

Funkce QPmR pro výpočet kořenů kvazi-polynomu v jazyku C++

Bc. Michal Andresík



ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: Bc. Michal Andresík
Osobní číslo: A22265
Studijní program: N0613A140022 Informační technologie
Specializace: Softwarové inženýrství
Forma studia: Kombinovaná
Téma práce: Funkce QPmR pro výpočet kořenů kvazi-polynomu v jazyku C++
Téma práce anglicky: QPmR Function for Calculating the roots of a Quasi-Polynomial in C++

Zásady pro vypracování

- Popište stavový a vstupně-výstupní model lineárních spojitých dynamických systémů se zpožděním.
- Definujte pojem charakteristického kvazi-polynomu a uveďte vlastnosti spektra různých typů systémů se zpožděním.
- Podrobně popište algoritmy QPmR a QPmR v.2 pro numerický výpočet kořenů kvazi-polynomu.
- Naprogramujte oba výše uvedené algoritmy v jazyku C++ a doplňte je podrobnou dokumentací zdrojového kódu.
- Ověřte funkčnost a správnost vytvořených programů na několika úlohách.

Seznam doporučené literatury:

1. MICHIELS, Wim a VYHLÍDAL, Tomáš. *An eigenvalue based approach for the stabilization of linear time-delay systems of neutral type*. Online. Automatica. 2005, roč. 41, č. 6, s. 991-998. ISSN 00051098. Dostupné z: <https://doi.org/10.1016/j.automatica.2004.11.032>. [cit. 2024-11-06].
2. PEKAŘ, Libor. *On a quasioptimal spectrum assignment for delayed systems*. Online. In: Control Strategy for Time-Delay Systems. Elsevier, 2021, s. 137-187. ISBN 9780128205990. Dostupné z: <https://doi.org/10.1016/B978-0-12-820599-0.00009-4>. [cit. 2024-11-06].
3. VYHLÍDAL, Tomáš. *Analysis and Synthesis of Time Delay System Spectrum*. Praha, 2003. Disertační práce. České vysoké učení technické v Praze, Fakulta strojní.
4. VYHLÍDAL, Tomáš a ZÍTEK, Pavel. *Quasipolynomial mapping based rootfinder for analysis of time delay systems*. Online. IFAC Proceedings Volumes. 2003, roč. 36, č. 19, s. 227-232. ISSN 14746670. Dostupné z: [https://doi.org/10.1016/S1474-6670\(17\)33330-X](https://doi.org/10.1016/S1474-6670(17)33330-X). [cit. 2024-11-06].
5. VYHLÍDAL, Tomáš a ZÍTEK, Pavel. *Mapping Based Algorithm for Large-Scale Computation of Quasi-Polynomial Zeros*. Online. IEEE Transactions on Automatic Control. 2009, roč. 54, č. 1, s. 171-177. ISSN 0018-9286. Dostupné z: <https://doi.org/10.1109/TAC.2008.2008345>. [cit. 2024-11-06].
6. VYHLÍDAL, Tomáš a ZÍTEK, Pavel. *QPmR – Quasi-Polynomial Root-Finder: Algorithm Update and Examples*. Online. In: VYHLÍDAL, Tomáš; LAFAY, Jean-François a SIPAHI, Rifat (ed.). Delay Systems. Advances in Delays and Dynamics. Cham: Springer International Publishing, 2014, s. 299-312. ISBN 978-3-319-01694-8. Dostupné z: https://doi.org/10.1007/978-3-319-01695-5_22. [cit. 2024-11-06].
7. LIPPMAN, Stanley B., Josée LAJOIE a Barbara E. MOO. *C++ Primer*. 5th ed. Addison-Wesley, 2012. ISBN 9780321714114.

Vedoucí diplomové práce: **doc. Ing. Libor Pekař, Ph.D.**
Ústav automatizace a řídicí techniky

Datum zadání diplomové práce: **27. října 2024**
Termín odevzdání diplomové práce: **2. června 2025**



doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

PROHLÁŠENÍ AUTORA DIPLOMOVÉ PRÁCE

Beru na vědomí, že

- odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení diplomové práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – diplomovou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky diplomové práce využít ke komerčním účelům;
- pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na diplomové práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato diplomová práce se zaměřuje na analýzu a výpočet kořenů kvazi-polynomů, které jsou klíčové pro posouzení stability lineárních dynamických systémů se zpožděním. V práci jsou popsány modely pro popis těchto systémů – stavový a vstupně-výstupní model. Je odvozen přechod mezi těmito modely a pozornost je zaměřena na charakteristický kvazi-polynom, jeho vlastnosti a stabilitu. Jsou odvozeny algoritmy pro výpočet kořenů kvazi-polynomu QPmR a QPmR v2. Existující implementace těchto algoritmů v matlabu jsou popsány a je vytvořena C++ aplikace s jejich implementací s cílem optimalizace výpočetního času. Výpočetní čas a dosažené výsledky jsou ukázány na příkladech.

Klíčová slova: QPmR, QPmR v2, kvazi-polynomy, stabilita, lineární dynamické systémy se zpožděním, numerické metody v C++

ABSTRACT

This thesis focuses on the analysis and computation of roots of quasi-polynomials, which are crucial for assessing the stability of linear dynamical systems with delays. The thesis describes models used for representing these systems—the state-space and input-output models. The transition between these models is derived, with particular attention given to the characteristic quasi-polynomial, its properties, and stability criteria. Algorithms for computing the roots of quasi-polynomials, QPmR and QPmR v2, are derived. Existing matlab implementations of these algorithms are described, and a C++ application implementing these algorithms has been created to optimize computational time. Computational performance and achieved results are demonstrated through examples.

Keywords: QPmR, QPmR v2, quasi-polynomials, stability, linear time-delay systems, numerical methods in C++

Rád bych poděkoval vedoucímu diplomové práce panu doc. Ing. Liboru Pekařovi, PhD. za ochotu a cenné rady při psaní této diplomové práce.

OBSAH

ÚVOD	13
I TEORETICKÁ ČÁST	15
1 ÚVOD DO DYNAMICKÝCH SYSTÉMŮ	16
1.1 DYNAMICKÉ SYSTÉMY	16
1.1.1 Klasifikace dynamických systémů	16
1.1.2 Příklady dynamických systémů	17
1.2 LINEÁRNÍ DYNAMICKÉ SYSTÉMY	18
1.2.1 Popis lineárních dynamických systémů	18
1.3 SYSTÉMY SE ZPOŽDĚNÍM	19
1.3.1 Rozdělení systémů se zpožděním	19
1.3.2 Příklady použití zpoždění	20
1.4 LAPLACEOVA TRANSFORMACE	20
1.4.1 Definice Laplaceovy transformace	21
1.4.2 Využití v dynamických systémech	21
2 LINEÁRNÍ DYNAMICKÝ SYSTÉM SE ZPOŽDĚNÍM	22
2.1 STAVOVÝ MODEL	22
2.1.1 Matematická formulace	22
2.2 PŘECHOD MEZI STAVOVÝM A VSTUPNĚ-VÝSTUPNÍM MODELEM	23
2.3 VSTUPNĚ-VÝSTUPNÍ MODEL	25
2.3.1 Charakteristická funkce	25
3 CHARAKTERISTICKÝ KVAZI-POLYNOM	27
3.1 MATEMATICKÁ FORMULACE	27
3.2 SPEKTRUM KVAZI-POLYNOMU	28
3.2.1 Esenciální spektrum	28
3.3 VLASTNOSTI SPEKTRA	29
3.3.1 Vlastnosti esenciálního spektra	29
3.3.2 Vlastnosti spektra	30
3.4 STABILITA SYSTÉMU	31

3.4.1	Exponenciální stabilita.....	31
3.4.2	Silná stabilita.....	32
3.4.3	Ukázka vlastností a stability systému.....	32
4	NUMERICKÉ METODY PRO HLEDÁNÍ KOŘENŮ KVAZI-POLYNOMU.....	36
4.1	ITERAČNÍ METODY	36
4.2	ARGUMENTAČNÍ PRINCIP	37
4.3	DALŠÍ NUMERICKÉ METODY	38
4.4	SOFTWAREVÉ NÁSTROJE PRO VÝPOČET SPEKTRA	38
4.4.1	TRACE-DDE	39
4.4.2	DDE-BIFTOOL	39
5	METODA QPMR PRO VÝPOČET KOŘENŮ KVAZI-POLYNOMU	40
5.1	QPMR	40
5.1.1	Běh algoritmu	41
5.2	QPMR v2	44
5.2.1	Inovace původní verze	45
5.2.2	Pseudokód QPmR v2.....	48
5.3	AQPMR	49
5.3.1	Běh programu	50
6	ALGORITMY PRO URČENÍ KONTURNÍCH KŘIVEK	51
6.1	MARCHING SQUARES.....	51
6.2	DISJOINT SET UNION	53
6.3	GRAPH TRAVERSAL	54
7	VYUŽITÍ JAZYKA C++ V NUMERICKÝCH VÝPOČTECH	56
7.1	CHARAKTERISTIKA JAZYKA C++	56
7.2	KNIHOVNY A NÁSTROJE VYUŽITÉ V PRÁCI	57
7.2.1	Standardní knihovna C++ (STL)	57
7.2.2	OpenMP.....	58
7.2.3	Gnuplot.....	59
7.3	VÝVOJOVÉ PROSTŘEDÍ VISUAL STUDIO.....	59

II	PRAKTICKÁ ČÁST	61
8	IMPLEMENTACE V MATLABU	62
8.1	QPMR	62
8.2	QPMR v2	67
9	IMPLEMENTACE V C++	69
9.1	PRÁCE S APLIKACÍ	69
9.2	STRUKTURA APLIKACE	71
9.3	VOLÁNÍ APLIKACE	72
9.4	VÝSTUPY APLIKACE	74
10	PŘÍKLADY	75
10.1	PŘÍKLAD 1	75
10.2	PŘÍKLAD 2	76
10.3	PŘÍKLAD 3	77
	ZÁVĚR	79
	SEZNAM POUŽITÉ LITERATURY	81
	SEZNAM OBRÁZKŮ	87
	SEZNAM TABULEK	89
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK	90
	SEZNAM PŘÍLOH	91

ÚVOD

Tato práce je rozdělena na teoretickou a praktickou část. Teoretická část práce se věnuje popisu systémů, kvazi-polynomů a jejich numerickému řešení. Praktická část je zaměřena na implementaci numerického řešení.

První kapitola poskytuje obecný přehled o dynamických systémech, jejich klasifikaci a základních vlastnostech. Předmětem této práce jsou lineární spojité dynamické systémy se zpožděním. Jednotlivé vlastnosti takových systémů jsou vysvětleny s uvedením příkladů. Na závěr je zavedena Laplaceova transformace, která slouží jako důležitý nástroj pro popis a analýzu těchto systémů.

Druhá kapitola se věnuje matematické formulaci již zmíněných lineárních dynamických systémů se zpožděním. Nejprve je úloha formulována v časové oblasti pomocí stavových, vstupních a výstupních proměnných. Následně je pomocí Laplaceovy transformace proveden detailní přechod na popis vstupně-výstupní, který je definován v komplexní rovině. Nakonec je uveden význam vzniklého popisu pro budoucí analýzu a vlastnosti systému.

Ve třetí kapitole již dochází k přiblížení k praktické části práce. Je zde zaveden pojem charakteristického kvazi-polynomu, jeho spektrum a esenciální spektrum. Jsou zkoumány vlastnosti charakteristického kvazi-polynomu, které jsou dále využity pro posouzení stability systému. Na závěr je uveden ilustrační příklad.

Čtvrtá kapitola shrnuje přehled numerických metod, které lze využít k numerickému hledání kořenů kvazi-polynomu. Podrobněji jsou popsány Newtonova metoda a princip argumentu z důvodu jejich využití v algoritmech QPmR. Ostatní metody jsou zmíněny okrajově. Na závěr je pozornost přesunuta k existujícím softwarovým nástrojům.

V páté kapitole je pozornost věnována algoritmům QPmR. Hlavním předmětem práce jsou verze QPmR a QPmR v2. Ty jsou v této kapitole detailně odvozeny a popsány z důvodu budoucí implementace v praktické části. Nakonec je stručně popsán algoritmus aQPmR.

V šesté kapitole jsou popsány algoritmy potřebné pro výpočet a sestavení konturních křivek, které jsou stěžejní pro funkci algoritmu QPmR. Vlastní implementace těchto algoritmů zajišťuje vyšší výpočetní rychlost.

V sedmé a poslední kapitole teoretické části práce je pozornost zaměřena na nástroje a techniky jazyka C++, jakožto opěrný bod pro implementaci řešení. Pro zvýšení rychlosti

programu je popsána optimalizace a paralelní režie. Dále je popsána knihovna využitá pro zobrazení grafického výstupu nalezeného řešení a na závěr je zmíněno použité vývojové prostředí.

V praktické části práce jsou podrobně popsány algoritmy QPmR a QPmR v2 pro výpočet kořenů kvazi-polynomů v matlabu. Dále je představena C++ aplikace s implementací zmíněných metod za účelem snížení výpočetního času a celková optimalizace. V závěru jsou uvedeny příklady a srovnání metod.

I TEORETICKÁ ČÁST

1 ÚVOD DO DYNAMICKÝCH SYSTÉMŮ

V přírodě a technických oblastech se mnoho dějů vyznačuje svým měnícím se průběhem. Popis a pochopení dějů umožňuje teorie dynamických systémů. Jedná se o matematické modely popisující vývoj veličin v závislosti na čase. Typickými příklady mohou být automobil, který zrychluje, mění se teplota tělesa, tlak v nádrži, koncentrace chemické látky v roztoku nebo napětí a proud v elektrickém obvodu. Jejich porozuměním a analýzou lze získat přehled o fungování celého systému. Při modelování nového systému je možné odhadnout jeho budoucí vývoj a navrhnout efektivní řešení [1].

1.1 Dynamické systémy

Dynamický systém lze obecně popsat pomocí soustavy diferenciálních rovnic

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t), \quad (1.1)$$

kde $\mathbf{x}(t)$ je vektor stavových proměnných v čase t charakterizující aktuální stav systému, $\dot{\mathbf{x}}(t)$ je časová derivace stavového vektoru vyjadřující okamžitou změnu stavových proměnných, $\mathbf{u}(t)$ je vektor vstupních veličin (např. vnější síla působící na systém, příkon tepla apod.) a obecná vektorová funkce $\mathbf{f}(\cdot)$ popisuje dynamické chování celého systému v závislosti na aktuálním stavu, vstupu a čase [2].

1.1.1 Klasifikace dynamických systémů

Dynamické systémy je možné rozdělit podle různých kritérií, která představují matematické vlastnosti, složitost systému a způsob chování. Důraz je kladen na spojitost, linearitu a prvek zpoždění [1].

- **Spojité a diskrétní:** Spojité systémy popisují vývoj plynule v čase pomocí spojitých funkcí (např. rychlost, teplota), zatímco diskrétní systémy jsou definované pouze v určitých časových krocích (např. měření teploty v různých okamžicích) [3].
- **Lineární a nelineární:** V případě lineárních systémů jsou funkce popisující dynamiku systému lineární, díky jednoduchým vztahům mezi vstupy a stavy. U nelineárních systémů jsou tyto funkce zcela obecné [3].

- **Bez zpoždění a se zpožděním:** V systémech bez zpoždění závisí stav systému pouze na aktuálních hodnotách stavových veličin, zatímco u systémů se zpožděním se uvažují i hodnoty z minulých stavů (např. regulační systémy) [1].

1.1.2 Příklady dynamických systémů

Dynamické systémy se objevují v mnoha různých oblastech. Díky obecnému matematickému popisu se s nimi lze setkat nejen v technických oborech, ale také např. v biologii a ekonomii [4].

Mechanické systémy

Typickým příkladem mechanického systému je pohybující se automobil, jehož stav (např. poloha, rychlost) se mění podle působících sil a momentů. Dalším příkladem může být kyvadlo, jehož pohyb je určován silou gravitace a počáteční polohou. Dalším mechanickým systémem mohou být robotická ramena, u kterých se pohyb popisuje pomocí polohy a rychlosti jednotlivých kloubů, na něž působí regulační síly a momenty [4, 5, 6].

Elektrické obvody

Elektrické systémy, jako jsou RC (rezistor, kondenzátor) a RLC (rezistor, cívka, kondenzátor) obvody, se dají modelovat pomocí diferenciálních rovnic popisujících vztah mezi napětím, proudem a časem. Tyto rovnice definují, jak rychle se napětí či proud mění v závislosti na vstupních signálech (např. změně napětí zdroje nebo připojení dalších komponent) [7].

Řízení teploty

Systémy řízení teploty jsou typickým příkladem systému se zpožděním. Pokud je žádoucí např. zvýšení teploty v místnosti, po zapnutí vytápění se bude teplota měnit s určitým časovým odstupem [1, 4, 8].

Biologické modely

Mezi známé biologické systémy patří např. model predátora a kořisti. V tomto modelu se sleduje velikost populací, jako jsou lišky a králíci, které se dynamicky mění. Dalším významným příkladem je model šíření nemocí. Ten předpovídá, jak se bude nemoc v populaci šířit na základě počtu nakažených a nenakažených jedinců a jejich vzájemném kontaktu

[4, 9, 10].

Ekonomické modely

Dynamické modely nacházejí své uplatnění také v ekonomii, kde umožňují modelovat různé ekonomické ukazatele. Příkladem může být vývoj hospodářského růstu země v čase na základě faktorů, jako jsou investice, shromažďování kapitálu, technologický pokrok a pracovní síla [11].

1.2 Lineární dynamické systémy

Lineární dynamické systémy představují v rodině dynamických systémů důležitou kategorií. Charakteristickou vlastností je právě linearita, a to díky splnění principů homogenity a superpozice. Díky těmto principům dochází k výraznému zjednodušení jejich popisu, což umožňuje analýzu chování systému, návrh regulační strategie a předpověď budoucího vývoje. Z tohoto důvodu jsou modely lineárních systémů široce používány v technické praxi a tvoří důležitý matematický rámec pro analýzu a řízení [1, 12].

Základním předpokladem lineárních systémů je, že vztah mezi vstupem a výstupem systému je popsán lineárními funkcemi, tedy takovými, které neobsahují žádné součiny ani jiné nelineární operace mezi stavovými proměnnými nebo vstupy systému. Obecně lze lineární dynamické systémy reprezentovat ve tvaru soustavy lineárních diferenciálních rovnic s konstantními nebo proměnnými koeficienty nebo také maticovou formou stavového popisu systému [3, 12].

1.2.1 Popis lineárních dynamických systémů

Lineární časově-variantní dynamické systémy lze obecně popsat pomocí soustavy lineárních diferenciálních rovnic

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t), \\ \mathbf{y}(t) &= \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t),\end{aligned}\tag{1.2}$$

kde $\mathbf{x}(t)$ je vektor stavových proměnných popisující stav systému v čase t , $\dot{\mathbf{x}}(t)$ je derivace

stavových proměnných podle času, vyjadřující okamžitou změnu stavových proměnných, $\mathbf{u}(t)$ je vstupní vektor, obsahující veličiny působící na systém (např. vnější síly, řídicí signály), $\mathbf{y}(t)$ je výstupní vektor, reprezentující sledované veličiny systému (např. poloha, rychlost, napětí) a $\mathbf{A}(t)$, $\mathbf{B}(t)$, $\mathbf{C}(t)$, $\mathbf{D}(t)$ jsou matice příslušných rozměrů obecně závislé na čase, které charakterizují vnitřní dynamiku a propojují vstupy, stavy a výstupy systému [3].

Pokud mají matice konstantní členy a nejsou tak závislé na čase, pak se systém nazývá lineární systém s konstantními parametry [3].

1.3 Systémy se zpožděním

Systémy se zpožděním (anglicky *Time-Delay Systems*, zkráceně TDS) představují významnou skupinu dynamických systémů, u kterých existuje určitá prodleva mezi okamžikem, kdy systém přijme vstupní signál, a okamžikem, kdy se projeví jeho odpovídající odezva. Časové zpoždění se přirozeně vyskytuje v mnoha dynamických procesech a zásadně ovlivňuje jejich chování, vlastnosti, možnosti analýzy i řízení [1, 13].

Časové zpoždění lze definovat jako interval času, během kterého systém nereaguje bezprostředně na změnu vstupu nebo stavu, ale až po uplynutí určité doby. Důvodem zpoždění může být například vzdálenost mezi místem působení signálu a místem jeho projevu, čas potřebný k přenosu informace nebo samotná dynamika reakce systému na vstupní veličinu. Z hlediska matematického modelování se takový systém vyznačuje rovnicemi, ve kterých se objevují hodnoty vstupů nebo stavů z minulých časových okamžiků, tzv. zpožděné argumenty [1, 13].

Výskyt časového zpoždění má také důsledky pro matematickou analýzu. Přítomnost minulých hodnot stavů nebo vstupů způsobuje, že dynamický systém má teoreticky nekonečně dimenzionální charakter, což vede k výraznému nárůstu složitosti analýzy jeho dynamiky a stability [1, 13].

1.3.1 Rozdělení systémů se zpožděním

Systémy se zpožděním lze rozdělit do dvou základních kategorií podle toho, jakým způsobem se projevuje v jejich chování [13, 14].

- **Retardované systémy:** U retardovaných systémů se časové zpoždění vyskytuje pouze ve stavových nebo vstupních veličinách systému. To znamená, že aktuální změna stavu

(derivace stavových proměnných) závisí pouze na současných a minulých hodnotách stavů a vstupů, ale nezávisí už na tom, jak se stavy systému měnily v minulosti [13, 14].

- **Neutrální systémy:** U neutrálních systémů se časové zpoždění objevuje nejen ve stavových nebo vstupních veličinách systému, ale také přímo ve změně stavu (derivacích stavových proměnných). To znamená, že aktuální změna stavu systému je závislá jak na předchozích hodnotách stavů a vstupů, tak i na tom, jak se stav systému měnil v minulosti. [13, 14].

Rozdíl mezi retardovanými a neutrálními systémy je také v jejich složitosti. Systémy retardovaného typu jsou obecně snadněji analyzovatelné. Systémy neutrálního typu mají naopak složitější chování a jejich analýza a řízení je výrazně komplikovanější [13, 14].

1.3.2 Příklady použití zpoždění

Systémy se zpožděním se běžně objevují v mnoha praktických oblastech [4].

