovnanější. Proč tomu tak je, uvidíme lépe na dalším typu úlohy, kdy musíme navštívit všech 2^n ortantů. Takové úlohy můžeme získat volbou $\boldsymbol{b}$ obsahující nulový vektor. R volíme 0,5.

velikost matice	ilsjanssonhull	verintrevalhull	verifylss	obálka
2x2	0,09285	0,10931	0,00412	1,03861
4x4	0,51441	0,76710	0,00561	1,19807
6x6	2,58137	4,49316	0,00720	1,63785
8x8	19,53200	34,37164	0,01257	2,10563
10x10*	80,15230	119,04959	0,01691	2,30563

Tabulka 7.3: Výsledky testování pro úlohy velikosti 2^n a $R := 0{,}5$

Zde je již naše metoda `ilsjansonhull` výrazně rychlejší než metoda VERSOFTu. Výsledky ukazují, že se vzrůstajícím počtem ortantů je výpočet mezí založený na lineárním programování v ortantu efektivnější, než u konkurenčního algoritmu, jelikož dokáže využívat silné souvislosti mezi jednotlivými programy. Je však nutné poznamenat, že oba algoritmy běží velmi dlouho i pro relativně malé soustavy a se vrůstajícím počtem proměnných jsou jen těžko použitelné pro běžné výpočty. Navíc úlohy, kdy musíme prohledat všech 2^n ortantů můžeme generovat s libovolně úzkým R, příkladem mohou být úlohy $I_n = R * [-\mathbb{1}, \mathbb{1}]$.

Hodnoty naměřené v Tabulkách 7.1 a 7.2 tak maximálně vypovídají o průměrném chování algoritmu, může se však stát, že algoritmus poběží mnohem déle. Dobu výpočtu algoritmu lze typicky odhadnout součinem počtu ortantů a dobou výpočtu v jednom ortantu. Tu uvádíme v následující tabulce

velikost matice	ilsjanssonhull
5x5	0,15804
10x10	0,44684
15x15	1,07036
20x20	1,96660

Tabulka 7.4: Doba výpočtu v 1 ortantu

8. Uživatelská dokumentace

V této kapitole se nachází uživatelská příručka k přiložené implementaci řešiče. Zde se nachází návod instalace programu a knihovny INTLAB v prostředí MATLAB. Popíšeme způsob, jakým se pomocí INTLABu reprezentují intervalové soustavy rovnic a představíme práci s vlastním řešičem, a jak interpretovat výstup programu.

8.1 Instalace softwaru

8.1.1 Instalace INTLABu

Naše implementace využívá různé části INTLABu, především datový typ intval a intervalovou aritmetiku. Je nutné přidat složku s knihovnou INTLAB a její vybrané podsložky do cesty MATLABu. V adresáři knihovny INTLAB se nachází script `startup.m`, který toto provede automaticky. V MATLAB konzoli ho spustíme následovně:

```
>> startup
```

INTLAB je možné získat na adrese `www.ti3.tu-harburg.de/rump/intlab/` .

8.1.2 Instalace řešiče

Vlastní implementovaný řešič se nachází ve složce janssonsolver. Tuto složku a její podsložky je nutné přidat do cesty MATLABu. To se provádí příkazem:

```
>> addpath(genpath( cesta_k_adresari ))
```

Alternativně je možné použít script `ilsjanssonhullstart.m`. Script otevřeme a spustíme:

```
>> ilsjanssonhullstart
```

Tento script rozpozná složku, ve které se nachází hlavní soubor řešiče automaticky. Mimoto se tento script postará o překlad potřebné MEX funkce datových struktur na binární MEX-soubor. MEX-soubory jsou specifické pro různé platformy. Ve složce je přiložen binární MEX-soubor pro platformu Win32, pro ostatní je nutné provést překlad zvlášť. Alternativou ke scriptu je ruční instalace MEX funkcí.

8.1.3 Ruční instalace MEX funkcí

Nejprve je nutné mít nainstalovaný překladač C/C++ MEX souborů. Na adrese `http://www.mathworks.com/support/compilers/R2013a/` nalezneme seznam podporovaných překladačů. Námi preferovaný překladač vybereme v dialogu příkazem:

```
>> mex -setup
```

Poté již následujícím způsobem přeložíme MEX soubor, který se nachází ve složce janssonsolver.

```
>> mex mxTrie.c
```

Přípona vygenerovaného binárního souboru se liší podle platformy. Zjistit ji můžeme příkazem:

```
>> mexext
```

Například na platformě Win32 je tato přípona `.mexw32`.

8.2 Reprezentace intervalových soustav rovnic

Intervalové soustavy rovnic popisujeme pomocí intervalové matice $\boldsymbol{A}$ a intervalového vektoru $\boldsymbol{b}$. V knihovně INTLAB jsou intervalová data reprezentována datovým typem intval. INTLAB nabízí tři možnosti, jak vytvářet intervalové matice. Prvním způsobem je zadání pomocí dolní a horní meze matice funkcí `infsup()`. Přiklad:

```
>> Asup = [4  3; -1  1]
Asup =
     4     3
    -1     1
>> Ainf = [2  -1; -1  -3]
Ainf =
     2    -1
    -1    -3
>> A = infsup(Ainf,Asup)
intval A =
[    2.0000,    4.0000] [   -1.0000,    3.0000]
[   -1.0000,   -1.0000] [   -3.0000,    1.0000]
```

Střed intervalové matice a její poloměr získáme funkcemi `mid()` a `rad()`

```
>> mid(A)
ans =
     3     1
    -1    -1
>> rad(A)
ans =
     1     2
     0     2
```

Matici můžeme také zadat přímo pomocí středové matice a matice poloměru pomocí funkce `midrad()`. Příklad:

```
>> Amid = [0  0; 0  0]
Amid =
     0     0
     0     0
>> Arad = [1  3; 2  2]
Arad =
     1     3
     2     2
>> A = midrad(Amid,Arad)
intval A =
[   -1.0000,    1.0000] [   -3.0000,    3.0000]
[   -2.0000,    2.0000] [   -2.0000,    2.0000]
```

Meze intervalové matice pak nazpět získáme pomocí funkcí `inf()` a `sup()`.

```
>> inf(A)
ans =
    -1    -3
    -2    -2
>> sup(A)
ans =
     1     3
     2     2
```

Speciálním případem intervalové matice je klasická reálná matice, tedy intervalová matice obsahující jen degenerované intervaly. Převod reálné matice do intervalového typu intval provedeme následujícím způsobem:

```
>> A = [1  2  ; 3  4]
A =
     1     2
     3     4
>> A = intval(A)
intval A =
    1.0000    2.0000
    3.0000    4.0000
```

8.3 Volání řešiče a interpretace výstupu

Hlavní funkce řešiče se jmenuje `ilsjansonhull.m`. Funkce řeší intervalovou soustavu rovnic a vrací intervalový obal množiny řešení. Její volání provedeme jedním z následujících třech způsobů:

```
>> [x] = ilsjanssonhull(A,b)
>> [x,flag] = ilsjanssonhull(A,b)
>> [x,flag,orthants] = ilsjanssonhull(A,b)
```

Vstupní parametry jsou oba povinné a jsou to:

- A - intervalová matice koeficientů soustavy $\boldsymbol{A} \in \mathbb{IR}^{m \times n}$,
- b - intervalový vektor pravých stran $\boldsymbol{b} \in \mathbb{IR}^{m}$.

Výstupní parametry jsou:

- x - intervalový obal množiny řešení,
- flag - výstupní příznaky,
- orthants - počet navštívených ortantů.

Typické použití řešiče představíme na příkladu. Uvažme znovu intervalovou soustavu z Příkladu 3.1.

$$\begin{pmatrix} [2,4] & [-1,0] \\ [0,1] & [1,2] \end{pmatrix} x = \begin{pmatrix} [-2,5] \\ [-3,5] \end{pmatrix}$$

Nejprve definujeme intervalovou matici $\boldsymbol{A}$ a vektor pravých stran $\boldsymbol{b}$.

```
>> A = infsup([2 -1; 0 1], [4 0;1 2])
intval A =
[    2.0000,    4.0000] [   -1.0000,    0.0000]
[    0.0000,    1.0000] [    1.0000,    2.0000]
>> b = infsup([-2; -3], [5 ;5])
intval b =
[   -2.0000,    5.0000]
[   -3.0000,    5.0000]
```