- **Řídicí systémy:** Každý regulační systém, ať už jde o regulaci teploty, řízení motorů nebo průmyslových procesů, se setkává s jistým časovým zpožděním. Typicky se jedná o zpoždění způsobené dobou měření, zpracování signálu, přenosu informace nebo reakcí akčních členů (např. topení, motory) [1, 4].
- **Komunikační systémy:** V telekomunikacích a počítačových sítích je zpoždění způsobené časem přenosu dat. Typickým příkladem je dálkové ovládání zařízení přes internet, kde se signály a informace přenášejí s určitou prodlevou, což je nutné brát v úvahu při návrhu a řízení těchto systémů [15].
- **Biologické systémy:** Časové zpoždění se často vyskytuje i v biologických procesech. Například pokud je lidskému tělu podán nějaký lék, existuje prodleva mezi časem podání a pozorovatelnou reakcí organismu. Dalším příkladem je dynamika v populaci, kde existuje časové zpoždění mezi narozením nových jedinců a jejich schopností se dále množit [4].

1.4 Laplaceova transformace

Dynamické systémy jsou popsány diferenciálními rovnicemi a pro analýzu dynamických systémů je potřeba znát jejich řešení. Přímé řešení soustavy diferenciálních rovnic může být

ovšem velmi obtížné, zejména u složitějších systémů, jako jsou právě systémy se zpožděním. Efektivním nástrojem, který výrazně zjednodušuje řešení těchto rovnic, je Laplaceova transformace. Ta umožňuje převést lineární diferenciální rovnice v časové oblasti na algebraické rovnice v tzv. spektrální (frekvenční) oblasti. Výsledné algebraické rovnice jsou pak snadněji řešitelné a umožňují přehlednější analýzu dynamických vlastností, jako je např. stabilita [1, 13, 14].

Zvláštní význam má Laplaceova transformace právě u systémů se zpožděním, protože umožňuje elegantně pracovat s exponenciálními členy reprezentujícími časová zpoždění ve spektrální oblasti, čímž značně usnadňuje manipulaci a analýzu těchto systémů [1, 13, 14].

1.4.1 Definice Laplaceovy transformace

Laplaceova transformace je integrální transformace definovaná vztahem

$$F(s) = \mathcal{L}\{f(t)\} = \int_0^{\infty} f(t)e^{-st} dt, \quad (1.3)$$

kde $f(t)$ je po částech spojitá funkce závislá na čase t (např. stavová veličina, vstupní nebo výstupní signál) definovaná pro $t \geq 0$, $F(s)$ je její obraz v komplexní rovině s , která se nazývá spektrální rovina. Proměnná s je komplexní číslo ve tvaru $a + bi$, kde a a b jsou reálná čísla a i je imaginární jednotka [16].

1.4.2 Využití v dynamických systémech

Laplaceova transformace hraje v analýze dynamických systémů podstatnou roli, protože umožňuje přechod z původního popisu systému v časové oblasti do přehlednější reprezentace v oblasti spektrální (frekvenční). V praxi se dynamické systémy často popisují pomocí stavového modelu, který tvoří soustava lineárních diferenciálních rovnic. Použitím Laplaceovy transformace je možné tento časově závislý model převést na model nezávislý na čase, vyjádřený pomocí algebraických rovnic [13, 14].

Výsledkem této transformace je tzv. přenosová funkce $G(s)$, která představuje vztah mezi vstupem a výstupem systému v komplexní rovině. Takto vzniklý model je označován jako lineární časově invariantní (anglicky *Linear Time-Invariant*, zkráceně LTI) [13, 14].

2 LINEÁRNÍ DYNAMICKÝ SYSTÉM SE ZPOŽDĚNÍM

Lineární dynamické systémy se zpožděním představují významnou část dynamických systémů, které se často uplatňují v praxi. Je tomu díky jednoduchosti lineárních vztahů a schopnosti realisticky zachytit časová zpoždění, která se v reálných procesech nacházejí. Tato kombinace umožňuje efektivně analyzovat a řídit široké spektrum technických systémů, od regulačních obvodů až po komunikační nebo biologické systémy [1, 4].

2.1 Stavový model

Stavový model představuje výhodnou reprezentaci těchto systémů, protože zachycuje vnitřní dynamiku systému pomocí stavových proměnných. Stavový popis musí zachycovat nejen současné hodnoty stavových proměnných a vstupů, ale v důsledku časových prodlev také jejich minulé hodnoty [1, 4].

Stavový model lineárních dynamických systémů se zpožděním lze reprezentovat dvěma maticovými rovnicemi. První je rovnice stavová. Ta popisuje vnitřní dynamiku systému. Patří do skupiny funkcionálních diferenciálních rovnic (anglicky *functional differential equations*, zkráceně FDE), protože obsahuje nejen derivace stavových proměnných, ale také jejich zpožděné hodnoty. Druhou rovnicí je rovnice výstupní, která definuje vztah mezi stavovými proměnnými, vstupy a výstupy systému [13].

2.1.1 Matematická formulace

Stavový model lineárních dynamických systémů se zpožděním je reprezentován soustavou lineárních diferenciálních rovnic

$$\begin{aligned} \dot{\mathbf{x}}(t) - \sum_{i=1}^{n_H} \mathbf{H}_i \dot{\mathbf{x}}(t - \vartheta_{H,i}) - \int_0^T \mathbf{H}_d(\tau) \dot{\mathbf{x}}(t - \tau) d\tau \\ = \sum_{i=0}^{n_A} \mathbf{A}_i \mathbf{x}(t - \vartheta_{A,i}) + \sum_{i=0}^{n_B} \mathbf{B}_i \mathbf{u}(t - \vartheta_{B,i}) + \int_0^T [\mathbf{A}_d(\tau) \mathbf{x}(t - \tau) + \mathbf{B}_d(\tau) \mathbf{u}(t - \tau)] d\tau, \\ \mathbf{y}(t) = \sum_{i=0}^{n_C} \mathbf{C}_i \mathbf{x}(t - \vartheta_{C,i}) + \int_0^T \mathbf{C}_d(\tau) \mathbf{x}(t - \tau) d\tau, \end{aligned} \quad (2.1)$$

kde $\mathbf{x}(t)$ je vektor stavových proměnných systému, $\dot{\mathbf{x}}(t)$ je derivace vektoru stavových proměnných vyjadřující okamžitou změnu stavu, $\mathbf{u}(t)$ představuje vektor vstupních veličin a

$\mathbf{y}(t)$ představuje vektor výstupních veličin systému. $\mathbf{H}_i$, $\mathbf{H}_d(\tau)$, $\mathbf{A}_i$, $\mathbf{A}_d(\tau)$, $\mathbf{B}_i$, $\mathbf{B}_d(\tau)$, $\mathbf{C}_i$, $\mathbf{C}_d(\tau)$ jsou matice příslušných dimenzí, které určují vazby mezi současnými a zpožděnými hodnotami stavů a vstupů. Konstantní časová zpoždění $\vartheta_{H,i}$, $\vartheta_{A,i}$, $\vartheta_{B,i}$ definují přesné okamžiky v minulosti, kdy hodnoty stavů a vstupů ovlivňují aktuální dynamiku systému, platí $0 = \vartheta_{.,0} < \vartheta_{.,1} < \dots < \vartheta_{.,n} = T$. Integrované členy s distribuovanými zpožděními představují situace, kdy minulé hodnoty systému ovlivňují stav systému nejen v určitých okamžicích, ale spojitě a plynule od aktuálního okamžiku zpětně po dobu T , platí $0 \leq \tau \leq T$ [13].

Systém se nazývá retardovaný, pokud se u členů derivace stavové proměnné zpoždění nevyskytuje, a platí tedy $\forall \mathbf{H}_i = \mathbf{0}$ a zároveň $\mathbf{H}_d = \mathbf{0}$. Naopak, pokud existuje alespoň jeden člen, kde má derivace stavové proměnné zpoždění, systém se nazývá neutrální a platí $\exists \mathbf{H}_i = \mathbf{0}$ nebo $\mathbf{H}_d = \mathbf{0}$. $\mathbf{0}$ v těchto rovnostech označuje nulovou matici [13, 14].

2.2 Přejchod mezi stavovým a vstupně-výstupním modelem

Na rovnici (2.1) je aplikována Laplaceova transformace s nulovými počátečními podmínkami (hodnoty veličin jsou nulové pro $t < 0$). Tím dochází k transformaci časově závislých členů na algebraické výrazy ve spektrální oblasti s komplexní proměnnou s . Pro přehlednost je vypsána aplikace Laplaceovy transformace na všechny podoby členů této rovnice [1, 13].

Pro veličiny a jejich derivace v čase t platí

$$\mathcal{L}\{\mathbf{x}(t)\} = \mathbf{X}(s), \quad \mathcal{L}\{\dot{\mathbf{x}}(t)\} = s\mathbf{X}(s). \quad (2.2)$$

Pro veličiny v časech s konstantními zpožděními platí

$$\mathcal{L}\left\{\sum_{i=0}^n \mathbf{A}_i \mathbf{x}(t - \vartheta_i)\right\} = \mathbf{X}(s) \sum_{i=0}^n \mathbf{A}_i e^{-s\vartheta_i}. \quad (2.3)$$

Pro derivace veličin v časech s konstantními zpožděními platí

$$\mathcal{L}\left\{\sum_{i=0}^n \mathbf{A}_i \dot{\mathbf{x}}(t - \vartheta_i)\right\} = s\mathbf{X}(s) \sum_{i=0}^n \mathbf{A}_i e^{-s\vartheta_i}. \quad (2.4)$$

Pro veličiny s distribuovanými zpožděními platí

$$\mathcal{L} \left\{ \int_0^T \mathbf{A}_d(\tau) \mathbf{x}(t - \tau) d\tau \right\} = \int_0^T \mathbf{A}_d(\tau) e^{-s\tau} d\tau \mathbf{X}(s). \quad (2.5)$$

S využitím převodních vztahů (2.2) – (2.5) a následným vytknutím je možné soustavu (2.1) psát ve tvaru

$$(\mathbf{I} - \mathbf{H}(s))s\mathbf{X}(s) = \mathbf{A}(s)\mathbf{X}(s) + \mathbf{B}(s)\mathbf{U}(s),$$

$$\mathbf{Y}(s) = \mathbf{C}(s)\mathbf{X}(s), \quad (2.6)$$

kde $\mathbf{I}$ je jednotková matice a členy po vytknutí jsou dány

$$\mathbf{H}(s) = \sum_{i=1}^{n_H} \mathbf{H}_i e^{-s\vartheta_{H,i}} + \int_0^T \mathbf{H}_d(\tau) e^{-s\tau} d\tau,$$

$$\mathbf{A}(s) = \sum_{i=0}^{n_A} \mathbf{A}_i e^{-s\vartheta_{A,i}} + \int_0^T \mathbf{A}_d(\tau) e^{-s\tau} d\tau,$$

$$\mathbf{B}(s) = \sum_{i=0}^{n_B} \mathbf{B}_i e^{-s\vartheta_{B,i}} + \int_0^T \mathbf{B}_d(\tau) e^{-s\tau} d\tau,$$

$$\mathbf{C}(s) = \sum_{i=0}^{n_C} \mathbf{C}_i e^{-s\vartheta_{C,i}} + \int_0^T \mathbf{C}_d(\tau) e^{-s\tau} d\tau. \quad (2.7)$$

Převedením členu $\mathbf{A}(s)\mathbf{X}(s)$ v první rovnici a následným vytknutím přechází soustava (2.6) do tvaru

$$(s(\mathbf{I} - \mathbf{H}(s)) - \mathbf{A}(s))\mathbf{X}(s) = \mathbf{B}(s)\mathbf{U}(s),$$

$$\mathbf{Y}(s) = \mathbf{C}(s)\mathbf{X}(s). \quad (2.8)$$

Vynásobením první rovnice (2.8) maticí $(s(\mathbf{I} - \mathbf{H}(s)) - \mathbf{A}(s))^{-1}$ zleva a dosazením vyjádřeného $\mathbf{X}(s)$ do druhé rovnice vzniká

$$\mathbf{Y}(s) = \mathbf{C}(s) (s(\mathbf{I} - \mathbf{H}(s)) - \mathbf{A}(s))^{-1} \mathbf{B}(s) \mathbf{U}(s). \quad (2.9)$$

Rozepsáním inverzní matice $(s(\mathbf{I} - \mathbf{H}(s)) - \mathbf{A}(s))^{-1}$ podle definice je obdržen finální tvar

$$\mathbf{Y}(s) = \mathbf{C}(s) \frac{\text{adj}(s(\mathbf{I} - \mathbf{H}(s)) - \mathbf{A}(s))}{\det(s(\mathbf{I} - \mathbf{H}(s)) - \mathbf{A}(s))} \mathbf{B}(s) \mathbf{U}(s) \quad [4, 13]. \quad (2.10)$$

2.3 Vstupně-výstupní model

Vstupně-výstupní model explicitně vyjadřuje vztah mezi vstupními a výstupními veličinami systému. Tento přístup je velmi užitečný pro analýzu a návrh řízení systémů ve spektrální oblasti [13].

Vztah mezi vektorem vstupních veličin (signálů) $\mathbf{U}(s)$ a vektorem výstupních veličin (signálů) $\mathbf{Y}(s)$ lze vyjádřit pomocí přenosové funkce $\mathbf{G}(s)$ danou

$$\mathbf{Y}(s) = \mathbf{G}(s) \mathbf{U}(s) \quad [1, 13, 17]. \quad (2.11)$$

Přenosová funkce $\mathbf{G}(s)$ je maticí příslušných rozměrů podle počtu vstupních a výstupních veličin. Z rovnice (2.10) je zřejmé vyjádření této matice podle tvaru (2.11), platí

$$\mathbf{G}(s) = \mathbf{C}(s) \frac{\text{adj}(s(\mathbf{I} - \mathbf{H}(s)) - \mathbf{A}(s))}{\det(s(\mathbf{I} - \mathbf{H}(s)) - \mathbf{A}(s))} \mathbf{B}(s) \quad [13]. \quad (2.12)$$

2.3.1 Charakteristická funkce

Charakteristická funkce systému, označovaná $\Delta(s)$, je definována jako determinant matice ve jmenovateli přenosové funkce

$$\Delta(s) = \det(s(\mathbf{I} - \mathbf{H}(s)) - \mathbf{A}(s)) \quad [4, 13]. \quad (2.13)$$

Charakteristická funkce je zásadní pro analýzu stability a dynamických vlastností systému, protože její kořeny odpovídají právě pólům systému [4, 13].

Obecně může mít charakteristická funkce několik forem. V nejjednodušším případě, tj. u systému bez zpoždění, má podobu klasického polynomu s komplexní proměnnou s . U systémů, kde se objevují pouze konstantní zpoždění, má charakteristická funkce formu kvazi-polynomu. U systémů, kde existuje alespoň jedno distribuované zpoždění, se charakteristická funkce označuje termínem celá funkce (anglicky *entire function*). Tato celá funkce má podobu kvazi-polynomiálního zlomku, kde kořeny charakteristické funkce odpovídají kořenům kvazi-polynomu v čitateli. Společné kořeny čitatele a jmenovatele se algebraicky ruší. Řešením kvazi-polynomiální rovnice $\Delta(s) = 0$ jsou pak právě póly systému [4, 13].

3 CHARAKTERISTICKÝ KVAZI-POLYNOM

Výsledkem Laplaceovy transformace stavového popisu lineárních dynamických systémů se zpožděním je vztah mezi vstupy a výstupy tohoto systému. Ten je ve formě přenosové funkce, která má ve jmenovateli charakteristickou funkci. Kořeny charakteristické funkce určují tzv. póly systému. Póly popisují spektrální vlastnosti systému a rozhodují o jeho stabilitě a dynamickém chování [13, 14].

U těchto systémů vede řešení charakteristické funkce k řešení kvazi-polynomiální rovnice, která dále přechází na úlohu řešení kořenů kvazi-polynomu. Jelikož kořeny tohoto kvazi-polynomu (póly systému) charakterizují chování celého systému, nazývá se kvazi-polynom charakteristický. Ten na rozdíl od běžného polynomu obsahuje exponenciální členy s proměnnou s v exponentu, které vznikají v důsledku časových zpoždění ve stavových nebo vstupních veličinách. Charakteristický kvazi-polynom je tedy důležitým nástrojem pro analýzu spektra systémů se zpožděním a jeho studium je zásadní pro pochopení stability a návrh řízení těchto systémů [13, 14].

3.1 Matematická formulace

Charakteristický kvazi-polynom je funkce komplexní proměnné s daná předpisem

$$q(s) = \sum_{k=0}^N p_k(s) e^{-s\alpha_k}, \quad (3.1)$$

kde $p_k(s)$ jsou polynomy s reálnými koeficienty, $0 = \alpha_0 < \alpha_1 < \dots < \alpha_{N-1} < \alpha_N$ jsou jednotlivá zpoždění systému a členy $e^{-s\alpha_k}$ reprezentují vliv zpoždění ve spektrální oblasti. $p_k(s)$ jsou polynomy stupně m_k ve tvaru

$$p_k(s) = \sum_{l=0}^{m_k} p_{k,l} s^l, \quad (3.2)$$

kde $p_{k,l}$ jsou jednotlivé reálné koeficienty mocnin proměnné s a nejvyšší stupeň polynomu je n u členů p_N . [18].

3.2 Spektrum kvazi-polynomu

Charakteristické hodnoty, nebo také póly či vlastní čísla systému, tvoří tzv. spektrum systému označované jako Σ . Je definováno jako množina komplexních čísel $s \in \mathbb{C}$, pro která platí

$$\Sigma := \{s \in \mathbb{C} : \Delta(s) = 0\} \quad [13]. \quad (3.3)$$

Na rozdíl od klasických systémů bez zpoždění, jejichž spektrum je konečné (díky kořenům klasického polynomu), je spektrum systémů se zpožděním typicky nekonečné. Přítomnost exponenciálních členů $e^{-s\alpha_k}$ v charakteristickém kvazi-polynomu způsobuje, že tato funkce je transcendentní a její kořeny se často seskupují do tzv. řetězců pólů [13].

U retardovaných systémů, kde se zpoždění vyskytuje pouze ve stavech, ale už ne v jejich derivacích, má spektrum specifickou strukturu. Kořeny kvazi-polynomu zde tvoří nekonečnou množinu, která se v případě stabilního systému nachází vlevo od imaginární osy. V takovém případě jsou všechny póly izolované a rozprostřené do tzv. retardačních řetězců, které se s rostoucí imaginární částí blíží asymptotickým přímkám [13].

U neutrálních systémů je situace složitější. Navíc se zde vyskytuje tzv. esenciální spektrum. To v komplexní rovině tvoří vertikální pás, ke kterému se izolované póly s rostoucí imaginární částí asymptoticky přibližují [13].

3.2.1 Esenciální spektrum

Pro definici esenciálního spektra je nejprve zaveden tzv. přidružený exponenciální polynom $d_n(s)$. Ten odpovídá součtu členů u nejvyšší mocniny charakteristického kvazi-polynomu (3.1). Index n představuje právě nejvyšší mocninu u členů s^n a tento polynom lze psát ve tvaru

$$d_n(s) = \sum_{k=0}^{\bar{N}} d_k e^{-s\bar{\alpha}_k}, \quad (3.4)$$

kde d_k jsou odpovídající členy a $0 = \bar{\alpha}_0 < \bar{\alpha}_1 < \dots < \bar{\alpha}_{\bar{N}-1} < \bar{\alpha}_{\bar{N}}$ jsou odpovídající zpoždění v charakteristickém kvazi-polynomu. Esenciální spektrum Σ_{ess} je pak definováno jako množina nul přidruženého exponenciálního polynomu $d_n(s)$

$$\Sigma_{\text{ess}} := \{s \in \mathbb{C} : d_n(s) = 0\} \quad [13, 18]. \quad (3.5)$$

Tato část spektra má zásadní význam pro stabilitu neutrálních systémů. Pokud existuje jen a pouze jeden prvek esenciálního spektra ležící v pravé polorovině komplexní roviny, je systém automaticky nestabilní bez ohledu na polohu ostatních pólů. Dále je známo, že esenciální spektrum je citlivé na malé změny parametrů, zejména na změny časových zpoždění [13, 18, 19].

Z hlediska maticového popisu jsou póly systému shodné s vlastními čísly matice $s\mathbf{H}(s) + \mathbf{A}(s)$. Esenciální spektrum odpovídá vlastním číslům matice $\mathbf{H}(s)$ a lze jej získat řešením rovnice

$$\det(\mathbf{I} - \mathbf{H}(s)) = 0 \quad [13]. \quad (3.6)$$

3.3 Vlastnosti spektra

Spektrum kvazi-polynomu je oproti klasickým (polynomiálním) systémům jedinečné svou speciální strukturou. Ta plyne ze samotné struktury kvazi-polynomu, ve které se vyskytují exponenciální členy $e^{-s\alpha_k}$ reprezentující časová zpoždění [13].

3.3.1 Vlastnosti esenciálního spektra

Esenciální spektrum Σ_{ess} , které se vyskytuje pouze u neutrálních systémů, má oproti klasickému spektru několik charakteristik, které mají zásadní význam pro stabilitu systému [13].

1. Esenciální spektrum vytváří řetězce rovnoběžné s imaginární osou. S rostoucí vzdáleností jeho prvků od počátku roste jejich imaginární část, zatímco reálná část zůstává v konečném intervalu. [13, 20].
2. Hodnota α_{ess} se nazývá abscisa esenciálního spektra a je definovaná jako horní mez reálných částí prvků esenciálního spektra. Platí pro ni

$$\alpha_{\text{ess}} := \sup \{\operatorname{Re}(s) : s \in \Sigma_{\text{ess}}\}. \quad (3.7)$$

Tato hodnota je velmi citlivá na změny zpoždění v systému. I velmi malé změny zpoždění mohou způsobit skokovou změnu její hodnoty. Proto se zdůrazňuje, že jde o funkci závislou na zpoždění, tj. $\alpha_{\text{ess}} = \alpha_{\text{ess}}(\alpha)$, kde α představuje vektor složený ze všech zpoždění systému [13, 18, 19].

Naopak, změny parametrů, které nezahrnují zpoždění, ovlivňují hodnotu α_{ess} spojitě. Termín spojitost zde znamená, že při infinitesimální změně těchto parametrů nedochází k náhlým skokům této hodnoty. Vůči zpoždění je α_{ess} nespojitá. [13, 18, 19].

3. Pro posouzení chování systému při malých změnách zpoždění slouží bezpečná horní mez $\bar{c}_{\text{ess}}$. Nejprve je provedeno rozšíření abscisy esenciálního spektra $\alpha_{\text{ess}}(\alpha)$, které se značí $\bar{\alpha}_{\text{ess}}(\alpha)$. Je definována funkce

$$\alpha_{\varepsilon}(\alpha) = \sup \{ \alpha_{\text{ess}}(\alpha + \delta\alpha) : \forall \|\delta\alpha\| < \varepsilon, \varepsilon > 0 \} \quad (3.8)$$

a $\bar{\alpha}_{\text{ess}}$ je pak dána

$$\bar{\alpha}_{\text{ess}}(\alpha) = \lim_{\varepsilon \rightarrow 0+} \alpha_{\varepsilon}(\alpha) . \quad (3.9)$$

Díky (3.8) a (3.9) platí nerovnost $\bar{\alpha}_{\text{ess}}(\alpha) \geq \alpha_{\varepsilon}(\alpha)$. Horní mez $\bar{c}_{\text{ess}}$ funkce $\bar{\alpha}_{\text{ess}}(\alpha)$ je dána jako $\bar{c}_{\text{ess}} = c \in \mathbb{R}$, které odpovídá maximální řešení rovnice

$$\sum_{k=1}^{\bar{N}} \left| \frac{d_k}{d_0} \right| e^{-c\bar{\alpha}_k} = 1 \quad [13, 18, 19]. \quad (3.10)$$

3.3.2 Vlastnosti spektra

Spektrum Σ pro systémy se zpožděním vykazuje několik důležitých vlastností [13, 20].

1. Pokud je $\Delta(s)$ kvazi-polynom, nikoliv klasický polynom, pak je spektrum nekonečná množina [13, 20].

2. U retardovaných systémů jsou všechny póly spektra izolované. Pro jakékoliv reálné a konečné $\beta < 0$ má komplexní polorovina $\text{Re}(s) > \beta$ konečně mnoho pólů, zatímco ve zbylé polorovině $\text{Re}(s) < \beta$ jich leží nekonečně mnoho [13, 20].
3. U neutrálních systémů se mohou části spektra sdružovat do vertikálních pásů. Pro tyto sdružené póly platí, že se s rostoucí velikostí přibližují pólům esenciálního spektra, tj. spektrum Σ asymptoticky konverguje k esenciálnímu spektru Σ_{ess} . V podmnožině $\mathbb{C}_{\beta_1 \leq \text{Re}(s) \leq \beta_2} \subset \mathbb{C}$ pro konečná $\beta_1, \beta_2 \in \mathbb{R}$ může ležet nekonečně mnoho pólů. V důsledku je možné nalézt nekonečně mnoho pólů také v polorovině $\text{Re}(s) > 0$ [13, 20].
4. Pro podmnožinu pólů spektra, ve které jsou velikosti pólů rostoucí, tj. $|s_{k-1}| < |s_k|$, $s_k \in \Sigma_{\text{sub}} \subset \Sigma$, platí $\lim_{k \rightarrow \infty} \text{Im}(s_k) = \infty$ pro $\lim_{k \rightarrow \infty} s_k = \alpha_{\text{ess}}$. V komplexní polorovině $\text{Re}(s) > \bar{c}_{\text{ess}} \geq \bar{\alpha}_{\text{ess}}$ je konečný počet izolovaných pólů [13, 20].
5. Samostatné izolované póly se chovají hladce a spojitě vůči změnám zpoždění a parametrů bez zpoždění [13, 20].
6. Spektrální abscisa α definovaná $\alpha := \sup \{\text{Re}(s) : s \in \Sigma\}$ je nehladká, nekonvexní a není lipschitzovská, zejména pro neutrální systémy [13, 20].

3.4 Stabilita systému

Stabilita dynamického systému představuje klíčovou vlastnost, která rozhoduje o jeho praktické použitelnosti. Stabilita je spojena s polohou pólů v komplexní rovině. Pro účely této práce jsou zmíněny dva druhy stability, exponenciální a silná [13].

3.4.1 Exponenciální stabilita

Exponenciální stabilita lineárního systému se zpožděním je úzce spjata s polohou pólů v komplexní rovině. Znamená, že pokud dojde k vychýlení systému z jeho klidového, rovnovážného stavu, systém se do tohoto stavu opět vrátí, a sice exponenciální rychlostí díky exponenciálním členům. Exponenciální stabilita je definována pomocí spektrální abscisy α [13, 21].

Retardované systémy jsou exponenciálně stabilní, pokud spektrální abscisa leží v levé otevřené části komplexní roviny, tj. $\alpha < 0$ [13, 20].

U neutrálních systémů je podmínka silnější. Ty jsou exponenciálně stabilní, pokud existuje $\varepsilon > 0$ takové, že platí $\alpha < -\varepsilon$ [13, 20].

3.4.2 Silná stabilita

Neutrální systémy jsou velmi citlivé i na malé změny zpoždění. Z tohoto důvodu u nich byl zaveden pojem silné stability. Ten říká, že chování systému z hlediska vlastností a stability se při malých změnách zpoždění nezmění. Je to tedy schopnost esenciálního spektra zůstat v levé části komplexní roviny. [13, 19].

Systém je silně stabilní, právě tehdy, když platí

$$\bar{\alpha}_{\text{ess}} < 0. \quad (3.11)$$

Jelikož je $\bar{\alpha}_{\text{ess}}$ shora omezena bezpečnou horní mezí $\bar{c}_{\text{ess}}$, systém je silně stabilní, pokud platí

$$\bar{c}_{\text{ess}} < 0. \quad (3.12)$$

Navíc lze ještě silnou stabilitu definovat pomocí koeficientů přidruženého exponenciálního polynomu $d_n(s)$. Systém je silně stabilní, pokud platí

$$\sum_{k=1}^{\bar{N}} \left| \frac{d_k}{d_0} \right| < 1 \quad [13, 18, 19]. \quad (3.13)$$

3.4.3 Ukázka vlastností a stability systému

Jako příklad je uvedeno zadání se stavovým popisem. Z něj je ihned získána charakteristická funkce jako charakteristický kvazi-polynom a k němu odpovídající přidružený exponenciální polynom. Jsou zvolena zpoždění a systém je vyšetřen. Pro ukázkou je dosaženo změny chování systému pomocí malé změny zpoždění [19].

Zadání a charakteristiká funkce

Uvažuje se neutrální systém s jednou stavovou proměnnou $x(t)$ daný

$$\dot{x}(t) - \frac{3}{4}\dot{x}(t - \vartheta_1) + \frac{1}{2}\dot{x}(t - \vartheta_2) = \frac{1}{4}x(t) + \frac{3}{4}x(t - \vartheta_1) . \quad (3.14)$$

Převodem popisu na odpovídající tvar (2.6)

$$\dot{x}(t) - \left(\frac{3}{4}\dot{x}(t - \vartheta_1) - \frac{1}{2}\dot{x}(t - \vartheta_2) \right) = \frac{1}{4}x(t) + \frac{3}{4}x(t - \vartheta_1) \quad (3.15)$$

je možné s využitím převodních vztahů (2.7) jednoduše určit charakteristickou funkci (2.13), která je ve formě kvazi-polynomu a je rovna

$$\Delta(s) = s \left(1 + \frac{3}{4}e^{-s\vartheta_1} - \frac{1}{2}e^{-s\vartheta_2} \right) - \frac{3}{4}e^{-s\vartheta_1} - \frac{1}{4} . \quad (3.16)$$

Odpovídající přidružený exponenciální polynom (3.4) je dán

$$d_n(s) = 1 + \frac{3}{4}e^{-s\vartheta_1} - \frac{1}{2}e^{-s\vartheta_2} . \quad (3.17)$$

Pro systém jsou zvolena zpoždění $\boldsymbol{\vartheta} = (\vartheta_1, \vartheta_2) = (1, 2)$ [19].

Vlastnosti a stabilita

Díky rovnici (3.17) je možné pomocí podmínky (3.13) ihned určit, že systém není silně stabilní, protože platí

$$\sum_{k=1}^{\bar{N}=2} \left| \frac{d_k}{1} \right| = \left| \frac{3}{4} \right| + \left| -\frac{1}{2} \right| = \frac{5}{4} > 1 . \quad (3.18)$$

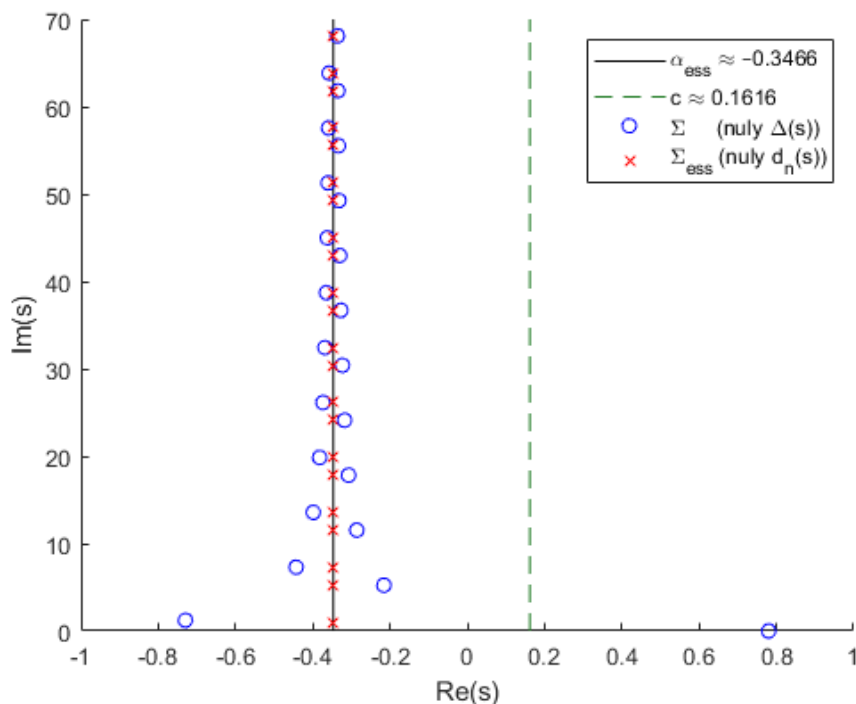
Dále je možné z vlastností esenciálního spektra (3.10) určit bezpečnou horní mez $\bar{c}_{\text{ess}}$ jako řešení rovnice

$$\sum_{k=1}^{\bar{N}} \left| \frac{d_k}{d_0} \right| e^{-c\bar{\alpha}_k} = \frac{3}{4}e^{-c} + \frac{1}{2}e^{-2c} = 1 \quad \implies \quad \bar{c}_{\text{ess}} \approx 0.1616 . \quad (3.19)$$

Z této hodnoty je také možné určit, že systém není silně stabilní, jelikož neplatí $\bar{c}_{\text{ess}} < 0$ [19].

Vykreslení polů systému je zobrazeno na obrázku 3.1. Nuly charakteristického polynomu (spektrum) jsou vykresleny modře, nuly přidruženého exponenciálního polynomu (esenciální spektrum) červeně, hodnota abscisy esenciálního spektra černě a hranice bezpečné horní meze zeleně.

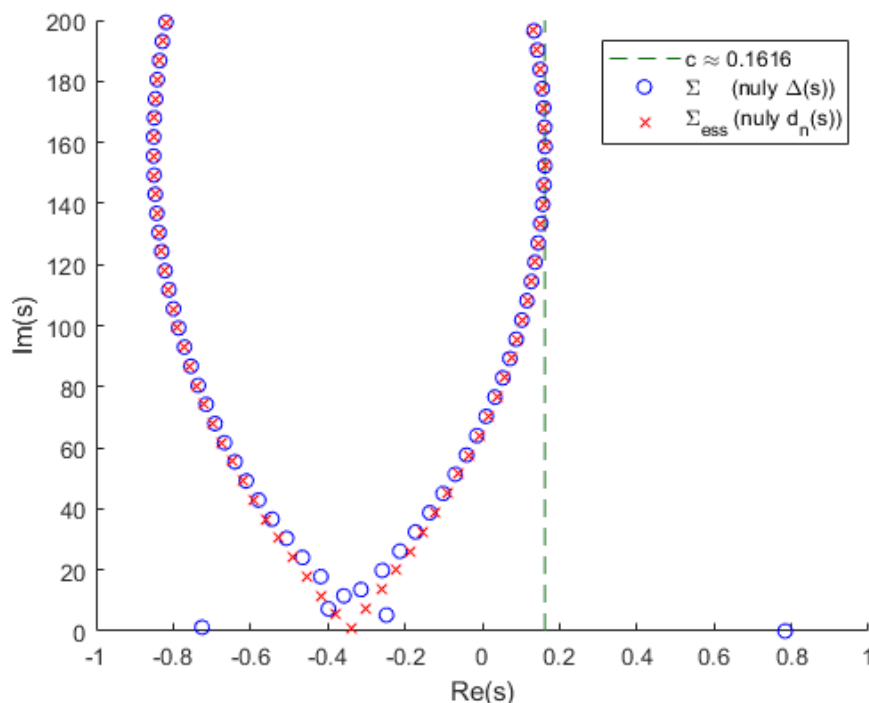
Z grafu je patrné, že systém není exponenciálně stabilní, protože existuje pól v pravé části komplexní roviny a pro spektrální abscisu neplatí $\alpha < -\varepsilon$ pro $\varepsilon > 0$. Opačná situace ovšem nastává pro abscisu esenciálního spektra, $\alpha_{\text{ess}} \approx -0.3466$ a kořeny přidruženého exponenciálního polynomu jsou exponenciálně stabilní. Tento fakt ilustruje, jak póly spektra asymptoticky konvergují k prvkům esenciálního spektra. Další viditelnou vlastností je, že napravo od hranice bezpečné horní meze je konečný počet izolovaných polů [13, 19].



Obrázek 3.1 Ukázka chování systému se zpožděním $\vartheta = (1, 2)$

(zdroj: vlastní podle [19])

Pomocí změny zpoždění na hodnotu $\vartheta = (0.99, 2)$ je zobrazena nespojitost abscisy esenciálního spektra vůči změně zpoždění, která ovlivňuje chování celého systému.



Obrázek 3.2 Ukázka chování systému se zpožděním $\vartheta = (0.99, 2)$

(zdroj: vlastní podle [19])

Přidružený exponenciální polynom je s volbou zpoždění $\vartheta = (1, 2)$ exponenciálně stabilní. Na obrázku 3.2 je zobrazeno chování stejného systému se změnou zpoždění na $\vartheta = (0.99, 2)$. Tato malá změna prvního zpoždění naprosto změnila hodnotu abscisy esenciálního spektra, která se nyní vyskytuje u bezpečné hranice horní meze. Platí pro ni $\alpha_{\text{ess}} > 0$ a přidružený exponenciální polynom už není exponenciálně stabilní. Tento příklad ilustruje význam silné stability, tedy schopnosti esenciálního spektra zůstat v levé části komplexní roviny [13, 19].

4 NUMERICKÉ METODY PRO HLEDÁNÍ KOŘENŮ KVAZI-POLYNOMU

Charakteristický kvazi-polynom vzniká jako výsledek Laplaceovy transformace systému se zpožděním a jeho kořeny (póly) určují spektrální vlastnosti a stabilitu systému. Na rozdíl od běžných polynomů, které mají konečný počet kořenů, je kvazi-polynom transcendentní funkcí s nekonečně mnoha kořeny [13, 14].

Cílem numerických metod je určit polohy těchto kořenů, alespoň v určité omezené oblasti komplexní roviny. Typicky se jedná o kořeny v pravé části komplexní roviny, protože se zde nacházejí potenciálně nestabilní póly. Výpočet všech kořenů není obecně možný, a proto existuje více přístupů, jak úlohu řešit [14, 17].

4.1 Iterační metody

Iterační metody patří k základním přístupům v hledání kořenů analytických funkcí, kam patří i kvazi-polynomy. Princip spočívá v opakovaném zpřesňování odhadu řešení. Jejich výhodou je nízká výpočetní náročnost. Nevýhodou je úspěšnost konvergence, protože zde hraje klíčovou roli odhad počáteční oblasti, ve které se řešení hledá [22, 23].

Newtonova metoda

Newtonova metoda patří mezi hojně využívané iterační metody. V každé iteraci je aktuální řešení upřesněno pomocí hodnoty funkce a její derivace. Iterace má tvar

$$s_{k+1} = s_k - \frac{q(s_k)}{q'(s_k)}, \quad (4.1)$$

kde s_k je kořen po k -té iteraci, $q(s)$ a $q'(s)$ jsou hodnoty funkce a její derivace v tomto bodě [22].

Newtonova metoda se často využívá v kombinaci s jinými metodami díky své rychlé konvergenci. V algoritmu QPmR slouží ke zpřesnění polohy kořenu po jeho odhadu. Díky předpisu kvazi-polynomu (3.1) lze jednoduše určit předpis pro derivaci a vyčíslit hodnoty kvazi-polynomu a jeho derivace v jakémkoliv bodu [17, 18].

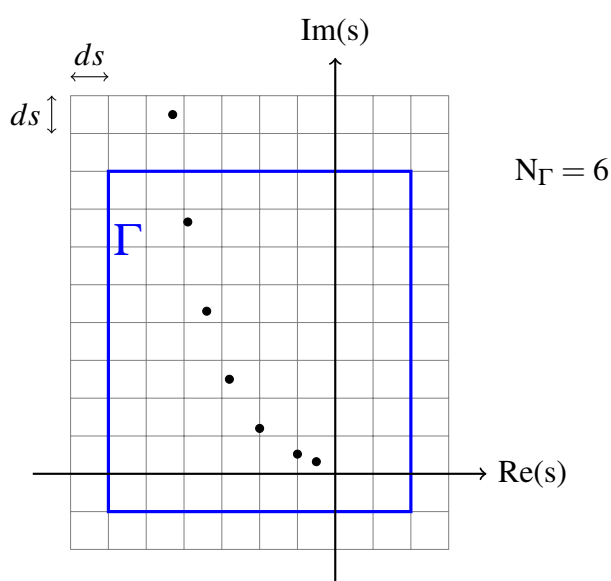
4.2 Argumentační princip

Dalším z důležitých nástrojů při hledání komplexních kořenů je tzv. argumentační princip. Ten slouží ke zjištění počtu nul uvnitř uzavřené oblasti komplexní roviny. V kontextu kvazi-polynomů se většinou používá jako doplňková metoda k ověření, zda byl nalezen správný počet nul [17, 18].

Počet nul N_Γ funkce $q(s)$ uvnitř uzavřené jednoduché křivky Γ je dán

$$N_\Gamma = \frac{1}{2\pi i} \oint_\Gamma \frac{q'(s)}{q(s)} ds \quad [17, 18]. \quad (4.2)$$

Jak již bylo zmíněno u Newtonovy metody, díky předpisu kvazi-polynomu (3.1) lze jednoduše určit předpis pro derivaci a vyčíslit hodnoty kvazi-polynomu a jeho derivace v jakémkoliv bodu. U numerických metod se většinou vyšetřuje obdélníková oblast v komplexní rovině. To výrazně zjednodušuje výpočet krivkového integrálu, protože dochází pouze k vertikálním nebo horizontálním posunům na komplexní mřížce. Tento princip je využit ve druhé verzi algoritmu QPmR. Zde dochází ke zmenšování kroků ds při procházení mřížky, což společně se zadanou přesností ovlivňuje počet nalezených nul. Dále také dochází k dělení oblastí na podoblasti [17, 18].



Obrázek 4.1 Ukázka principu argumentu na čtvercové mřížce

(zdroj: vlastní)

4.3 Další numerické metody

Pro výpočet spektra kvazi-polynomů existuje široká škála numerických přístupů. Mezi dva hlavní přístupy patří metody založené na diskretizaci operátoru a metody využívající charakteristickou matici [18, 23, 24].

Diskretizace operátoru

Tyto metody nahrazují původní matematický popis systému (který má nekonečně mnoho stavů) numerickým modelem, který má konečný počet stavů (konečně dimenzionální diskretizací). Typické metody v této oblasti jsou

- LMS (anglicky *Linear multistep methods*) – využívající diskrétní aproximaci derivací v časových krocích (např. Eulerova metoda),
- Runge–Kutta metody – podobné LMS, ale robustnější,
- pseudospektrální metody – velmi přesné metody, které používají interpolaci a ortogonální báze (např. Chebyshevovy polynomy) [18, 23, 24, 25].

Metody charakteristické matice

Metody této skupiny formulují spektrální úlohu jako problém nalezení vlastních čísel charakteristické matice, která je odvozena z původní diferenciální rovnice se zpožděním. Výsledný problém má podobu determinantové rovnice

$$\Delta(s) = \det(s\mathbf{I} - \mathbf{A}(s)) = 0 \quad [23, 24]. \quad (4.3)$$

4.4 Softwarové nástroje pro výpočet spektra

Kromě obecných numerických metod existují i konkrétní softwarové nástroje, které jsou určeny pro výpočet spektra lineárních systémů se zpožděním. Tyto nástroje využívají kombinace přístupů zmíněných v předchozí části – metody diskretizace operátoru a formulace problému jako determinantové rovnice. Oba zmíněné nástroje, TRACE-DDE a DDE-BIFTOOL, jsou dostupné pro prostředí MATLAB [18, 26, 27, 28].

4.4.1 TRACE-DDE

Nástroj TRACE-DDE (anglicky *Tool for Robust Analysis and Characteristic Equations of Delay Differential Equations*) je určen pro výpočet spektra lineárních zpožděných systémů pomocí tzv. pseudospektrálních metod. Tyto metody diskretizují diferenciální operátor v čase pomocí ortogonálních funkcí, např. Chebyshevových polynomů. Tím je nekonečně dimenzionální systém převeden na konečně dimenzionální a spektrum je získáno jako vlastní čísla vzniklé matice [26].

TRACE-DDE je vhodný pro přesný výpočet dominantních pólů, tj. těch s největší reálnou částí, které jsou rozhodující pro stabilitu systému. Vzhledem k vysoké přesnosti použité metody je výpočetně náročný a kvůli tomu je méně vhodný pro výpočet velkého množství pólů na rozsáhlé oblasti [26].