Poté zavoláme funkci `ilsjanssolnhull`.

```
>> [x,flag,orthants] = ilsjanssonhull(A,b)
intval x =
[   -2.5000,    5.0000]
[   -5.5000,    6.0000]
flag =
     1
orthants =
     4
```

V tomto případě našel program verifikované řešení. Pokud z různých důvodů nebylo možné získat verifikované řešení, obsahuje proměnná `x` vektor samých NaN (Not a Number). Různé důvody neúspěchu výpočtu rozlišuje výstupní parametr `flag`. Hodnoty `flag` a jejich vysvětlení najdeme v následující tabulce.

flag	význam
1	výpočet skončil úspěšně, matice $\boldsymbol{A}$ je regulární
-2	nenalezeno žádné verifikované řešení
-3	množinu řešení se nepodařilo verifikovaně omezit
-4	výpočet narazil na NaN

Tabulka 8.1: Význam hodnot parametru flag

Na závěr uvedeme odlišnost od některých metod pro řešení intervalových soustav rovnic. Uvažujme tuto úlohu.

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

Řešíme-li úlohu pomocí metody `verifylss` získáme následující výsledek.

```
>> [x] = verifylss(A,b)
intval x =
    0.2000
    0.4000
```

Pokud však řešíme úlohu naší metodou, získáme odlišný výstup.

```
[x,flag] = ilsjanssonhull(A,b)
intval x =
       NaN
       NaN
flag =
    -3
```

Hodnota parametru `flag` = -3 naznačuje, že je množina řešení pravděpodobně neomezená. Vskutku tomu tak je, přesto metoda `verifylss` vydá intervalový vektor řešení. To plyne z rozdílné metody řešení soustavy a přístupu k řešitelnosti podurčených soustav. Náš výstup je naopak konzistentní s metodou `verintervalhull`.

```
>> [x] = verintervalhull(A,b)
intval x =
       NaN
       NaN
```

8.4 Alternativní implementace

Jelikož se podpora MEX funkcí liší v různýc verzích MATLABu, přikládáme ještě implementaci, která MEX funkce nevyužívá. To sice vede k nižší efektivitě výpočtu některých úloh, protože nemůžeme využít vlastní datové sturktury, negativní efekt na dobu výpočtu je však většinou zanedbatelný. Výhodou této implementace je využití pouze standardních datových struktur MATLABu a z toho plynoucí podpora ve starších i novějších verzích. Tuto implementaci nalezeneme v souboru `ilsjanssonhullAlt.m`. Její použití je pak téměř totožné se standartní implementací (akorát změníme jméno funkce při volání).

9. Programátorská dokumentace

Zde popíšeme strukturu programu, použité algoritmy, datové struktury a implementační detaily. Představíme také základní strukturu MEX souboru a jeho využití.

9.1 Implementace Janssonova algoritmu

Řešič jsme implementovali v prostředí MATLAB 2013a (32-bit). Hlavní funkce řešiče je obsažená v souboru `ilsjanssonhull.m`. Zde implementujeme Janssonův algoritmus popsaný v oddílu 4.3. Inicializace výpočtu proběhne vyřešením soustavy rovnic pro středový systém a přidáním signatury řešení do fronty. Poté již algoritmus ve while-smyčce postupně z fronty odebírá znaménkové vektory, ve kterých je potřeba ohraničit množinu řešení a konstruuje výsledný intervalový obal. Důležité proměnné jsou především:

- `out` - intervalový obal množiny řešení ve zpracovaných ortantech;
- `newbounds` - meze množiny řešení v aktuálně zpracovávaném ortantu;
- `feasible` - příznak určující, zda již bylo nalezeno verifikované řešení v libovolném ortantu, hodnota 1 značí, že již bylo nalezeno, jinak je hodnota 0;
- `minFeasibles` - vektor nejnižších kladných hodnot v každé složce, kterých bylo dosaženo při výpočtu v aktuálním ortantu;
- `neighbours` - vektor, který označuje složky, ve kterých mají sousední ortanty opačnou signaturu, než právě zpracovávaný ortant.

Každý průchod while-smyčkou reprezentuje výpočet v jednom ortantu. Tento výpočet se skládá z následujících kroků:

1. odebrání signatury z fronty;
2. ohraničení množiny řešení lineárními podmínkami;
3. převedení lineárních programů do standardní formy;
4. získání minimálních hodnot mezí v každé složce podle 6.3;
5. ohraničení množiny řešení v ortantu pomocí lineárního programování;
6. přepočítaní intervalového obalu v proměnné `out` podle Věty 4.1;
7. zjištění sousedních ortantů a uložení jejich signatur do fronty.

Zároveň v proměnné `minFeasibles` udržujeme minimální dosažené hodnoty přípustných řešení. Ve standardní formě lineárního programování jsou všechna přípustná řešení kladná. Pokud některá složka vektoru `minFeasibles` nabyde hodnoty 0, vynecháváme příslušný minimalizační lineární program podle myšlenky z 6.2. Na konci výpočtu vnořená funkce `finalise` provede konsolidaci příznaků z verifikovaného lineárního programování do výstupního parametru `flag`.

9.2 Datová struktura signatur ortantů

Během výpočtu algoritmu musíme uchovávat navštívené signatury ortantů, abychom se během výpočtu nezacyklili. První možností, jak toto zařídit, je využít MATLAB matice jako seznam a do ní postupně přidávat znaménkové vektory. Výhodou tohoto řešení je snadná implementace a využití základní struktury MATLABu. Důsledkem je pak snadnější instalace a přenositelnost implementace ze systému na systém i mezi různými verzemi prostředí a zajištěná podpora v prostředí MATLABu. Hlavní nevýhoda tohoto přístupu je pak doba vyhledávání signatury v takovéto struktuře, ta je totiž potom lineárně závislá na počtu již navštívených ortantů. Těch může být až exponenciálně mnoho. To sice na první pohled vypadá jako značná neefektivita, naše testy však prokázaly, že pro velikosti úloh, jež je náš řešič schopen vyřešit v rozumném čase, je toto vzácně úzkým hrdlem programu. Lineární programy v ortantu zvláště ve své verifikované verzi zabírají mnohem více výpočetní doby. Přesto jsme se rozhodli implementovat efektivnější datovou strukturu.

V našem případě se nabízí využití binárního prefixového stromu, neboli binární trie. Bohužel v MATLABu neexistuje žádná standardní implementace stromové struktury. Využili jsme MEX (MATLAB Executable) API, pomocí které je možné v prostředí MATLABu spouštět funkce napsané v C/C++. Jedná se o nestandardní funkčnost MATLABu, proto zde popíšeme alespoň základní strukturu MEX souboru.

9.2.1 Struktura MEX souboru

MEX soubor se typicky skládá ze dvou hlavních částí:

- vlastní výpočetní funkce,
- funkce sloužící jako rozhraní mezi MATLABem a výpočetní funkcí.

Výpočetní funkce je klasická funkce jazyka C. Funkce rozhraní je povinná součást souboru, nahrazuje `main` funkci, zavádí výpočetní funkce a má následující pevně danou strukturu:

```
void mexFunction(int nlhs, mxArray *plhs[],
int nrhs, const mxArray *prhs[]) {}
```

Význam argumentů funkce rozhraní je následující:

- `nlhs` - počet výstupních proměnných,
- `plhs` - pole ukazatelů na výstupní proměnné,
- `nrhs` - počet vstupních parametrů,
- `prhs` - pole ukazatelů na vstupní proměnné.

Funkce:

```
y = inc(x)
```

bude mít hodnoty `nlhs = 1` a `nrhs = 1`, `plhs[0]` a `prhs[0]` budou ukazatele na `mxArray` obsahující `y`, resp. `x`, což je datový typ MATLAB C/C++ API, která slouží jako knihovna pro práci s maticovým typem v jazyce C/C++. C/C++ API obsahuje mimo jiné také funkce pro převod mezi typem `mxArray` a klasickými datovými typy jazyka C a funkce pro tvorbu a kontrolu dat. V mex souboru je pak možné spouštět kód z MEX a C/C++ API, stejně jako klasické funkce jazyka C/C++.