4.4.2 DDE-BIFTOOL

DDE-BIFTOOL je rozšířený MATLAB toolbox určený pro numerickou bifurkační analýzu systémů se zpožděním. Nástroj umožňuje výpočet spektra systému, určení parametrů ovlivňující stabilitu a další pokročilé analýzy [27, 28].

Spektrum je zde počítáno na základě aproximace řešení systému pomocí LMS metody, která slouží k diskretizaci systému a následnému výpočtu vlastních čísel. Na rozdíl od TRACE-DDE je tento nástroj zaměřen nejen na výpočet spektra, ale i na sledování vývoje spektra v závislosti na parametrech [27, 28].

5 METODA QPMR PRO VÝPOČET KOŘENŮ KVAZI-POLYNOMU

Výpočet kořenů kvazi-polynomu představuje klíčovou úlohu při analýze a řízení systémů se zpožděním. Charakteristický kvazi-polynom má obvykle nekonečně mnoho kořenů rozmístěných v komplexní rovině, což komplikuje jejich přesnou identifikaci. Poloha jeho kořenů, neboli póly systému, určují stabilitu a dynamické chování těchto systémů. Analytické řešení v naprosté většině případů není možné, a proto je nutné pro nalezení kořenů využívat numerické metody [1, 17, 18].

Numerická metoda QPmR (anglicky *Quasi-Polynomial mapping based Rootfinder*) je algoritmus vyvinutý k určení všech kořenů kvazi-polynomu nacházejících se v předem definované oblasti komplexní roviny. Základní myšlenkou je rozdělení hodnot kvazi-polynomu na reálnou a imaginární část a následné hledání jejich průsečíků na nulové hladině [1, 17, 18].

5.1 QPmR

Zadáním úlohy je výpočet kořenů kvazi-polynomu, který je dán ve tvaru

$$q(s) = \sum_{k=0}^N \sum_{l=0}^{m_k} p_{k,l} s^l e^{-s\alpha_k}, \quad (5.1)$$

kde $p_{k,l}$ a $\alpha_k > 0$ jsou reálná čísla a s je komplexní proměnná ve tvaru $s = \beta + i\omega$, přičemž $\beta, \omega \in \mathbb{R}$ a i je imaginární jednotka. Hledání kořenů probíhá na oblasti $\mathbb{D} \subset \mathbb{C}$, pro kterou platí $\beta_{\min} \leq \operatorname{Re}(\mathbb{D}) \leq \beta_{\max}$ a $\omega_{\min} \leq \operatorname{Im}(\mathbb{D}) \leq \omega_{\max}$ [1, 17, 18].

Kvazi-polynom, stejně jako každou funkci komplexní proměnné, lze rozdělit na její reálnou a imaginární část. Pro jednoduchost jsou zavedeny reálné funkce R a I , které odpovídají reálné a imaginární části hodnot kvazi-polynomu v bodech komplexní roviny

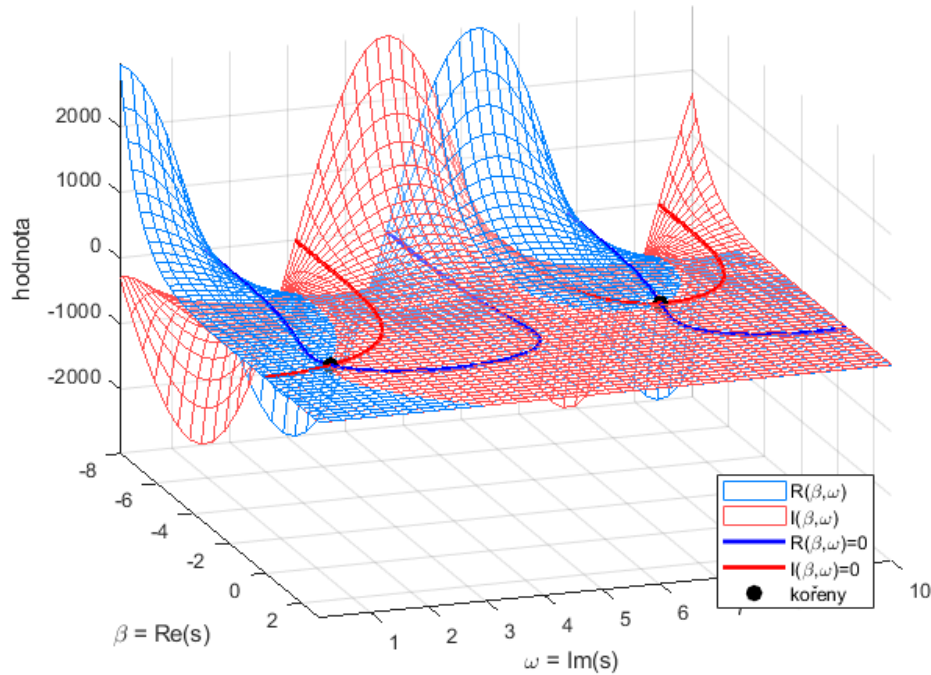
$$\begin{aligned} R(\beta, \omega) &= \operatorname{Re}(q(\beta + i\omega)), \\ I(\beta, \omega) &= \operatorname{Im}(q(\beta + i\omega)). \end{aligned} \quad (5.2)$$

Kořeny kvazi-polynomu $q(s)$ jsou obdrženy řešením rovnice $q(s) = 0$. Aby byla tato rovnice splněna, tj. aby byl kvazi-polynom $q(s)$ roven nule, musí být nulové obě jeho části, platí

$$q(s) = 0 \iff R(\beta, \omega) = I(\beta, \omega) = 0. \quad (5.3)$$

Kořeny jsou pak dány právě jako průsečíky funkcí $R(\beta, \omega)$ a $I(\beta, \omega)$, které splňují danou rovnost (5.3) [1, 17, 18].

Pro názornou ukázkou bude doprovodným příkladem kvazi-polynom $q(s) = s + e^{-s}$. Na obrázku 5.1 jsou zobrazeny funkce odpovídající kvazi-polynomu $q(s)$, a sice $R(\beta, \omega)$ (modře, mřížkovaně), $I(\beta, \omega)$ (červeně, mřížkovaně), $R(\beta, \omega) = 0$ (modrá křivka), $I(\beta, \omega) = 0$ (červená křivka) a černě jsou zobrazeny kořeny jako průsečíky těchto dvou křivek [1, 17, 18].



Obrázek 5.1 Ukázka funkcí $R(\beta, \omega)$, $I(\beta, \omega)$ a jejich nulových konturních křivek

(zdroj: vlastní podle [17])

5.1.1 Běh algoritmu

1. Oblast $\mathbb{D}$ je popsána diskretní mřížkou Γ jako matice komplexních čísel, která má n_{max} řádků, m_{max} sloupců a je formálně zapsána

$$\mathbf{\Gamma} = \begin{bmatrix} \beta_0 + i\omega_0 & \cdots & \beta_0 + i\omega_{m_{\max}} \\ & \ddots & \\ \vdots & \beta_n + i\omega_m & \vdots \\ & & \ddots \\ \beta_{n_{\max}} + i\omega_0 & \cdots & \beta_{n_{\max}} + i\omega_{m_{\max}} \end{bmatrix},$$

$$\beta_n = \beta_{\min} + n\Delta_{\text{grid}}, \quad n = 0, 1, \dots, n_{\max},$$

$$\omega_m = \omega_{\min} + m\Delta_{\text{grid}}, \quad m = 0, 1, \dots, m_{\max}, \quad (5.4)$$

kde Δ_{grid} je velikost kroku ve mřížce.

2. V každém bodu mřížky $\mathbf{\Gamma}$ je vypočtena funkční hodnota kvazi-polynomu $q(s)$ v tomto bodě. Její reálná a imaginární část je rozdělena zvlášť do matic $\mathbf{R}$ a $\mathbf{I}$.

$$\mathbf{R} = \left[R(\beta_n, \omega_m) \right], \quad \mathbf{I} = \left[I(\beta_n, \omega_m) \right]. \quad (5.5)$$

Na obrázku 5.1 tvoří hodnoty těchto matic právě plochy $R(\beta, \omega)$ a $I(\beta, \omega)$ [1, 17, 18].

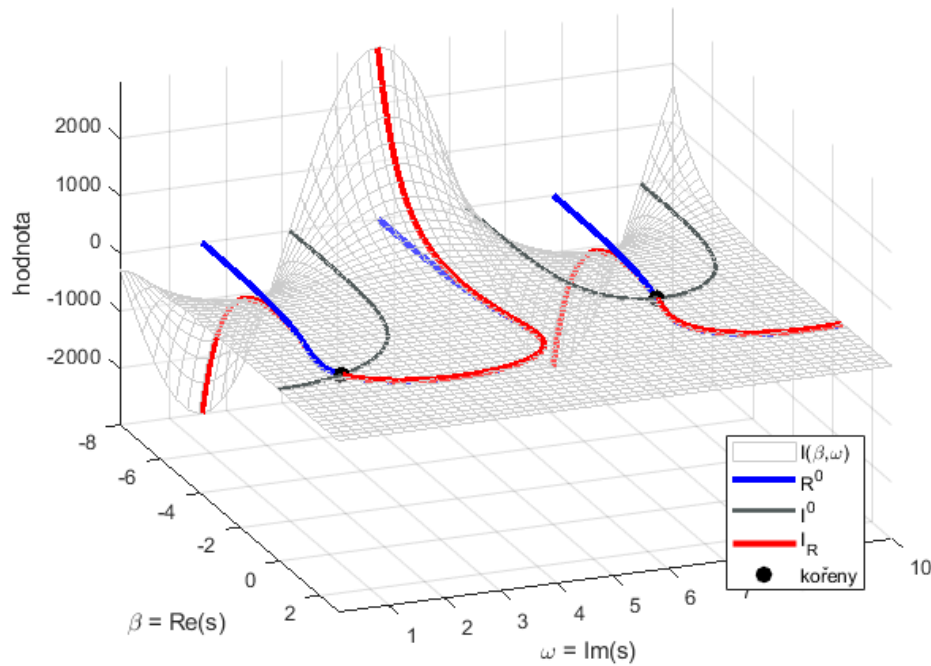
3. Vhodným algoritmem pro výpočet konturních křivek jsou pomocí matic $\mathbf{R}$ a $\mathbf{I}$ vypočteny nové body, které tvoří konturní křivky nulové hladiny. Tyto křivky jsou určeny po sobě jdoucími body v komplexní rovině. Tvoří posloupnosti (popř. vektory) bodů a dále budou značeny $\mathbf{R}^0$ a $\mathbf{I}^0$. Tyto posloupnosti bodů odpovídají numerickému řešení funkcí $R(\beta, \omega) = 0$ a $I(\beta, \omega) = 0$. V matlabu lze provést výpočet konturních křivek pomocí funkce *contour* [1, 17, 18].

V daném příkladu je na obrázku 5.1 vidět, že $\mathbf{R}^0$ obsahují body dvou konturních křivek a stejně je tomu i pro $\mathbf{I}^0$ [1, 17, 18].

4. Dalším cílem je nalézt průsečíky konturních křivek. Myšlenkou je, že při pohybu po ploše hodnot $I(\beta, \omega)$ v bodech $\mathbf{R}^0$ dochází k průchodu nulové hladiny právě v průsečíku konturních křivek. V bodech $\mathbf{R}^0$ jsou tedy vypočteny hodnoty kvazi-polynomu $q(s)$ a z těch se uloží pouze imaginární část, která bude značena $\mathbf{I}_R$. Pro průsečík tedy platí, že hodnota $\mathbf{I}_R$ má v dalším bodě posloupnosti jiné znaménko, a tedy

$$\mathbf{I}_R(r_i)\mathbf{I}_R(r_{i+1}) \leq 0, \quad (5.6)$$

kde r_i a r_{i+1} jsou dva po sobě jdoucí body posloupnosti $\mathbf{I}_R$. Tato skutečnost je ilustrována na obrázku 5.2. Červenou křivkou jsou zobrazeny hodnoty $\mathbf{I}_R$ v bodech konturní křivky $\mathbf{R}^0$. V bodech, kde křivky $\mathbf{I}_R$ a $\mathbf{R}^0$ leží těsně na sobě, je výraznější barva křivky s vyšší hodnotou. Ke změně barev dochází právě v průsečíku, kde se znaménka prohodí. Tento nalezený průsečík slouží jako počáteční odhad kořene kvazi-polynomu $q(s)$ [1, 17, 18].



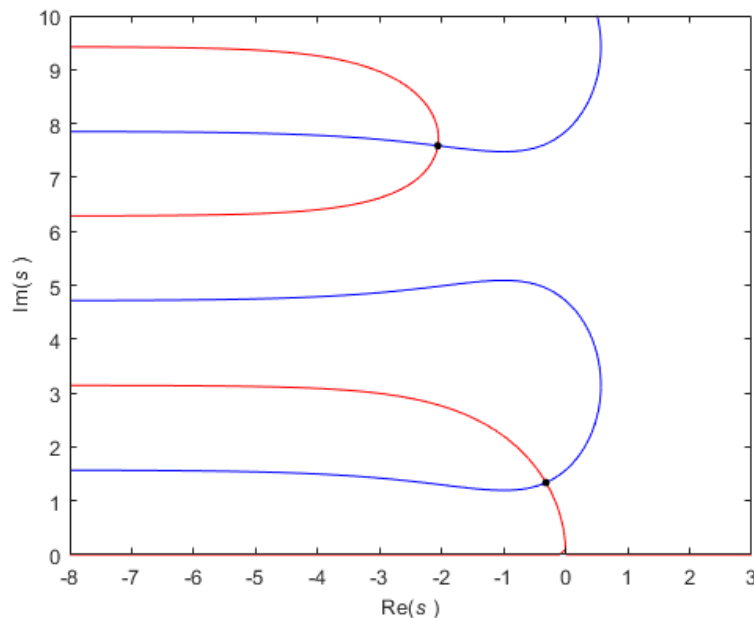
Obrázek 5.2 Ukázka nalezení odhadu kořene v algoritmu QPmR

(zdroj: vlastní podle [17])

- Po nalezení všech průsečíků je využita Newtonova metoda (4.1) pro nalezení přesnějších hodnot kořenů. Jak bylo v této kapitole zmíněno, Newtonova metoda konverguje rychle k přesnému řešení. Za to velmi závisí na zvolené oblasti, ve které se řešení hledá. Pokud je tedy zadán příliš velký krok mřížky Δ_{grid} , počáteční odhad může být dán na široké oblasti a nalezené řešení nemusí být správné [1, 17, 18].

Po ukončení výpočtu je řešení vykresleno. Na obrázku 5.3 je ukázka řešení s využitím me-

tody QPmR v matlabu [29]. Modrou barvou je konturní křivka reálné části kvazi-polynomu, červeně je konturní křivka imaginární části a černě jsou nalezené kořeny [1, 17, 18].



Obrázek 5.3 Ukázka řešení algoritmu QPmR

(zdroj: 29)

5.2 QPmR v2

Pro posouzení chování modelu je často potřeba vyšetření kořenů kvazi-polynomu na větší oblasti, než jaká byla použita v předchozím příkladu. Pokud se použije oblast větších rozměrů, pak je nedostatkem původní verze metody QPmR konstantní velikost kroku mřížky Δ_{grid} na celé oblasti. Ta způsobuje zbytečné výpočetní náklady v oblastech, kde je hustota kořenů menší [18, 29].

Verze 2 algoritmu představuje rozšířenou verzi algoritmu QPmR, ale její jádro zůstává stejné. Známou vlastností kořenů charakteristického kvazi-polynomu je, že se sdružují do asymptotických řetězců. To umožňuje volit menší krok mřížky Δ_{grid} v oblastech, kde se kořenů vyskytuje méně, nebo zde dokonce nejsou žádné [18, 29, 30].

V následujícím textu se větou – V algoritmu QPmR v2 je hodnota x nastavena ... – míní odkaz na nastavení hodnoty x v originální implementaci této metody v matlabu [29].

5.2.1 Inovace původní verze

Mezi hlavní přednosti verze 2 patří adaptace kroku mřížky, posuzování úspěšnosti celého algoritmu pomocí splnění jednotlivých podmínek a finální vyšetření kořenů na konečné podoblasti [18, 29].

Adaptace kroku mřížky

První novinkou je adaptivní velikost kroku mřížky. Ta je ovšem využita pouze v případě, pokud uživatel její hodnotu při volání metody nenastavuje. Pokud je žádoucí vyšetřovat oblast s danou velikostí kroku mřížky, žádná adaptace mřížky prováděna nebude.[18, 29].

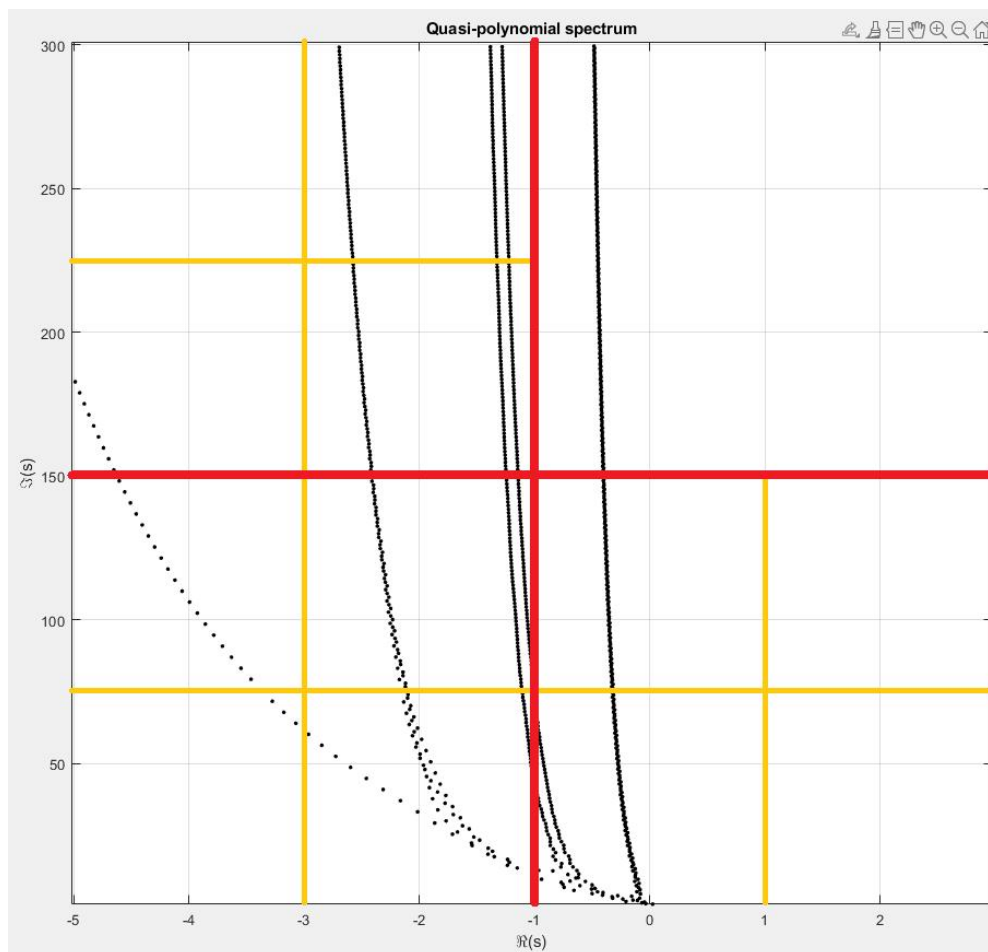
Tento parametr tedy není potřeba zadávat a pro každý běh algoritmu se upravuje. Z velikosti kroku mřížky je také odvozena přesnost. Po spuštění metody se Δ_{grid} společně s přesností e nastaví [18, 29].

V každém běhu je metoda vystavena řadě kontrol, které posuzují správnost výpočtu. Tyto podmínky určují, jestli dojde k upravení hodnoty Δ_{grid} . Pokud nebyl aktuální běh úspěšný, dojde k jejímu zmenšení a metoda se znovu zavolá. Tento krok, tedy zmenšení Δ_{grid} na celé zadané oblasti, může být proveden maximálně dvakrát. V algoritmu QPmR v2 je Δ_{grid} zmenšována na třetinu nebo čtvrtinu své původní hodnoty [18, 29].

Po druhém neúspěšném zmenšení dochází k rozdělení celé (aktuální) oblasti na čtyři stejně velké podoblasti. Zde znovu dochází ke zmenšení velikosti Δ_{grid} a novému volání metody. Tento krok může být proveden také nejvíce dvakrát [18, 29].

Použití metody QPmR v2 a náčrt rozdělení oblastí podle běhu algoritmu je znázorněn na obrázku 5.4. Celá oblast byla po dvou zmenšeních Δ_{grid} dále rozdělena na 4 podoblasti a 3 z těchto podoblastí byly ještě jednou rozděleny a vyšetřeny [18, 29].

Ikdyž se výpočet před zmenšením Δ_{grid} volá znovu na stejnou oblast, tak je kvůli nedostatečně malému kroku mřížky výpočet proveden relativně rychle a výpočetní čas se prodlužuje právě až v hlubším vyšetřování. Na přiloženém obrázku 5.4 je dále vidět, že pravá horní podoblast už nepotřebovala další zmenšení. V pravé spodní (červené) podoblasti jsou obě pravé (žluté) podoblasti bez kořene a podobně je tomu v levé horní (žluté) podoblasti. Celý pás levých (žlutých) podoblastí má mnohem menší hustotu kořenů, než je tomu v pásu $-3 \leq \text{Re}(s) \leq -1$. Adaptivní velikost mřížky a dělení oblastí na podoblasti tedy nachází své uplatnění [18, 29].



Obrázek 5.4 Ukázka rozdělení oblastí u metody QPmR v2

(zdroj: 18, 29)

Podmínky úspěšnosti algoritmu

V každém běhu metody s danou velikostí kroku mřížky dochází k posouzení, zda bylo aktuální nastavení Δ_{grid} dostatečné pro obdržení dostatečně přesných kořenů [18, 29].

První kritérium představuje počet běhů v iterační Newtonově metodě (4.1). Ta slouží k dosažení požadované přesnosti u předem odhadnutých kořenů. Pokud se odhadnutý kořen nachází blízko skutečného kořenu, je s použitím Newtonovy metody zaručena rychlá konvergence. Nastavením dostatečně velké hranice pro opakování je možné posoudit, zda byl nalezen kořen, nebo byl odhad kořene proveden s příliš velkou velikostí kroku mřížky a je tak nesprávný. Ve verzi 2 algoritmu QPmR se nastavuje počet opakování na hodnotu 100 [18, 29].

Druhou podmínkou je princip argumentu (4.2). Algoritmus QPmR slouží k nalezení všech

kořenů na zvolené oblasti. Protože se vyšetřuje několik oblastí různých velikostí, je vhodné zkontrolovat, zda počet nalezených kořenů odpovídá počtu kořenů, jaký se na dané oblasti vyskytuje. Pro zaručení numerické přesnosti se porovnávají počty kořenů dvou oblastí. Pomocí přesnosti ϵ je jedna oblast o malý kus zvětšená a druhá naopak zmenšená. Nalezený počet kořenů pak musí být mezi těmito obdrženyými hodnotami [18, 29].

Poslední podmínkou je vzdálenost mezi odhadem kořene a jeho přesnější formou obdrženu po iteraci Newtonovy metody. Může se stát, že byla Newtonova metoda úspěšná i po volbě větší velikosti kroku mřížky. Jednou z možností může být také mnohonásobnost nalezených kořenů. V takovém případě je potřeba zkontrolovat, jak daleko se přesnější verze kořenu nachází od jeho odhadu. Na čtvercové mřížce by neměly být tyto hodnoty rozdílné více než $\Delta_{\text{grid}}\sqrt{2}$ (velikost úhlopříčky ve čtverci o stranách Δ_{grid}). Jedná se ale o numerickou metodu a dostatečnou podmínkou je rozdíl $2\Delta_{\text{grid}}$ [18, 29].

Pokud dojde k nesplnění jedné z těchto podmínek, úspěšnost aktuálního nastavení je nastavena na hodnotu neúspěchu a dojde buď ke zmenšení Δ_{grid} nebo vrátí aktuální volání tuto hodnotu neúspěchu [18, 29].

Finální kontrola kořenů

V případě, že nalezené kořeny splnily všechny podmínky přesnosti (5.2.1), provede se jejich finální kontrola. Před ukončením algoritmu, tj. po posledním zmenšování velikosti kroku Δ_{grid} , se provede kontrola, zda jsou nalezené hodnoty kořenů správné. V závislosti na nalezených kořenech je nastavena oblast pro další a už poslední zavolání metody QPmR. Společně s oblastí je nastaven také menší krok Δ_{grid} . Oblast a krok mřížky pro volání QPmR se nastaví podle následujících podmínek. [18, 29] .

- Pokud nebyl v této oblasti nalezen žádný kořen, provede se vyšetření na stejné oblasti se zmenšeným Δ_{grid} . Ve verzi 2 algoritmu QPmR je nastaveno trojnásobné zmenšení [18, 29].
- Pokud byl nalezen pouze jeden kořen, nebo bylo nalezeno mnoho kořenů na velmi malé oblasti (vysoká hustota kořenů), oblast je omezena na oblast výskytu (s využitím přesnosti ϵ) a hodnota Δ_{grid} je zmenšena stokrát [18, 29].
- Pokud je nalezen více jak jeden kořen na normální (ne příliš malé) oblasti, tak je nově

vyšetřená oblast nastavena podle poloh kořenů tak, aby v této oblasti všechny kořeny ležely, ale aby se zároveň jednalo o co nejmenší možnou obdelníkovou oblast. Hodnota Δ_{grid} je zmenšena dvakrát [18, 29].

Po nastavení oblasti a velikosti kroku mřížky se ještě jednou zavolá metoda QPmR. Nejprve se porovná počet nalezených kořenů. Pokud byl nalezen další kořen nebo naopak nějaký ubyl, nastaví se neúspěch. Poté se porovnají aktuální kořeny s nově nalezenými, pokud je v nějakých dvou odpovídajících si kořenech velký rozdíl, nastaví se neúspěch. Jelikož byla volaná metoda nastavena s menší hodnotou Δ_{grid} , obdržené kořeny nahradí kořeny současně [18, 29].

5.2.2 Pseudokód QPmR v2

S využitím popisu funkcionalit verze 2 algoritmu QPmR je možné jeho chod zachytit ve formě následujícího pseudokódu. Pseudokód odpovídá nastavení s adaptací mřížky. Účelem je ukázat na jednotlivá volání metody QPmR, není zde zobrazeno např. sjednocování výsledků. Dále, volání QPmR do podoblastí je zapsáno jedním příkazem s přidaným komentářem (řádek 14). Prvotní volání bude ve tvaru QPMR(-1,0,0). lvl_{oblast} a $lvl_{\text{podoblast}}$ označují hloubku rekurzivního volání na celé oblasti, resp. podoblasti.

Algoritmus QPmR v2

```

1: procedure Y=QPMR( $\Delta_{\text{grid}}$ ,  $lvl_{\text{oblast}}$ ,  $lvl_{\text{podoblast}}$ )
2:   Nastav  $\Delta_{\text{grid}}$ ,  $e$ 
3:   Získej konturní křivky a urči odhad kořenů
4:   Zpřesni hodnoty kořenů a proved' jejich kontrolu ( $reduku_{j\Delta} \leftarrow true \vee false$ )
5:   Zde nastav  $Y$  ▷ nastavena návratová hodnota
6:    $finalni\ kontrola \leftarrow reduku_{j\Delta} = false \wedge (\Delta_{\text{grid}} = -1 \vee lvl_{\text{oblast}} \neq 0)$ 
7:   if  $finalni\ kontrola$  then
8:      $Y \leftarrow \text{QPMR}(\Delta_{\text{grid}}/4, 0, 0)$  ▷ úrovně 0,0 záměrně (podm. finální kontroly)
9:   end if
10:  if  $reduku_{j\Delta} \wedge lvl_{\text{oblast}} < 2$  then
11:     $Y \leftarrow \text{QPMR}(\Delta_{\text{grid}}/4, lvl_{\text{AREA}} + 1, 0)$  ▷ voláno na stejnou oblast
12:  end if
13:  if  $reduku_{j\Delta} \wedge lvl_{\text{oblast}} = 2 \wedge lvl_{\text{podoblast}} < 2$  then ▷ běž do podoblastí
14:     $Y \leftarrow \text{QPMR}(\Delta_{\text{grid}}/4, 2, lvl_{\text{SUBAREA}} + 1)$  ▷ voláno 4krát na 4 podoblasti
15:  end if
16: end procedure

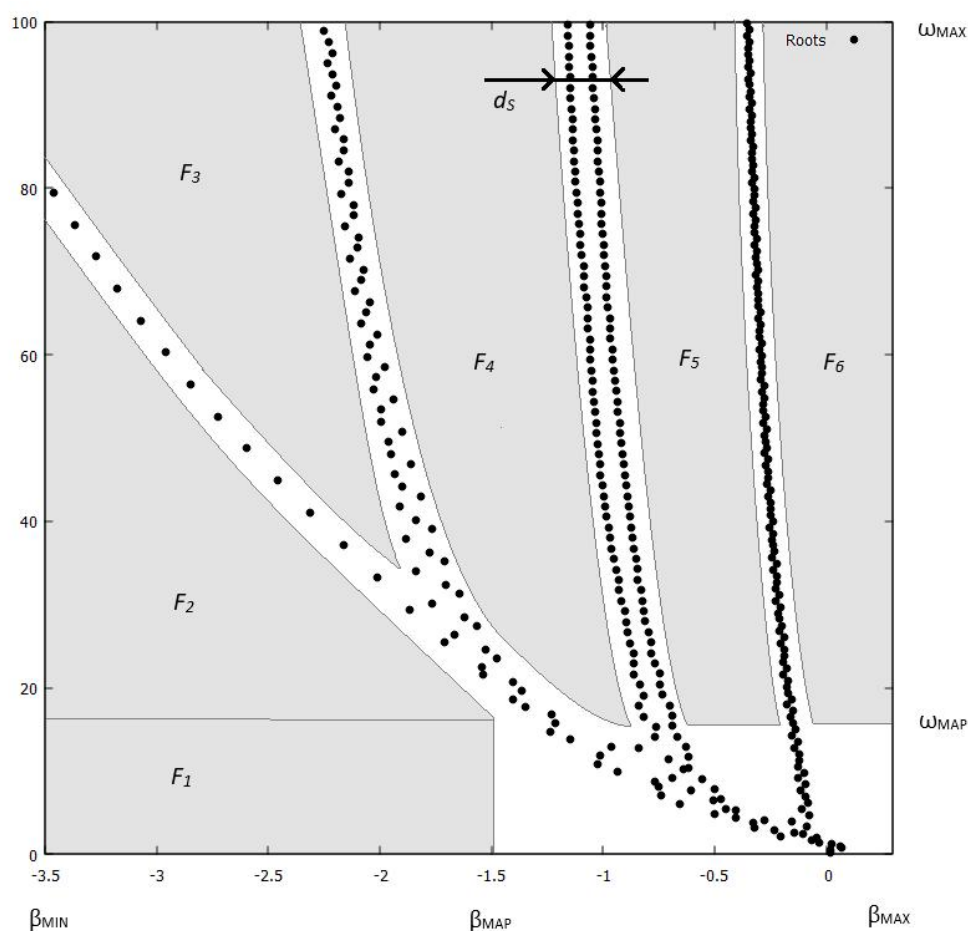
```

5.3 aQPmR

aQPmR (anglicky *advanced QPmR*), je podobně jako QPmR v2, rozšířenou verzí algoritmu QPmR vyvinutou s cílem zvýšení efektivity výpočtu kořenů kvazi-polynomu. Její charakteristikou je kompletní eliminace oblastí, ve kterých se nevyskytují žádné kořeny [30].

Algoritmus využívá vlastnosti struktury rozprostření prvků spektra. Ty se vyskytují v asymptotických exponenciálních řetězcích a oblasti bez výskytu kořenů jsou určeny na základě analýzy asymptotických vlastností spektra kvazi-polynomu [30].

Metoda aQPmR je popsána stručně, jelikož není předmětem této práce. K doprovodu popisu běhu algoritmu slouží obrázek 5.5. Podrobnější popis tohoto algoritmu s uvedením potřebných rovnic je uveden v [30].



Obrázek 5.5 Rozdělení oblasti v algoritmu aQPmR

(zdroj: vlastní podle [30])

5.3.1 Běh programu

1. Nejprve jsou určeny exponenciely S_p , které vytvářejí asymptotické pásy s kořeny o šířce d_S [30].
2. Definují se oblasti F_i , na kterých se s nejvyšší pravděpodobností nenachází žádné kořeny [30].
3. Pro zbylou oblast je zavolán původní algoritmus QPmR (??), tj. jsou namapovány konturní křivky a je obdržén odhad kořenů. Ten je poté zpřesněn Newtonovou iterační metodou [30].

6 ALGORITMY PRO URČENÍ KONTURNÍCH KŘIVEK

Algoritmus pro určení konturních křivek je stěžejní částí algoritmu QPmR. V matlabu je možné pro jejich výpočet využít funkci *contour*. Stejně tak i pro jazyk C++ existují knihovny, např. Gnuplot, které tuto funkci nabízejí. Ikdyž je použití již existující funkce pohodlné a spolehlivé, může být také výhodná vlastní implementace, např. pro testování paralelizace různých částí kódu ve snaze optimalizovat výpočetní čas [17, 31].

Konturní křivky lze implementovat s využitím těchto algoritmů. Algoritmem s pochodujícími čtverci po diskrétní mřížce (anglicky *Marching squares*) jsou určeny jednotlivé úsečky, které tvoří konturní křivky. Algoritmus pro disjunktní sjednocení (anglicky *Disjoint set union*) sjednotí úsečky, které tvoří právě jednu konturní křivku. Pomocí techniky pro práci s grafy a jejich procházením (anglicky *Graph traversal*) je každá konturní křivka sestavena jako posloupnost po sobě jdoucích navazujících bodů [32, 33, 34].

6.1 Marching squares

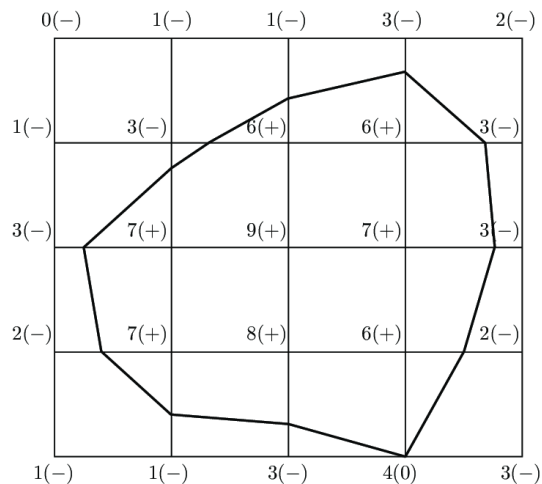
Marching squares je algoritmus využívaný pro nalezení konturních úseček na jednotlivých čtvercích diskrétní mřížky. Postup hledání je znázorněn na obrázku 6.1 [32].

Vrcholům každého čtverce je přiřazena nějaká hodnota. V kontextu této práce je to funkční hodnota kvazi-polynomu v bodu mřížky. Každému vrcholu je přiřazen jeden ze dvou možných stavů. Buď je hodnota větší, než je požadovaná hledaná hladina, nebo je naopak menší nebo rovna. Pomocí těchto stavů spadá každý čtverec právě do jedné ze 16 možných kategorií zobrazených na obrázku 6.2. Na obrázku 6.1 je hledána hladina 4. V každém vrcholu je každý stav znázorněn znaménkem + (resp. -) podle toho, zda se nachází nad (resp. pod) touto hladinou [32, 35].

Pro určení koncového bodu konturní úsečky, který leží na straně čtverce mezi dvěma vrcholy, je využita lineární interpolace

$$x = x_0 + (x_1 - x_0) \frac{(L - y_0)}{(y_1 - y_0)}, \quad (6.1)$$

kde x_0 a x_1 jsou souřadnice vrcholů, y_0 a y_1 jsou hodnoty v těchto vrcholech, $y = L$ je hledaná hladina a x je souřadnice mezi těmito vrcholy [32, 36].

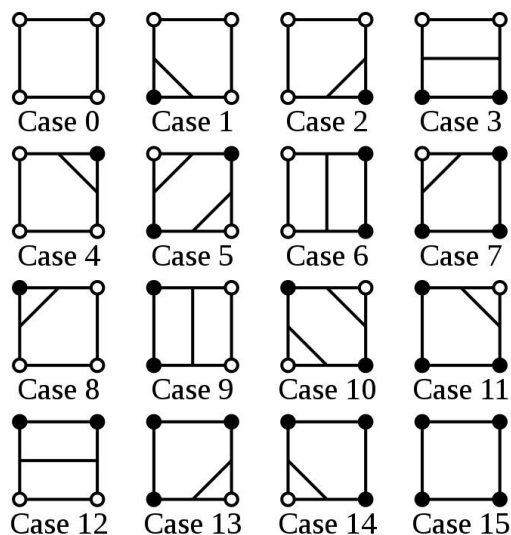


Obrázek 6.1 Hledání konturních křivek pomocí algoritmu Marching squares na hladině 4

(zdroj: 32)

Na obrázku 6.2 jsou jednotlivé stavy ve vrcholech čtverců zobrazeny pomocí vyplněných a prázdných bodů. Ty v každém čtverci určují právě jednu kategorii reprezentující, jak konturní křivka protíná hledanou hladinu [35].

Look-up table contour lines



Obrázek 6.2 Tabulka kategorií čtverců v algoritmu Marching squares

(zdroj: 35)

Výpočty na jednotlivých čtvercích probíhají nezávisle na sobě. To umožňuje paralelní běh algoritmu na jednotlivých čtvercích.

6.2 Disjoint set union

Algoritmus Disjoint set union (zkráceně DSU) vytváří disjunktní množiny, ve kterých jsou všechny prvky vázány nějakým vztahem. Jeho předností je rychlost, jakou dokáže prvky do množin sjednocovat [33].

Algoritmus Marching squares dokáže určit všechny úsečky, ze kterých jsou složeny jednotlivé konturní křivky. Nyní je potřeba sjednotit body do oddělených množin, které tyto křivky reprezentují. Použití DSU je v tomto případě ideální, protože žádné dvě rozdílné křivky neobsahují společný bod.

Z obrázku 6.1 zřejmé, že jsou ve vztahu nejen body spojené úsečkou uvnitř čtverce, ale že jsou spojeny také body sousedních čtverců, protože se jedná o totožný bod.

Algoritmus DSU reprezentuje každou množinu bodů pomocí stromové struktury. Každý prvek má přiřazenou hodnotu svého zástupce. V algoritmu existují dvě základní operace [33].

- Funkce pro nalezení reprezentanta, v angličtině označovaná *find*. V počátku algoritmu je každý prvek sám svým reprezentantem [33].
- Funkce pro sjednocení dvou množin, v angličtině označovaná *union*. Pokud je dán vztah mezi prvky x a y , operací *union* se sjednotí množiny, ve kterých se tyto prvky nachází [33].

Pro provedení operace *union* pro dva různé prvky je nejprve potřeba najít reprezentanty těchto prvků. Z tohoto důvodu existují optimalizační techniky pro tuto operaci. Jednou z nich je sjednocení podle velikosti. Při sjednocování podle velikosti se vždy menší množina stane podmnožinou větší množiny. Reprezentant větší množiny se stane reprezentantem reprezentanta menší množiny. Velikost množiny, kterou reprezentuje prvek větší množiny, se zvětší o velikost menší množiny [33].

Příklad

Pro ukázkou rychlosti algoritmu DSU je vytvořen následující příklad.

- Prvek x leží v množině, která má 1200 prvků a prvek y leží v množině, která má 2500 prvků.

- Reprezentant prvku x je prvek A a reprezentant prvku y je prvek B .
- Množina, ve které je reprezentantem prvek A se stane podmnožinou množiny, kterou reprezentuje prvek B .
- Z hlediska reprezentace dojde k těmto změnám. Prvku A je předepsán reprezentant B . Reprezentantu B je připočtena velikost menší množiny, nyní reprezentuje množinu s 3700 prvky.
- V tomto příkladu bylo ukázáno, že se změnila reprezentace 1200 prvků v jednom kroku.

Optimalizovaná operace *union* dosahuje téměř konstantní časové náročnosti. Časová náročnost této operace se označuje pojmem amortizovaná časová náročnost $O(\alpha(n))$, kde $\alpha(n)$ je inverzní Ackermannova funkce známá svým velmi pomalým růstem [33].

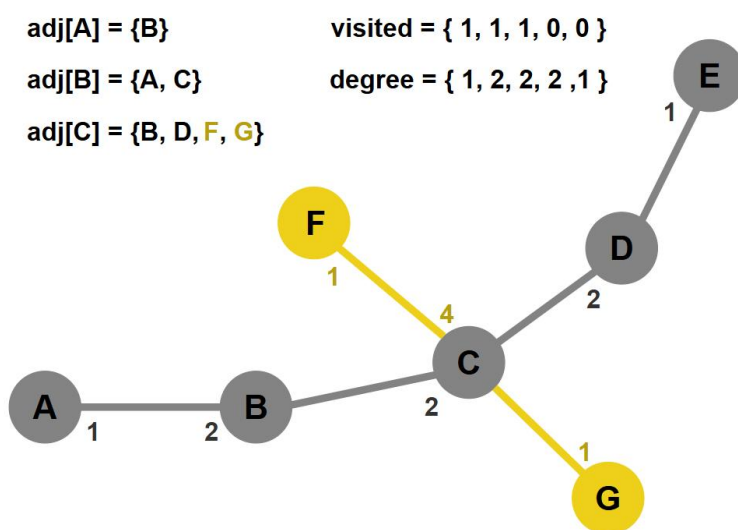
6.3 Graph traversal

Grafem se rozumí uzly (body), které jsou vzájemně propojeny hranami (spojnicemi). Procházení grafu pak označuje skupinu algoritmů, které tyto uzly po jednotlivých hranách procházejí. V souvislosti s konturními křivkami lze jednotlivé body chápat jako uzly. Jejich spojením jsou hrany, které tvoří konturní křivku. Z povahy konturních křivek je situace mnohem jednodušší, než je tomu u obecného grafu. Každý uzel zde může být spojen maximálně dvěma hranami. Díky tomu není potřeba uvažovat algoritmy pro procházení grafu, jako jsou např. DFS, BFS, ale stačí pouze využít techniku, jaká se k pohybu mezi jednotlivými uzly využívá [34].

Procházení grafu je prováděno pomocí množiny sousedů sestavené pro každý uzel (anglicky *adjacency list*, zkráceně *adj*). Dále existuje množina vytvořená pro sledování, zda už byl uzel navštíven (anglicky *visited*). Poslední množinou je množina s počtem sousedů jednotlivých bodů, také nazývaná stupeň (anglicky *degree*). Ta slouží pro konkrétní využití v případech konturních křivek. Cílem je, aby křivka obsahovala po sobě následující body. Pokud existuje uzel se stupněm 1, pak v něm křivka začíná a procházení začne právě v tomto uzlu. Pokud uzel se stupněm 1 neexistuje, tak se jedná o cyklus a na počátečním bodu nezáleží [34].

Na obrázku 6.3 je ukázka grafu s popisem množin sousedů, navštívení a stupně. Žlutou barvou je ukázka případu, který v případě konturních křivek nenastane. Pozice uzlů v množinách *visited* a *degree* odpovídají abecednímu seřazení.

Na příkladu je vidět, že procházení začlo v uzlu *A*, protože je zde hodnota stupně 1. V množině *visited* se nastavila hodnota na 1, protože byl uzel navštíven. V množině *adj(A)* se vyskytuje uzel *B*, který ještě nebyl navštíven. Dalším bodem je tedy bod *B* a průběh se opakuje. Příklad odpovídá stavu po navštívení bodu *C*.



Obrázek 6.3 Ukázka procházení grafu (Graph traversal)

(zdroj: vlastní)

Jednotlivé množiny bodů konturních křivek jsou navzájem disjunktní. Sestavování těchto křivek probíhá nezávisle na sobě, což umožňuje paralelní běh algoritmu na jednotlivých množinách.

7 VYUŽITÍ JAZYKA C++ V NUMERICKÝCH VÝPOČTECH

Volba vhodného programovacího jazyka hraje u výpočetně náročných úloh klíčovou roli. U numerických metod, jako je například metoda hledání kořenů kvazi-polynomu QPmR, je důležité zvolit jazyk, který je efektivní a rychlý [37].