9.2.2 Definované struktury

Soubor `mxTrie.c` obsahuje definice struktur v jazyce C. Prefixový strom signatur orthnatů implementuje struktura `bin_tree`. Jelikož do stromu ukládáme pouze vektory obsahující ± 1, je tento strom binární. Průchod stromem od kořene k listu jednoznačně charakterizuje signaturu ortantu. Signatury ortantů také vystupují ve frontě ortantů, které ještě během výpočtu musíme navštívit. Abychom uspořili místo v paměti, neimplementujeme tuto frontu jako samostatný seznam, nýbrž využíváme listů struktury prefixového stromu. Jednotlivé listy si udržují svého následníka ve frontě pomocí ukazatele na pravého syna, jenž by byl jinak nastavený na hodnotu NULL. Hlavní struktura `trie` udržuje ukazatel na kořen prefixového stromu a ukazatele na první a poslední prvek fronty.

Dále se v souboru `mxTrie.c` nachází metody pro práci s datovými strukturami. V závislosti na prvním parametru volání funkce `mxTrie` je provedena akce z Tabulky 9.1.

parametr	akce
'create'	inicializace datových struktur
'count'	zjištění počtu prvků ve frontě
'insert'	vložení signatury do prefixového stromu a do fronty
'dequeue'	odebrání prvku z fronty
'delete'	smazání datové struktury

Tabulka 9.1: Parametry funkce mxTrie

Tato implementace zajišťuje vyhledávání navštívených ortantů v čase lineárně závislém na velikosti soustavy. To vede u některých typů intervalových soustav k rychlejšímu výpočtu, zvláště pak u úloh, kdy musíme navštívit všech 2^n ortantů.

9.3 Verifikované lineární programování

Verifikovanou metodu lineárního programování, založenou na verifikaci optimální báze, implementuje funkce `verSimplex`. Jako aproximační metodu lineárního programování jsme zvolili funkci `linprog`, což je standardní metoda pro řešení lineárních programů v MATLABu. Funkce `verLinProg` potom zapouzdřuje námi implementovanou metodu `verSimplex` a v případě neúspěchu kritéria využívá k získání verifikovaných mezí funkce z balíku VSDP.

10. Závěr

Výsledkem naší práce je především implementace řešiče intervalových lineárních rovnic využívající Janssonův algoritmus. Řešič, který je dostupný na přiloženém CD, je implementován v prostředí MATLABu za využití intervalové knihovny INTLAB.

Představili jsme několik postupů, jak vylepšit základní algoritmus využitím souvislosti lineárních programů v jednotlivých ortantech. Všechna popsaná vylepšení jsme implementovali a změřili jejich efektivitu. Ukázalo se, že zejména optimalizace na základě ořezávání množiny řešení lineárních programů vede ke značné časové úspoře.

Naši metodu jsme porovnali s již existujícími implementacemi algoritmů pro řešení intervalových soustav rovnic. Naměřené hodnoty naznačují, že pro lineární systémy, jejichž množina proniká velké množství ortantů, je naše implementace výrazně rychlejší než konkurenční metoda VERSOFTu. Pokud je počet prohledávaných ortantů malý, je situace opačná. Obě metody, které počítají verifikovaný intervalový obal pak byly pomalejší než metoda knihovny INTLAB, která počítá pouze intervalovou obálku množiny řešení.

Veškeré meze množiny řešení, ze kterých sestavujeme výsledný intervalový obal, poskytují hodnoty získané z lineárního programování. Aby naše výsledky byly skutečně správné, zvolili jsme a implementovali verifikovanou metodu lineárního programování.

Implementaci algoritmu je možné dále zlepšovat. V současnosti využíváme standardní funkce pro řešení lineárních programů z prostředí MATLABu. To v sobě skýtá omezení pro návrh a nižší efektivitu verifikačních metod pro lineární programování. Dále je možné zvážit naznačený postup výpočtu vnitřní obálky pro optimalizaci ořezáváním množiny řešení lineárních programů.

Seznam použité literatury

DANTZIG, George B. *Linear programming and extensions.* Princeton University Press, Princeton, N.J., 1963. ISBN 0691059136.

FIEDLER, M., NEDOMA, J., RAMÍK, J., ROHN, J., ZIMMERMANN, K. *Linear Optimization Problems with Inexact Data.* Springer, New York, 2006. ISBN 978-0-387-32697-9.