7.1 Charakteristika jazyka C++

Jazyk C++ je rozšířením programovacího jazyka C, který je dlouhodobě využíván v oblastech využívajících vysokou výkonnost. Mezi takové oblasti patří vestavěné systémy, simulace nebo vědecké výpočty. C++ je víceparadigmatický a umožňuje tak kombinovat procedurální, objektově orientované a generické programování. Od svého vzniku se C++ neustále vyvíjí a pravidelně vycházejí nové standardy. Mezi nejvýznamnější patří C++11, C++14, C++17, C++20 a nejnovější C++23. Pro implementaci C++ aplikace v této práci je použit standard C++20 [37, 38, 39].

Velkým přínosem C++ je schopnost snadno integrovat paralelní výpočty, např. pomocí knihovny OpenMP. To umožňuje využít vícejádrové procesory a dosáhnout tak významného urychlení právě u velkého počtu vzájemně nezávislých výpočtů. U numerické metody QPmR se tato paralelizace přímo nabízí, protože se zde počítají hodnoty na diskrétní mřížce, která může být pro dosažení větší přesnosti velmi jemná [40, 41].

Další výhodou C++ je široká škála dostupných knihoven a nástrojů. Ty mohou usnadňovat nejen výpočty, ale také vizualizaci výsledků a jejich analýzu. Po dokončení výpočtu QPmR metody je důležité grafické vykreslení nalezených kořenů. K tomu může být použita např. knihovna Gnuplot [31, 39].

Jedním z významných faktorů je také možnost optimalizace výsledného kódu. C++ jako kompilovaný jazyk umožňuje během procesu kompilace zvolit různé úrovně optimalizace. To dovoluje vytvoření kompromisu mezi výslednou rychlostí a velikostí programu. Například optimalizace O2 vede ke zvýšení výpočetního výkonu aplikace za cenu mírně větší velikosti výsledného kódu [37, 42].

Vzhledem k uvedeným faktorům představuje jazyk C++ vhodnou a efektivní volbu pro implementaci numerických metod a algoritmů, které jsou výpočetně náročné.

7.2 Knihovny a nástroje využité v práci

Pro vývoj numerických metod a algoritmů je důležité mít k dispozici nejen výkonný programovací jazyk, ale také vhodné knihovny a nástroje. Použití kvalitních nástrojů snižuje náročnost implementace a zvyšuje tak efektivitu vývoje. V této kapitole jsou představeny knihovny a nástroje, které byly využity pro implementaci metody QPmR v jazyce C++.

7.2.1 Standardní knihovna C++ (STL)

Standardní knihovna C++ (anglicky *Standard Template Library*, zkráceně STL) představuje základní součást jazyka C++ a poskytuje široké spektrum datových struktur, algoritmů a nástrojů [38, 39].

Níže jsou uvedeny konkrétní části standardní knihovny STL, které byly využity při implementaci aplikace s algoritmem QPmR v této práci.

- `std::vector` – Dynamické pole umožňující snadné a efektivní ukládání a manipulaci s daty. V práci byl použit například pro ukládání hodnot kvazi-polynomu na diskrétní mřížce, ukládání bodů tvořících konturní křivky a v hlavní řadě také k ukládání nalezených kořenů kvazi-polynomu [39].
- `std::complex` – Datový typ pro práci s komplexními čísly, který výrazně zjednodušuje numerické výpočty v komplexní rovině. Tento typ je vhodný pro implementaci algoritmu QPmR z důvodu pohybu po komplexní mřížce a práci s komplexními kořeny. STL knihovna umožňuje pro `std::complex` využití mnoha operací, jako jsou `std::abs` (absolutní hodnota komplexního čísla), `std::exp` (komplexní číslo v exponentu e^x) nebo také `std::pow` (mocnění komplexního čísla).
- `std::pair` – Struktura umožňující uložení dvojice hodnot. Může se využívat pro vrácení dvou hodnot z funkce (např. hodnoty kvazi-polynomu a jeho derivace v jednom bodě) [39].
- `std::string` – Datový typ určený k práci s textovými řetězci. V implementaci je použit například pro správu názvů souborů, sestavování konzolových zpráv aplikace nebo pro práci s výstupy [39].

- Algoritmy z knihovny `<algorithm>` – Standardní algoritmy, například `std::sort`, sloužící k seřazení nalezených kořenů podle jejich reálné či imaginární části, což usnadňuje analýzu výsledků [39].
- Numerické konstanty a funkce ze standardní knihovny `<numbers>` – Konstanty jako `std::numbers::pi`, které jsou využity při numerických výpočtech (např. princip argumentu, tj. počet kořenů v oblasti) [39].
- Měření času pomocí standardní knihovny `<chrono>` – Umožňuje měření doby trvání výpočtů, což je užitečné pro porovnání efektivity jednotlivých implementací či optimalizačních variant algoritmu. Dále je možné porovnávání různých částí kódu pro nalezení kritických částí kódu a následnou optimalizaci [39].
- Formátování výstupu pomocí `std::format` – Funkce umožňující přehledné sestavení textových výstupů, jako jsou zprávy do konzolového okna, popis výsledků výpočtu a popis běhu výpočtu [39].
- `std::ifstream` a `std::ofstream` z knihovny `<fstream>` – Třídy pro čtení a zápis dat do souborů. Tyto nástroje jsou použity pro načítání vstupních parametrů algoritmu, nastavení aplikace a ukládání vypočtených výsledků do textových souborů. To usnadňuje práci se vstupními a výstupními daty algoritmu QPmR a jejich správu [39].

7.2.2 OpenMP

Tato knihovna představuje rozhraní pro paralelní programování se sdílenou pamětí. OpenMP umožňuje snadnou a efektivní paralelizaci výpočtů na vícejádrových procesorech pomocí jednoduchých direktiv a pragmat, které automaticky rozdělí výpočty mezi dostupná procesorová jádra [40, 41].

Ukázka příkazu pro paralelní režii je zobrazena na obrázku 7.1. Zpracování vzájemně nezávislých množin z množiny *groups* je rozděleno podle počtu dostupných jader.

```
#pragma omp parallel for
for (int i = 0; i < groups.size(); i++)
{
```

Obrázek 7.1 Ukázka příkazu knihovny OpenMP

(zdroj: vlastní)

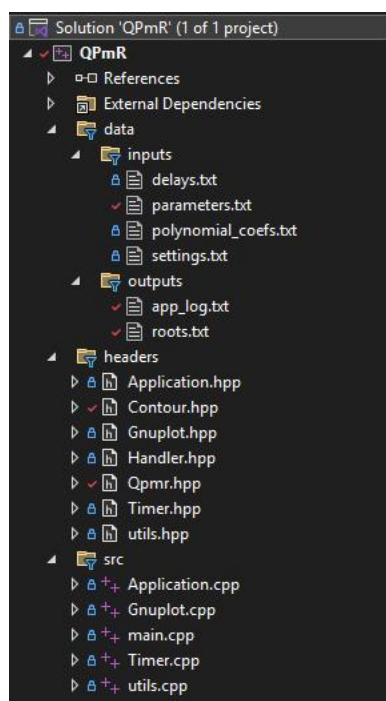
V algoritmu QPmR se výpočty často provádí na rozsáhlých diskretních mřížkách bodů, což představuje ideální příležitost k paralelizaci. Díky využití OpenMP může být dosaženo významného zrychlení výpočtů a efektivnějšího využití dostupného výpočetního výkonu.

7.2.3 Gnuplot

Knihovna Gnuplot je vhodná pro vykreslení dosažených výsledků a k tomu je také v C++ implementaci využita. V první verzi algoritmu QPmR byla tato knihovna použita také k výpočtu konturních křivek. Velkou výhodou této knihovny je jednoduchost použití. Knihovna umožňuje také uložení dosažených grafických výsledků [31].

7.3 Vývojové prostředí Visual Studio

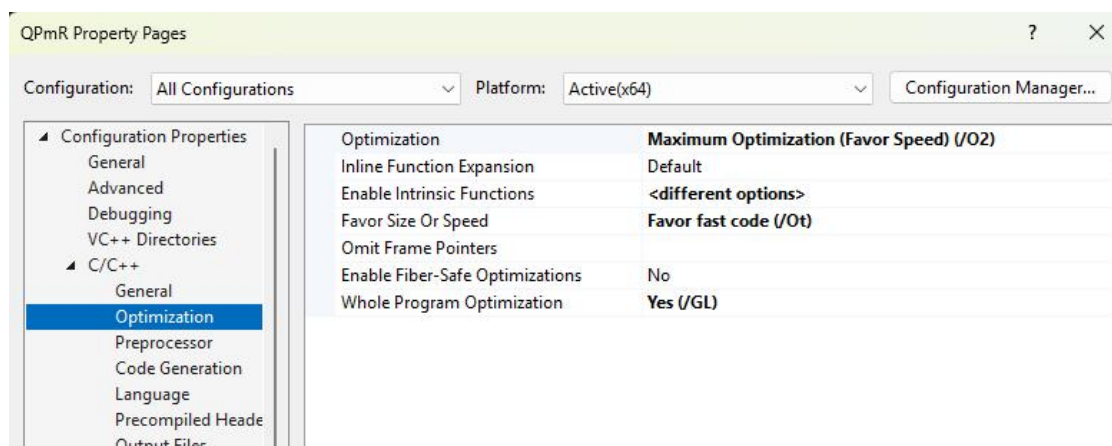
Pro úspěšnou implementaci, ladění a optimalizaci algoritmu QPmR v jazyce C++ bylo vybráno vývojové prostředí Visual Studio. Patří mezi jedny z nejrozšířenějších a nejoblíbenějších integrovaných vývojových prostředí (IDE) pro jazyk C++. Díky své komplexní sadě nástrojů umožňuje efektivní návrh, implementaci, ladění, analýzu výkonu a následnou optimalizaci. Zároveň je ale vhodné také díky své jednoduchosti a intuitivní orientaci v prostředí. Struktura projektu (Solution Explorer) ve Visual Studiu je zobrazena na obrázku 7.2 [43].



Obrázek 7.2 Ukázka Solution Exploreru ve Visual Studiu

(zdroj: vlastní)

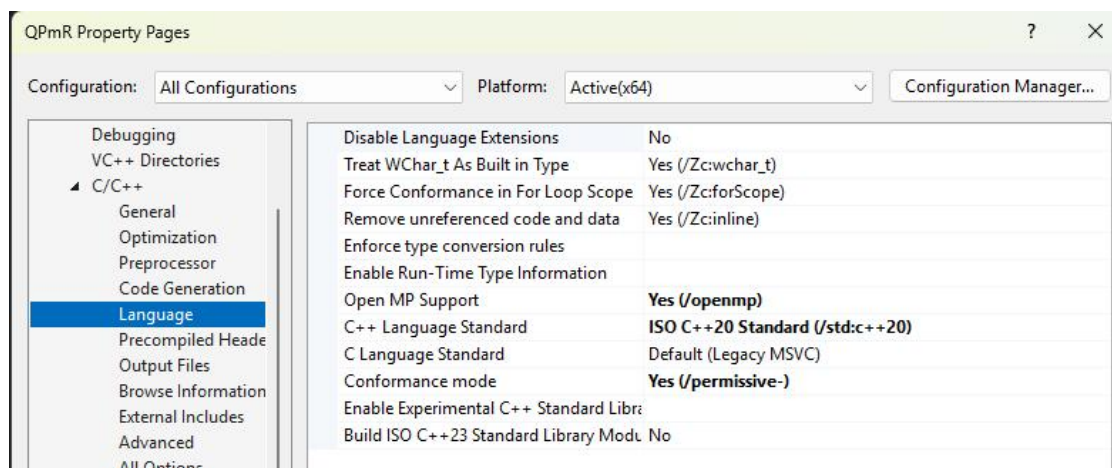
Ve Visual Studiu lze také velmi snadno nastavit dříve zmíněné různé úrovně optimalizace kódu. V rámci této práce byla zvolena úroveň /O2, která maximalizuje rychlost za cenu mírného zvýšení velikosti generovaného kódu. Nastavení optimalizace je zobrazeno na obrázku 7.3 [42].



Obrázek 7.3 Nastavení optimalizace ve Visual Studiu

(zdroj: vlastní)

Podobně snadné je zahrnutí knihovny OpenMP. V prostředí stačí aktivovat podporu podporu OpenMP v nastavení. Nastavení OpenMP je zobrazeno na obrázku 7.4 [44].



Obrázek 7.4 Nastavení podpory OpenMP ve Visual Studiu

(zdroj: vlastní)

II PRAKTICKÁ ČÁST

8 IMPLEMENTACE V MATLABU

Tato kapitola se věnuje podrobnějšímu popisu kódů QPmR a QPmR v2, které byly vyvinuty v matlabu autory těchto metod a je možné je nalézt v [29].

Oba tyto algoritmy jsou popsány pomocí ukázek kódů, které jsou doplněny podrobným komentářem s cílem snadného pochopení jejich funkčnosti.

8.1 QPmR

Algoritmus QPmR, nebo také QPmR v1, tvoří prvotní implementaci algoritmu QPmR. Tato varianta byla následně rozšířena na verze QPmR v2 a aQPmR, které vhodněji pracují s vyšetřovanou oblastí. Výpočet kořenů na dané oblasti, který tvoří výpočetní jádro všech uvedených algoritmů, zůstává stejný. V teoretické části práce je algoritmus popsán (5.1), společně s popisem jeho běhu (5.1.1).

Popis metody

Obrázek 8.1 představuje hlavičku metody QPMR s příkladem zadání. Vstupní kvazi-polynom M se zadává pomocí symbolické proměnné s . Následuje vyšetřovaná oblast Reg , krok mřížky d a požadovaná přesnost kořenů e .

```

1 %QPmR(QP,D,d,e)
2 %   Quasi-Polynomial Mapping based Rootfinder
3 %   Finds all roots of the characteristic quasi-polynomial QP
4 %   of a Time Delay System (TDS) located in the defined region
5 %   of the complex plane D. The algorithm maps the zero-level
6 %   curves of both real and imaginary parts of the
7 %   quasi-polynomial using the Matlab function contour.
8 %   The roots are located as the intersection points of the
9 %   curves. Accuracy of the roots is increased using Newton's
10 %   iteration method.
11 %
12 %   QP - quasi-polynomial in symbolic variable s
13 %   D=[Real_min Real_max Imag_min Imag_max] - region D in the
14 %   complex plane
15 %   d - grid-step of the mesh spread over D for mapping the
16 %   zero-level curves, (d<0.1*pi/tau_max, tau_max is the
17 %   maximum delay in the TDS dynamics)
18 %   eps - required accuracy of the roots
19 %
20 %   Example
21 %   syms s
22 %   QP=s^3+3.1*s+(1.8*s^3+2.3)*exp(-0.31*s)+1.9*s^2*exp(-1.43*s)+0.7*exp(-2.5*s)
23 %   P=qpmr(QP,[-15 10 0 100],0.1*pi/2.5,0.000001);
24 %
25 %   see also AQPMR
26
27
28 function P=QPmR_v1(M,Reg,d,e)
29

```

Obrázek 8.1 Popis algoritmu QPmR v matlabu, část 1

(zdroj: 29)

Na obrázku 8.2 jsou první řádky metody QPmR. Je definována symbolická proměnná s ,

pomocí které je kvazi-polynom M zadán. Hranice oblasti pro výpočet je rozšířena o 3 kroky mřížky d . Pokud je kvazi-polynom M zadán pomocí matice a ne pouze jako jedna funkce, je M převeden na odpovídající polynom s proměnnou s . Do proměnné P se nastaví iniciální výstupní hodnota.

```

30     syms s
31     %Initialization
32     b_min=Reg(1)-3*d;
33     b_max=Reg(2)+3*d;
34     w_min=Reg(3)-3*d;
35     w_max=Reg(4)+3*d;
36
37     m=size(M);
38     if m(1)>1
39         M=poly(M,s);
40     end
41
42     P='No roots in the selected region';
43

```

Obrázek 8.2 Popis algoritmu QPmR v matlabu, část 2

(zdroj: 29)

Na obrázku 8.3 dochází k určení mřížky (diskretizaci oblasti). Nejprve jsou určeny počty prvků imaginární nW a reálné nB osy. Poté jsou osy W a B sestaveny pomocí jejich hranic a počtu prvků. Pomocí vektorů jedniček oB a oW jsou vytvořeny matice reálných a imaginárních částí hodnot mřížky WW a BB , ze kterých je poté sestavena oblast D jako mřížka odpovídající tvaru Γ v popisu běhu algoritmu 5.1.1. Jsou určeny hodnoty kvazi-polynomu v těchto bodech MM , které jsou rozděleny na reálné R a imaginární I části.

```

44     nW=round((w_max-w_min)/d+1);
45     nB=round((b_max-b_min)/d+1);
46     W=linspace((w_min),(w_max),nW);
47     B=linspace((b_min),(b_max),nB);
48
49     oB=ones(1,nW);
50     oW=ones(1,nB);
51
52     WW=oW'*W;
53     BB=oB'*B;
54
55     D=BB+WW'*j;
56
57     MM=subs(M,s,D);
58     MM=double(MM);
59     R=real(MM);
60     I=imag(MM);
61

```

Obrázek 8.3 Popis algoritmu QPmR v matlabu, část 3

(zdroj: 29)

Na obrázku 8.4 jsou vypočteny a obdrženy body konturních křivek C_i imaginárních a C_r reálných částí. Ty jsou vykresleny v grafu.

```

63 figure(2);
64 hold off;
65 title('System Poles');
66 [Ci,Hi]=contour(B,W,I,[0 0],'r-');
67 hold on;
68 [Cr,Hr]=contour(B,W,R,[0 0],'b-');
69 axis([Reg(1) Reg(2) Reg(3) Reg(4)]);
70 xlabel('Re\its \rm','FontSize',10)
71 ylabel('Im\its \rm','FontSize',10)
72 pause(0.1);
73 p=0;

```

Obrázek 8.4 Popis algoritmu QPmR v matlabu, část 4

(zdroj: 29)

Obrázek 8.5. Obdržené hodnoty C_r jsou ve tvaru matice o dvou řádcích, které odpovídají souřadnicím v komplexní rovině. Tyto hodnoty jsou zachovány v matici Crr a matice C_r je převedena na vektor komplexních čísel. Tento vektor obsahuje konturní křivky, kde každá křivka je dána po sobě jdoucími komplexními body na této křivce. V těchto bodech jsou určeny hodnoty kvazi-polynomu M a je uložena jejich imaginární část do vektoru Iic . Tento postup je také zobrazen na obrázku 5.2 v popisu běhu algoritmu.

```

74 %uprava Cr
75
76 pCr=size(Cr);      %kontrola po sobe jdoucich hodnot
77 pCr=pCr(2);
78 Crr=Cr;
79 Cr=Cr(1,:)+Cr(2,:)*j;
80 Iic=imag(subs(M,s,Cr));

```

Obrázek 8.5 Popis algoritmu QPmR v matlabu, část 5

(zdroj: 29)

Na obrázku 8.6 jsou určeny hodnoty (odhady) kořenů pomocí hodnot Iic . Tyto hodnoty v sousedních bodech kořenů střídají své znaménko. Dále je vyšetřeno, zda se kořen nachází na uživatelem definované oblasti Reg a zda je kořen a jeho sousední bod obsahem jedné konturní křivky. Při splnění těchto podmínek je bod uložen.

Určování kořene pomocí sousedních bodů a následná kontrola, zda nejsou tyto kořeny vzdáleny více než $2d$, je možná právě kvůli specifické struktuře uložení konturních křivek v matlabu. Ta vždy jako první bod křivky nese informaci o počtu bodů, z kolika se konturní křivka skládá. V tomto algoritmu dochází k určování hodnot i v těchto "bodech", protože

konturní křivky nejsou nijak rozděleny. V implementaci QPmR je tento fakt ošetřen právě předchozími podmínkami.

```

82     np=1;
83     ch=1;
84     for no=2:pCr-2
85         if ((IIc(no-1)*IIc(no+1))<0)&(ch==1)
86             if (Reg(1)<=real(Cr(no))&(Reg(2)>=real(Cr(no))&(Reg(3)<=imag(Cr(no))&(Reg(4)>=imag(Cr(no)))
87                 if (abs(Crr(1,no)-Crr(1,no+1))<2*d)&(abs(Crr(2,no)-Crr(2,no+1))<2*d)
88                     Pp(np)=Cr(no);
89                     np=np+1;
90                     ch=0;
91                 end
92             end
93         else
94             ch=1;
95         end
96     end
97
98     P=Pp;

```

Obrázek 8.6 Popis algoritmu QPmR v matlabu, část 6

(zdroj: 29)

Na obrázku 8.7 dochází ke zpřesnění hodnot nalezených kořenů pomocí Newtonovy metody. Pokud byl nalezen nějaký kořen (návratová hodnota neobsahuje zprávu, ale kořeny), jsou všechny kořeny vyšetřeny. Výpočet zpřesnění běží dokud není splněna podmínka přesnosti Newtonovy metody pro všechny kořeny. Následně pokud má nějaký kořen nulovou imaginární část, je uložen pouze s reálnou částí. Všechny kořeny jsou poté přidány do grafu ke konturním křivkám.

```

100     %numeric calculation of poles
101     if ~(ischar(P))
102         clear P
103         H=M/diff(M);
104         n=size(Pp);
105         n=n(2);
106         for o=1:n
107             a=Pp(o);
108             Newton_run=1;
109             while Newton_run
110                 a_k1=a;
111                 s=a;
112                 a=a-eval(H);
113                 Newton_run=(abs(a-a_k1)>e);
114             end
115             if (~isreal(a))&(abs(imag(a))<0.1*e) %orezani imag polu
116                 a=real(a);
117             end
118
119             P(o,1)=a;
120             figure(2);
121             hold on;
122         end
123         plot(real(P),imag(P),'ko','MarkerFaceColor',[0 0 0],'MarkerSize',3);
124         pause(0.01)
125     end

```

Obrázek 8.7 Popis algoritmu QPmR v matlabu, část 7

(zdroj: 29)

Na obrázku 8.8 dochází k finální úpravě. Kořeny jsou seřazeny podle vzrůstajícího modulu a poté jsou nastaveny jako návratová hodnota.

Seřazení kořenů probíhá tak, že se vezme největší hodnota modulu kořenu a ta se uloží do P_{max} . V každé iteraci se vezme aktuální nejmenší hodnota z matice kořenů P a ta se uloží do nové matice PM na pozici aktuálně vyšetřovaného indexu. Tato hodnota se pak v matici s původními kořeny P přepíše na dvojnásobek nejvyšší hodnoty P_{max} . Tím je zajištěno, že již vybrané prvky už nebudou uvažovány, protože se vždy vybírá prvek s nejmenším modulem a počet iterací odpovídá počtu kořenů.

```

127 Pmax=max(abs(P));
128 for k=1:n
129     [Pm o]=min(abs(P));
130     PM(k,1)=P(o);
131     P(o)=2*Pmax;
132 end
133 P=PM;

```

Obrázek 8.8 Popis algoritmu QPmR v matlabu, část 8

(zdroj: 29)

Zavolání metody

Na obrázku 8.11 je ukázka příkazu pro zavolání této metody.

```

>> syms s
M = s + exp(-s);
Reg = [-7 3 0 20];
d = 0.1;
e = 0.0001;
fx >> P = QPmR(M, Reg, d, e)

```

Obrázek 8.9 Volání metody QPmR

(zdroj: 29)

8.2 QPmR v2

Algoritmus QPmR v2 je rozšířenou verzí QPmR v1. Dochází zde k adaptivnímu nastavování velikosti kroku mřížky na oblasti, na které je výpočet prováděn. Dále je vyšetřovaná oblast dělena na podoblasti podle výskytu kořenů. Jeho popisu je věnována kapitola (5.2).

Kvůli jeho složitosti došlo k výraznému nárůstu počtu řádků v kódu metody QPmR v2 (ze 130 přibližně na 1000). Přehledně je metoda znázorněna na pseukódu v teoretické části (5.2.2). Metoda QPmR v2 může být volána více způsoby.

První je možnost zadání kvazi-polynomu. Ten může být zadán buď pomocí vektoru zpoždění a matice polynomů, nebo jako anonymní funkce.

Další je možnost zadání vstupních parametrů. Pro možnost adaptace mřížky musí být hodnota pro krok mřížky nastavena na hodnotu -1. Pokud je nastavena nějaká požadovaná přesnost, pak se žádná adaptace neprovádí a výpočet na mřížce je proveden jednou s požadovanou přesností. Hodnota pro požadovanou přesnost může, ale nemusí být zadána.

Poslední možností pro zadání je vykreslování grafu. Uživatel zadává, zda chce výsledky vykreslit, či nikoliv.

Popis metody

Z hlediska zadání kvazi-polynomu je první část metody rozdělena do dvou větví. První je zadání pomocí vektoru a matice a druhá je pomocí anonymní funkce. Při zadání vektoru a matice při prvním výpočtu metody je na konci metody sestavena anonymní funkce a při dalším výpočtu už se volá metoda se zadanou anonymní funkcí. Z tohoto důvodu bude popsána větev se zadáním anonymní funkce.

Na začátku algoritmu je nastavena velikost kroku mřížky a přesnost. Poté algoritmus probíhá stejně ve verzi 1. Je rozšířena vyšetřovaná oblast a jsou vypočteny konturní křivky.

Poté jsou určeny odhady kořenů. Dále dochází ke zpřesnění odhadu kořenů. Společně s odhadem kořenů je nastavena hodnota proměnné určující, zda byla Newtonova metoda úspěšná.

Až po provedení upřesnění kořenů jsou vyloučeny kořeny, které se nachází mimo uživatelem zadanou oblast.

Dále jsou vyšetřeny podmínky, zda nalezené kořeny splňují zadanou přesnost. První kon-

```

409 while min(abs(Rback(k)-Rr))<2*ds&brk
410     [Vi I]=min(abs(Rback(k)-Rr));
411     Rr(I)=Inf;
412     k=k+1;
413     if k>length(R)
414         k=k-1;
415         brk=0;
416     end
417 end
418 if k==length(R)
419     reduce_ds1=0;
420 end
421 end

```

Obrázek 8.10 Kontrola kořenů v QPmR v2 podle mřížky

(zdroj: 29)

trolou je, zda se kořeny nachází v rámci mřížky. Následně je vyšetřen princip argumentu.

```

423 %checking by argument principle
424 Ir=argp(Funfin,Reg,ds/10,e);
425 Ir2=argp(Funfin,Reg+[ds/10 -ds/10 ds/10 -ds/10],ds/10,e)
426 reduce_ds2=1;
427 if Ir==length(R)|Ir2==length(R)
428     reduce_ds2=0;
429 end

```

Obrázek 8.11 Kontrola kořenů v QPmR v2 podle principu argumentu

(zdroj: 29)

Dále probíhá algoritmus podobně, jako bylo naznačeno na pseudokódu 5.2.2.

- Pokud není potřeba dále zjemňovat mřížku nebo už nelze mřížku dále zjemňovat, je provedena kontrola kořenů.
- Pokud se algoritmus nenachází v podoblastech původní oblasti a je potřeba zjemnit současnou mřížku, dojde ke zjemnění mřížky v případě, že už nebyla mřížka dvakrát zjemněna. Algoritmus QPmR je znovu zavolán.
- Pokud již byla mřížka dvakrát zjemněna a je potřeba mřížku dále zjemnit, dojde ke zjemnění oblasti. Algoritmus QPmR je znovu zavolán.

Po volání metody QPmR se vrací návratové hodnoty. Po každém návratu dochází k jejich sjednocení. Nakonec jsou nastaveny výsledné hodnoty.

9 IMPLEMENTACE V C++

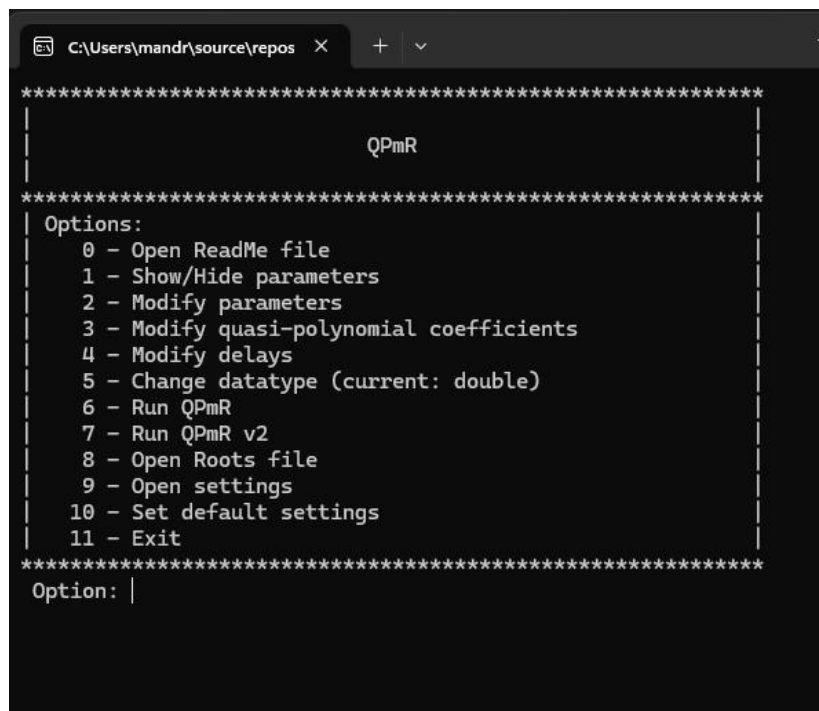
Pro implementaci v jazyku C++ byla zvolena možnost konzolové aplikace. Cílem je implementovat metodu, která bude časově méně náročná než implementace v matlabu. Z tohoto důvodu je pozornost zaměřena na výpočetní čas, ne na vizuální stránku aplikace. Zadávaní vstupů probíhá pomocí textových souborů. Stejně tak výstupy jsou obdrženy v textových souborech.

9.1 Práce s aplikací

Komunikace probíhá pomocí `menu` v konzolové aplikaci. Důležité možnosti jsou vypsány níže.

- Zadání parametrů – uživatel zadává hranice vyštrěvané oblasti, velikost kroku a požadovanou přesnost.
- Zadání kvazi-polynomiálních parametrů – uživatel zadává koeficienty polynomů jednotlivých zpoždění. Zadání koeficientů probíhá sestupně podle mocniny.
- Zadání zpoždění – uživatel zadává jednotlivá zpoždění v systému.
- Změna datového typu – uživatel má možnost změnit datový typ. Dostupné datové typy jsou `float`, `double` a `long double`. Výchozí datový typ je `double`.
- Spuštění QPmR a QPmR v2.
- Otevření souboru s kořeny.
- Otevření souboru s nastavením (je popsáno níže).
- Nastavení výchozích hodnot nastavení.
- Ukončení aplikace.

Na obrázku 9.1 je ukázána konzolová aplikace s menu.



```
*****
|                                     QPmR                                     |
|*****|
|Options:|
| 0 - Open ReadMe file|
| 1 - Show/Hide parameters|
| 2 - Modify parameters|
| 3 - Modify quasi-polynomial coefficients|
| 4 - Modify delays|
| 5 - Change datatype (current: double)|
| 6 - Run QPmR|
| 7 - Run QPmR v2|
| 8 - Open Roots file|
| 9 - Open settings|
|10 - Set default settings|
|11 - Exit|
|*****|
|Option: |
```

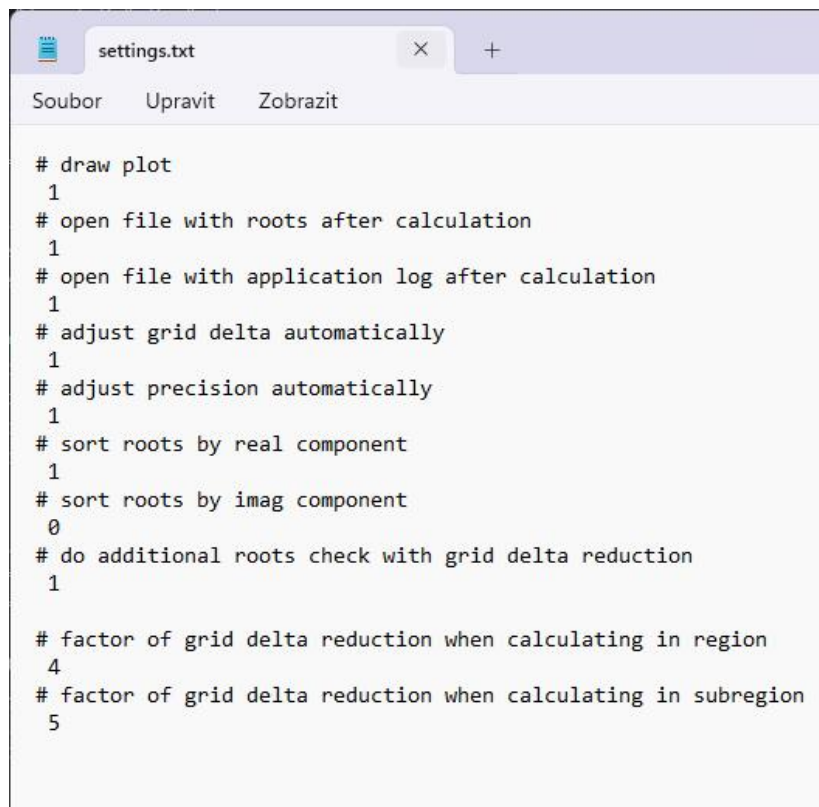
Obrázek 9.1 Menu aplikace

(zdroj: vlastní)

Aplikace nabízí nastavení s následujícími parametry.

- Vykreslení grafu.
- Otevření souboru s nalezenými kořeny.
- Otevření souboru, ve kterém je zobrazen průběh výpočtu.
- Nastavení možnosti adaptace mřížky a nastavení přesnosti.
- Seřazení nalezených kořenů podle reálné nebo imaginární části.
- Možnost provedení kontroly kořenů při běhu s adaptací mřížky.
- Dělitel zmenšení kroku mřížky při výpočtu na dané oblasti.
- Dělitel zmenšení kroku mřížky při výpočtu v podoblastech.

Soubor s nastavením je zobrazen na obrázku 9.2.



```
# draw plot
1
# open file with roots after calculation
1
# open file with application log after calculation
1
# adjust grid delta automatically
1
# adjust precision automatically
1
# sort roots by real component
1
# sort roots by imag component
0
# do additional roots check with grid delta reduction
1

# factor of grid delta reduction when calculating in region
4
# factor of grid delta reduction when calculating in subregion
5
```

Obrázek 9.2 Nastavení výpočtu

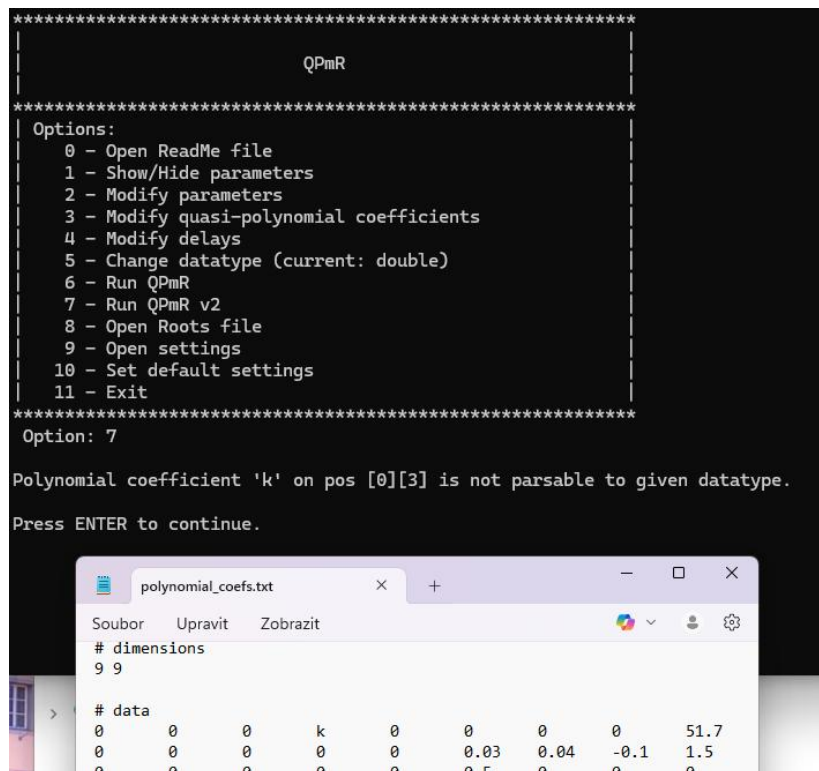
(zdroj: vlastní)

9.2 Struktura aplikace

Aplikace je členěna následujícím způsobem.

1. Při spuštění dojde v metodě main k vytvoření instance třídy Aplikace a pomocí metody run je aplikace spuštěna.
2. Třída Aplikace je zodpovědná za komunikaci s uživatelem pomocí konzole. Nabízí aplikační menu a spouští jednotlivé akce.
3. Namespace utils slouží ke komunikaci prostřednictvím textových souborů. Řádky v textových souborech, které začínají znakem # jsou brány jako komentář.
4. Třída Handler se stará o kontrolu zadaných textových souborů a o převod zadaných textových řetězců na čísla zvoleného datového typu. Chyba v zadání je zobrazena na obrázku.
5. Třída Timer se stará o měření výpočtů v různých částech kódu.

6. Třída Gnuplot slouží k vykreslování nalezených kořenů.
7. Třída Contour slouží k výpočtu konturních křivek.
8. Třída Qpmr obsahuje metody QPmR a QPmR v2 pro výpočet kořenů kvazi-polynomu.



```

*****
|                                     |
|                               QPmR |
|                                     |
|*****|
|Options:|
| 0 - Open ReadMe file|
| 1 - Show/Hide parameters|
| 2 - Modify parameters|
| 3 - Modify quasi-polynomial coefficients|
| 4 - Modify delays|
| 5 - Change datatype (current: double)|
| 6 - Run QPmR|
| 7 - Run QPmR v2|
| 8 - Open Roots file|
| 9 - Open settings|
|10 - Set default settings|
|11 - Exit|
|*****|
Option: 7

Polynomial coefficient 'k' on pos [0][3] is not parsable to given datatype.

Press ENTER to continue.

```

polynomial_coefs.txt

# dimensions	
9	9

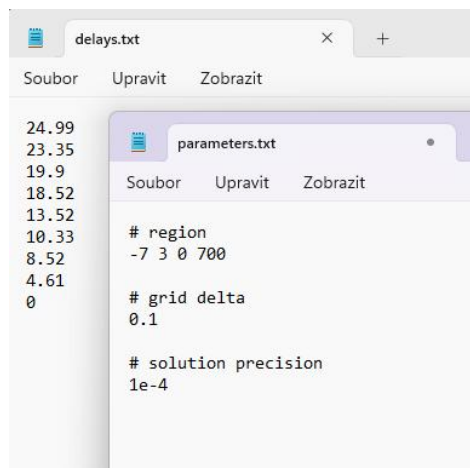
# data	
0	0 0 k 0 0 0 0 51.7
0	0 0 0 0 0 0.03 0.04 -0.1 1.5
0	0 0 0 0 0 0.5 0 0 0

Obrázek 9.3 Ukázka chybného zadání

(zdroj: vlastní)

9.3 Volání aplikace

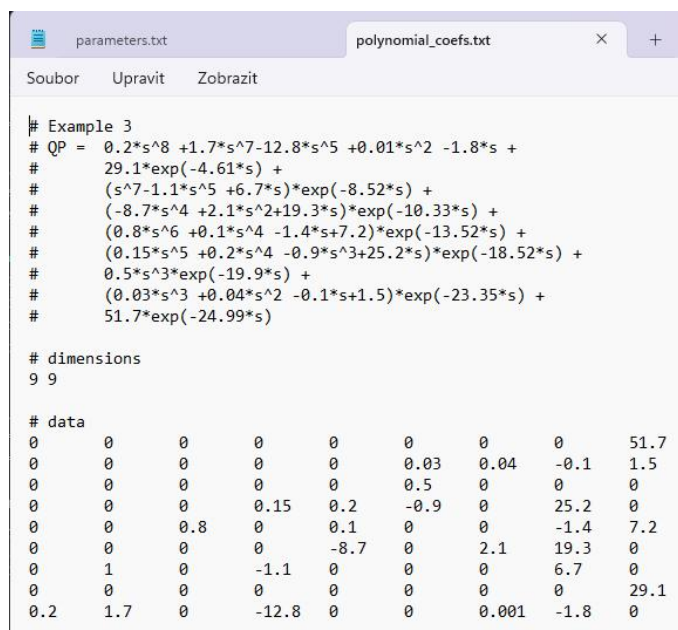
V této podkapitole je ukázáno vzorové zadání pro kvazi-polynom daný na obrázku 9.5. Zadání zpoždění a parametrů je zobrazeno na obrázku 9.4. Při zadání zpoždění není potřeba zadávat zpoždění seřazeně, jen i -té zpoždění musí odpovídat i -tému zadanému polynomu.



Obrázek 9.4 Ukázka zadání zpoždění a parametrů

(zdroj: vlastní)

Ukázka zadání kvazi-polynomiálních koeficientů je zobrazena na obrázku 9.5.



Obrázek 9.5 Ukázka zadání koeficientů kvazi-polynomu

(zdroj: vlastní)

9.4 Výstupy aplikace

Aplikace kromě vykreslení a vypsání kořenů nabízí vypsání průběhu výpočtu. Tento záznam je v nastavení označen jako application log, jeho ukázka je na obrázku 9.6. Jeho výhodou je záznam o kompletním rozdělení oblastí a sledování kritických částí výpočtu.

```
Recursion level 1, subregion: 4

Zeros count: 441
Flag: 1
Accuracy: 3e-06
Grid delta: 0.0075

---- FINAL RESULTS ----

Zeros count: 1196
Flag: 1
Accuracy: 3e-06
Grid delta: 0.0075

----- TIME OVERVIEW -----

Time values      : 12.280340400000002
Time values (calc): 9.484952299999998
Time contours     : 7.2398309
Time vals on cntr : 0.17513980000000007
Time roots        : 0.005038199999999998
Time newton        : 0.09575799999999998
Time argp (both)  : 4.8840098000000001
```

Obrázek 9.6 Ukázka zadání koeficientů kvazi-polynomu

(zdroj: vlastní)

10 PŘÍKLADY

V této kapitole jsou zobrazeny grafické výstupy implementací v matlabu a C++. Dále jsou porovnány výpočetní časy.

10.1 Příklad 1

Prvním příkladem je jednoduchý kvazi-polynom $q(s) = s + e^{-s}$. Jsou sledovány výpočetní časy QPmR v2 metod v matlabu a C++.

Na obrázku 10.1 je zobrazeno volání metody QPmR v2 v matlabu.

```
>> QP=@(s) (s+exp(-s));
>> Reg=[-10 3 0 300];
>> tic; [R Y]=QPmR(Reg, QP, -1, -1, 1); toc;
Grid size reduced from 3.9 to 0.975
Elapsed time is 0.128418 seconds.
fx \>
```

Obrázek 10.1 Ukázka zadání QPmR v2

(zdroj: vlastní)

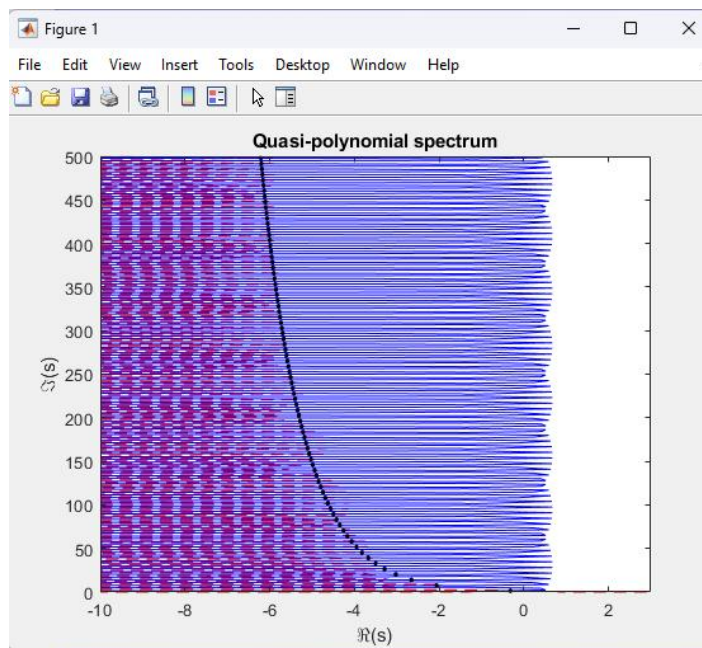
Tabulka 10.3 zobrazuje výpočetní časy pro dvě různé oblasti.