HARTER, V., JANSSON, C., LANGE, M. Vsdp: A matlab toolbox for verified semidefinite-quadratic-linear programming. Report, Institute for Reliable Computing, Hamburg University of Technology, 2012.

JANSSON, Christian. A self-validating method for solving linear programming problems with interval input data. In *Scientific Computation with Automatic Result Verification*, s. 33–45. Springer, New York, USA, 1988. ISBN 3211820639.

JANSSON, Christian. Calculation of exact bounds for the solution set of linear interval systems. *Linear Algebra and Its Applications*, **251**:321–340, 1997. ISSN 0024-3795.

JANSSON, Christian. Rigorous lower and upper bounds in linear programming. *SIAM Journal on Optimization*, **14**(3):914–935, 2004. ISSN 1052-6234.

JANSSON, Christian, ROHN, Jiří. An algorithm for checking regularity of interval matrices. *SIAM J. Matrix Anal. Appl.*, **20**(3):756–776, 1999. ISSN 0895-4798.

KENDALL, Maurice G. Ill-conditioned matrices in linear programming. *Metrika*, **6**(1):60–64, 1963. ISSN 0026-1335.

KRAMER, Walter, ZIMMER, Michael. *Fast (Parallel) dense linear system solvers in C-XSC using error free transformations and BLAS*, s. 230–249. Springer, Dagstuhl, Germany, 2009. ISBN 3642015905.

KREINOVICH, Vladik, ROHN, Jiří, KAHL, Patrick, LAKEYEV, Anatoly. *Computational complexity and feasibility of data processing and interval computations.* Springer, Dordrecht, Netherlands, 1998. ISBN 1475727933.

MOORE, Ramon E., KEARFOTT, R. Baker, CLOUD, Michael J. *Introduction to Interval Analysis.* SIAM, Philadelphia, PA, 2009. ISBN 0898716691.

NEUMAIER, Arnold. *Interval Methods for Systems of Equations*, volume **37** of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, Cambridge, 1990. ISBN 9780521102148.

NEUMAIER, Arnold, SHCHERBINA, Oleg. Safe bounds in linear and mixed-integer linear programming. *Mathematical Programming*, **99**(2):283–296, 2004. ISSN 0025-5610.

OETTLI, Werner, PRAGER, William. Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides. *Numerische Mathematik*, **6**(1):405–409, 1964. ISSN 0029-599X.

POLJAK, Svatopluk, ROHN, Jiří. Checking robust nonsingularity is NP-hard. *Mathematics of Control, Signals and Systems*, **6**(1):1–9, 1993. ISSN 0932-4194.

ROHN, Jiří. Systems of linear interval equations. *Linear algebra and its applications*, **126**:39–78, 1989. ISSN 0024-3795.

ROHN, Jiří. Enclosing solutions of linear interval equations is NP-hard. *Computing*, **53**(3-4):365–368, 1994. ISSN 0010-485X.

ROHN, Jiří. Solvability of systems of linear interval equations. *SIAM journal on matrix analysis and applications*, **25**(1):237–245, 2003. ISSN 0895-4798.

ROHN, Jiří. Versoft: verification software in MATLAB/INTLAB, 2009.

RUMP, S.M. INTLAB - INTerval LABoratory. In CSENDES, Tibor, editor, *Developments in Reliable Computing*, s. 77–104. Kluwer Academic Publishers, Dordrecht, 1999. Dostupné z: `http://www.ti3.tuhh.de/rump/`.

Seznam použitých zkratek

LP: Lineární programování
VSDP : Verified SemiDefinite Programming

Seznam tabulek

6.1 Optimalizace odhadem přípustného řešení 20
6.2 Optimalizace vynecháním programů na základě mezí 21
6.3 Optimalizace ořezáváním množiny přípustných řešení LP 21

7.1 Výsledky testování pro $R := 0{,}00001$ 23
7.2 Výsledky testování pro $R := 0{,}1$ 23
7.3 Výsledky testování pro úlohy velikosti 2^n a $R := 0{,}5$ 23
7.4 Doba výpočtu v 1 ortantu 24

8.1 Význam hodnot parametru flag 28

9.1 Parametry funkce mxTrie 32

Seznam obrázků

3.1 Množina řešení příkladu 3.1 8
3.2 Množina řešení příkladu 3.1 s vyznačenými osami souřadnic . . . 10