Tabulka 10.1 Výpočetní časy u příkladu 1

	matlab	C++
[-10 3 0 300]	0.128 s	0.013 s
[-10 3 0 500]	1.029 s	0.020 s

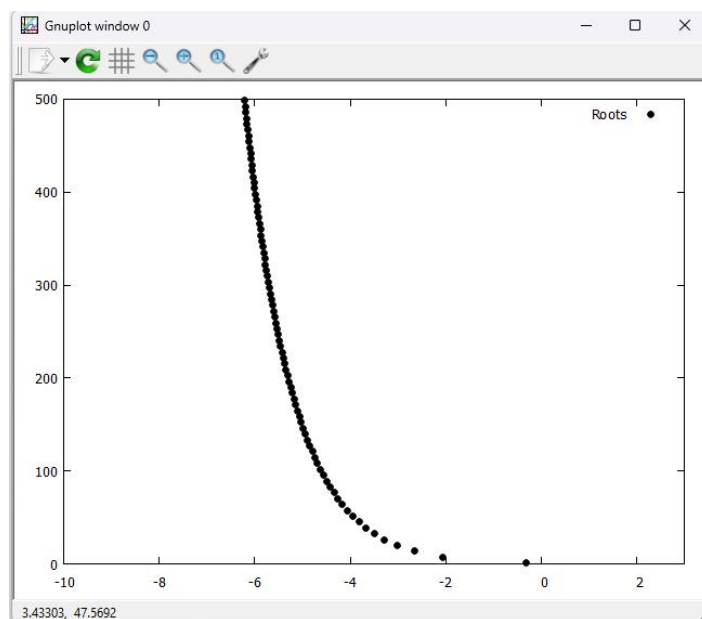
(zdroj: vlastní)

Dále jsou na následujících obrázcích zobrazeny grafické výstupy matlabu a C++.



Obrázek 10.2 Ukázka matlab výstupu QPmR v2

(zdroj: vlastní)



Obrázek 10.3 Ukázka C++ výstupu QPmR v2

(zdroj: vlastní)

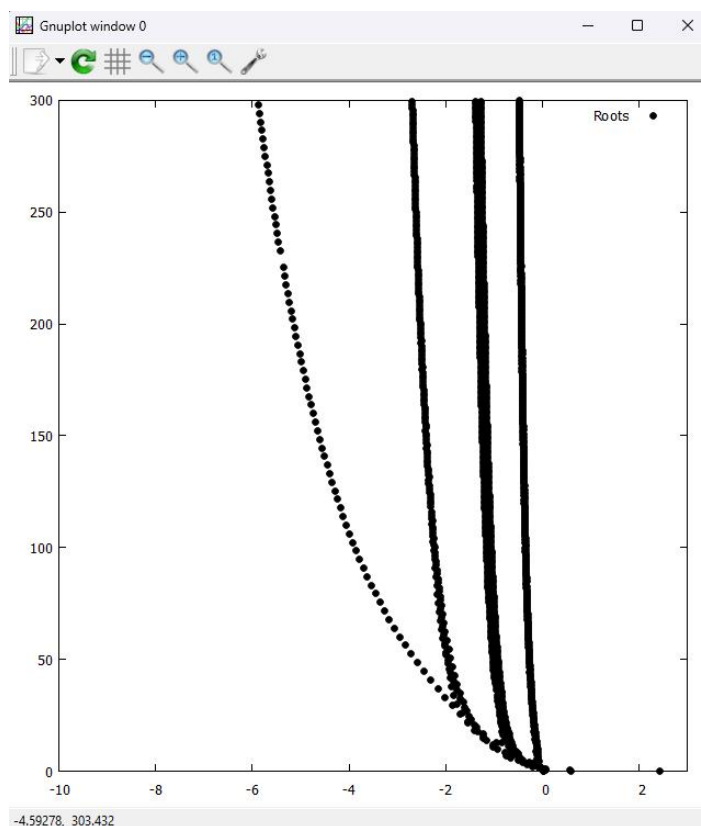
10.2 Příklad 2

Dalším příkladem je kvazi-polynom daný na obrázku 9.5. Tentokrát je na obrázku 10.4 vykresleno jen řešení v C++.

Tabulka 10.2 Výpočetní časy u příkladu 2

	matlab	C++
$[-10 \ 3 \ 0 \ 300]$	21.659 s	6.623 s

(zdroj: vlastní)



Obrázek 10.4 Ukázka C++ výstupu, příklad 2

(zdroj: vlastní)

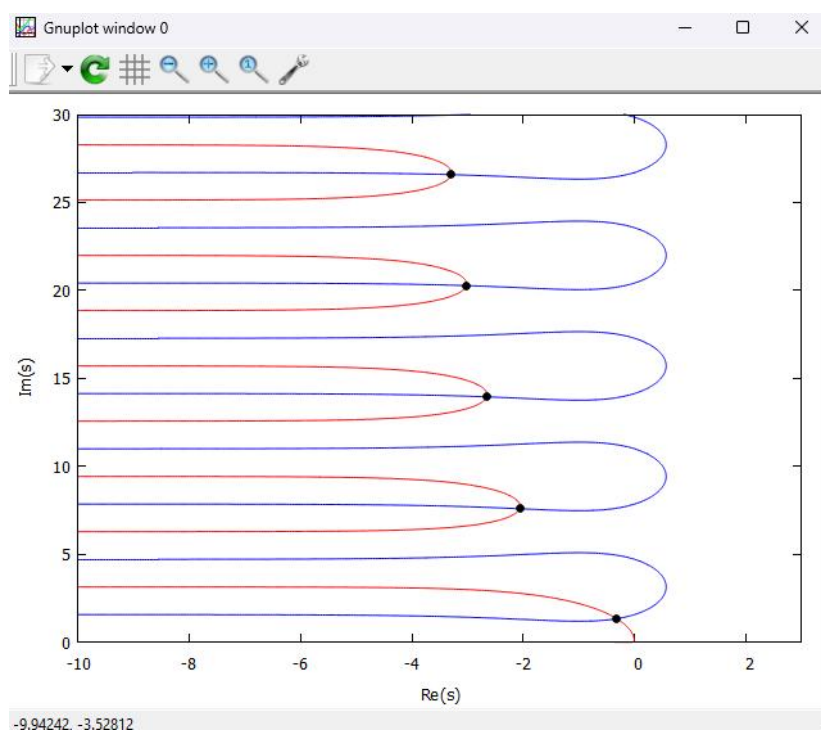
10.3 Příklad 3

V příkladu 3 jsou srovnány všechny metody, na které je tato práce zaměřena. Nastavením kroku mřížky a přesnosti u metody QPmR v2 lze docílit stejného chodu, jako je tomu u verze 1. V matlabu je QPmR v1 implementována s použitím symbolických funkcí, zatímco verze využívá anonymní funkce. V C++ je v této práci u verze 1 využita knihovna Gnuplot k výpočtu konturních křivek. Komunikace s knihovnou gnuplot probíhá pomocí textového souboru, u nějž je zápis a čtení zdlouhavé. V tomto příkladu jsou porovnány výpočetní časy verzí 1 a 2 v matlabu a C++. Zadání kvazi-polynomu je stejné jako v příkladu 1. Krok mřížky je zvolen 0.05 a přesnost 0.0001.

Tabulka 10.3 Výpočetní časy u příkladu 3

	matlab v1	matlab v2	C++ v1	C++ v2
[-10 3 0 30]	14.433 s	0.220 s	1.095 s	0.007 s

(zdroj: vlastní)



Obrázek 10.5 Grafický výstup QPmR v1 v C++

(zdroj: vlastní)

ZÁVĚR

V této diplomové práci byla popsána úloha výpočtu kořenů kvazi-polynomů, což je nezbytný krok pro posouzení stability lineárních dynamických systémů se zpožděním. Byly detailně popsány stavové a vstupně-výstupní modely těchto systémů, a byl odvozen jejich vzájemný přechod. Významná část práce se soustředila na vlastnosti charakteristického kvazi-polynomu a jeho vliv na stabilitu systému.

Byly popsány algoritmy QPmR a QPmR v2 a jejich stávající implementace v matlabu. Byla vytvořena C++ aplikace s implementací těchto metod s cílem optimalizovat a snížit výpočetní čas. Výsledky ukázaly, že implementace v C++ přinesla urychlení výpočtů, což bylo potvrzeno na vybraných příkladech.

SEZNAM POUŽITÉ LITERATURY

- [1] VYHLÍDAL, Tomáš, 2003. *Analysis and Synthesis of Time Delay System Spectrum*. Disertační práce. České vysoké učení technické v Praze, Fakulta strojní. [cit. 2025-05-27].
- [2] BELLMAN, Richard a COOKE, Kenneth L., 1963. *Differential-Difference Equations*. Academic Press, New York.
- [3] BOYD, Stephen, 2007. *Linear dynamical systems with inputs & outputs*. Online. Stanford University. Dostupné z: <https://see.stanford.edu/materials/lsoeldsee263/13-lin-sys.pdf>. [cit. 2025-05-27].
- [4] PEKAŘ, Libor, 2019. *INTRODUCTION TO TIME-DELAY SYSTEMS (ERASMUS+ STAFF MOBILITY FOR TEACHING MARIBOR, SLOVENIA)*. Online. Tomas Bata University in Zlín. Dostupné z: <http://dx.doi.org/10.13140/RG.2.2.22357.37601>. [cit. 2025-05-27].
- [5] TENDRAKE, Russ, 2023. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation (Course Notes for MIT 6.832)*. Online. Dostupné z: <https://underactuated.csail.mit.edu/multibody.html>. [cit. 2025-5-27].
- [6] MATHWORKS, [b.r.]. *Multibody Dynamics*. Online. Dostupné z: <https://www.mathworks.com/help/sm/multibody-dynamics.html>. [cit. 2025-05-27].
- [7] MIT, 2006. *Introduction to Electronics, Signals, and Measurement (Lecture Notes)*. Online. MIT Electrical Engineering and Computer Science. Dostupné z: https://ocw.mit.edu/courses/6-071j-introduction-to-electronics-signals-and-measurement-spring-2006/resources/16_transint_rlc2/. [cit. 2025-05-27].
- [8] MATHWORKS, [b.r.]. *Model House Heating System*. Online. Dostupné z: <https://www.mathworks.com/help/simulink/ug/model-a-house-heating-system.html>. [cit. 2025-05-27].

- [9] HUSSEIN, Shaza, 2010. Predator-Prey Modeling. *Undergraduate Journal of Mathematical Modeling: One + Two*. 2010. roč. 3, č. 1. ISSN 2326-3652. Dostupné z: <http://dx.doi.org/10.5038/2326-3652.3.1.32>. [cit. 2025-05-27].
- [10] BROWNE, Christopher A.; AMCHINAND, Daniel B.; SCHNEIDER, Joanna a DATTA, Sujit S., 2021. Infection Percolation: A Dynamic Network Model of Disease Spreading. *Frontiers in Physics*. 2021. roč. 9. ISSN 2296-424X. Dostupné z: <https://doi.org/10.3389/fphy.2021.645954>. [cit. 2025-05-27].
- [11] ANGELETOS, George-Marios, 2000. *Introduction and The Solow Model (14.05 Lecture Notes)*. Online. MIT Department of Economics. Dostupné z: https://ocw.mit.edu/courses/14-05-intermediate-macroeconomics-spring-2013/resources/mit14_05s13_lecnot_int_sol/. [cit. 2025-05-27].
- [12] BOYD, Stephen, 2007. *Linear functions and examples*. Online. Stanford University. Dostupné z: <https://see.stanford.edu/materials/lsoeldsee263/02-linear-fcts.pdf>. [cit. 2025-05-27].
- [13] PEKAŘ, Libor, 2021. *On a quasioptimal spectrum assignment for delayed systems*. In: *Control Strategy for Time-Delay Systems*. Elsevier, s. 137–187. ISBN 978-0-128-20599-0. Dostupné z: <https://doi.org/10.1016/B978-0-12-820599-0.00009-4>.
- [14] PEKAŘ, Libor, 2019. *On the numerical determination of stability regions in the delay space via dominant pole estimation*. In: *Stability, Control and Application of Time-Delay Systems*. Elsevier, s. 1–22. ISBN 978-0-12-814928-7. Dostupné z: <https://doi.org/10.1016/B978-0-12-814928-7.00001-9>.
- [15] MICHIELS, Wim, 2017. *Delays Effects in Dynamical Systems and Networks: Analysis and Control Interpretations*. In: van de Wouw, N., Lefeber, E., Lopez Arteaga, I. (eds) *Nonlinear Systems. Lecture Notes in Control and Information Sciences*. Springer, Cham. ISBN 978-3-319-30356-7. Dostupné z: https://doi.org/10.1007/978-3-319-30357-4_6.
- [16] VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ, FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ, [b.r.]. *Laplaceova transformace*. Online. Dostupné z: <https://www.umat.fekt.vut.cz/~svobodaz/JMA2/laplace.pdf>. [cit. 2025-05-27].

- [17] VYHLÍDAL, Tomáš a ZÍTEK, Pavel, 2003. Quasipolynomial mapping based rootfinder for analysis of time delay systems. *IFAC Proceedings Volumes*. 2003. roč. 36, č. 19, s. 227–232. ISSN 1474-6670. Dostupné z: [https://doi.org/10.1016/S1474-6670\(17\)33330-X](https://doi.org/10.1016/S1474-6670(17)33330-X). [cit. 2025-05-27].
- [18] VYHLÍDAL, Tomáš a ZÍTEK, Pavel, 2014. *QPmR - Quasi-Polynomial Root-Finder: Algorithm Update and Examples*. In: VYHLÍDAL, Tomáš; LAFAY, Jean-François a SI-PAHI, Rifat (ed.). *Delay Systems. Advances in Delays and Dynamics*. Cham: Springer International Publishing, s. 299–312. ISBN 978-3-319-01694-8. Dostupné z: https://doi.org/10.1007/978-3-319-01695-5_22.
- [19] MICHIELS, Wim a VYHLÍDAL, Tomáš, 2005. An eigenvalue based approach for the stabilization of linear time-delay systems of neutral type. *Automatica*. 2005. roč. 41, č. 6, s. 991–998. ISSN 0005-1098. Dostupné z: <https://doi.org/10.1016/j.automatica.2004.11.032>. [cit. 2025-05-27].
- [20] PEKAŘ, Libor, 2019. *LTI TDS SPECTRUM, ITS ANALYSIS AND SYNTHESIS; STABILITY*. Online. Tomas Bata University in Zlín. Dostupné z: <http://dx.doi.org/10.13140/RG.2.2.28229.40162>. [cit. 2025-05-27].
- [21] HAYAT, Amaury, 2021. Boundary stabilization of 1D hyperbolic systems. *Annual Reviews in Control*. 2021. roč. 52, s. 222–242. ISSN 1367-5788. Dostupné z: <https://doi.org/10.1016/j.arcontrol.2021.10.009>. [cit. 2025-05-27].
- [22] OPENCOURSEWARE, MIT, 2010. *Newton's Method (MIT Lecture Notes in 18.01SC Single Variable Calculus)*. Online. MIT OpenCourseWare. Dostupné z: https://ocw.mit.edu/courses/18-01sc-single-variable-calculus-fall-2010/40bc34f4471c7c813910d7e006b28919_MIT18_01SCF10_Ses33a.pdf. [cit. 2025-05-27].
- [23] VERHEYDEN, Koen; LUZYANINA, Tatyana a ROOSE, Dirk, 2008. Efficient computation of characteristic roots of delay differential equations using LMS methods. *Journal of Computational and Applied Mathematics*. 2008. roč. 214, s. 209–226. ISSN 0377-0427. Dostupné z: <https://doi.org/10.1016/j.cam.2007.02.025>. [cit. 2025-05-27].

- [24] BREDÁ, Dimitri; VERMIGLIO, Rossana a MASET, Stefano, 2005. Pseudospectral Differencing Methods for Characteristic Roots of Delay Differential Equations. *SIAM Journal on Scientific Computing*. 2005. roč. 27, s. 482–495. ISSN 1095-7197. Dostupné z: <https://doi.org/10.1137/030601600>. [cit. 2025-05-27].
- [25] BREDÁ, Dimitri, 2006. Solution operator approximations for characteristic roots of delay differential equations. *Applied Numerical Mathematics*. 2006. roč. 56, s. 305–317. ISSN 1873-5460. Dostupné z: <https://doi.org/10.1016/j.apnum.2005.04.010>. [cit. 2025-05-27].
- [26] BREDÁ, Dimitri; MASET, Stefano a VERMIGLIO, Rossana, 2009. *TRACE-DDE: a Tool for Robust Analysis and Characteristic Equations for Delay Differential Equations*. In: Loiseau, J.J., Michiels, W., Niculescu, S.I., Sipahi, R. (eds) *Topics in Time Delay Systems. Lecture Notes in Control and Information Sciences*. Springer, Berlin, Heidelberg, s. 145–155. ISBN 978-3-642-02897-7. Dostupné z: https://doi.org/10.1007/978-3-642-02897-7_13.
- [27] ENGELBORGH, Koen; LUZYANINA, Tatyana a ROOSE, Dirk, 2000. Numerical bifurcation analysis of delay differential equations. *Journal of Computational and Applied Mathematics*. 2000. roč. 125. ISSN 0377-0427. Dostupné z: [https://doi.org/10.1016/S0377-0427\(00\)00472-6](https://doi.org/10.1016/S0377-0427(00)00472-6). [cit. 2025-05-27].
- [28] ENGELBORGH, Koen; LUZYANINA, Tatyana a SAMAEY, Giovanni, 2001. DDE-BIFTOOL v. 2.00: a Matlab package for bifurcation analysis of delay differential equations. Technical report, KU Leuven, Department of Computer Science. Dostupné z: <https://www.cs.kuleuven.be/publicaties/rapporten/tw/TW330.pdf>.
- [29] VYHLÍDAL, Tomáš a ZÍTEK, Pavel, 2003. *QPmR – Quasi-Polynomial Mapping Based Rootfinder*. Online. České vysoké učení technické v Praze, Fakulta strojní. Dostupné z: <https://www.cak.fs.cvut.cz/algorithms/qpmr>. [cit. 2025-05-27].
- [30] VYHLÍDAL, Tomáš a ZÍTEK, Pavel, 2009. Mapping Based Algorithm for Large-Scale Computation of Quasi-Polynomial Zeros. *IEEE Transactions on Automatic Control*. 2009. roč. 54, č. 1, s. 171–177. ISSN 0018-9286. Dostupné z: <https://doi.org/10.1109/TAC.2008.2008345>. [cit. 2025-05-27].

- [31] WILLIAMS, Thomas a KELLEY, Colin, [b.r.]. *Gnuplot 6.0*. Online. Dostupné z: <http://gnuplot.info/>. [cit. 2025-05-30].
- [32] XU, Gang, 2015. High-order Extraction Method of Iso-curvature Lines on B-spline Surfaces. *Journal of Information and Computational Science*. 2015. roč. 12, s. 3945–3952. ISSN 1548-7741. Dostupné z: https://www.researchgate.net/publication/281198223_High-order_Extraction_Method_of_Iso-curvature_Lines_on_B-spline_Surfaces. [cit. 2025-05-30].
- [33] GEEKSFORGEEKS, Sanchhaya Education Private Limited, [b.r.]. *Introduction to Disjoint Set (Union-Find Algorithm)*. Online. Dostupné z: <https://www.geeksforgeeks.org/introduction-to-disjoint-set-data-structure-or-union-find-algorithm/>. [cit. 2025-05-30].
- [34] GEEKSFORGEEKS, Sanchhaya Education Private Limited, [b.r.]. *Depth First Search or DFS for a Graph*. Online. Dostupné z: <https://www.geeksforgeeks.org/depth-first-search-or-dfs-for-a-graph/>. [cit. 2025-05-30].
- [35] NGO, Roger, [b.r.]. *MARCHING SQUARES*. Online. Dostupné z: <https://urbansprinter.github.io/marchingsquares/>. [cit. 2025-05-30].
- [36] MATHFULL.COM, [b.r.]. *Linear Interpolation: Definition, Formula, Example*. Online. Dostupné z: <https://mathful.com/hub/linear-interpolation>. [cit. 2025-05-30].
- [37] GEEKSFORGEEKS, Sanchhaya Education Private Limited, [b.r.]. <https://www.geeksforgeeks.org/cpp-programming-intro/>. Online. Dostupné z: <https://www.geeksforgeeks.org/cpp-programming-intro/>. [cit. 2025-05-30].
- [38] STROUSTRUP, Bjarne, 2013. *The C++ Programming Language*. Addison-Wesley. 4 vyd. ISBN 978-0275967307.
- [39] CPPREFERENCE.COM, [b.r.]. *C++ reference*. Online. Dostupné z: <https://en.cppreference.com/w/>. [cit. 2025-05-30].
- [40] OPENMP, 2024. *OpenMP 6.0 API Syntax Reference Guide*. Online. Dostupné z: <https://www.openmp.org/wp-content/uploads/OpenMP-RefGuide-6.0-OMP60SC24-web.pdf>. [cit. 2025-05-30].

- [41] OPENMP, 2024. *openmp.com*. Online. Dostupné z: <https://www.openmp.org/>. [cit. 2025-05-30].
- [42] MICROSOFT, 2021. */O1, /O2 (Minimize Size, Maximize Speed)*. Online. Dostupné z: <https://learn.microsoft.com/en-us/cpp/build/reference/o1-o2-minimize-size-maximize-speed?view=msvc-170>. [cit. 2025-05-30].
- [43] MICROSOFT, 2025. *Visual Studio documentation*. Online. Dostupné z: <https://learn.microsoft.com/en-us/visualstudio/windows/?view=vs-2022&preserve-view=true>. [cit. 2025-05-30].
- [44] MICROSOFT, 2021. *OpenMP in Visual C++*. Online. Dostupné z: <https://learn.microsoft.com/en-us/cpp/parallel/openmp/openmp-in-visual-cpp?view=msvc-170>. [cit. 2025-05-30].

SEZNAM OBRÁZKŮ

Obrázek 3.1	Ukázka chování systému 1	34
Obrázek 3.2	Ukázka chování systému 2	35
Obrázek 4.1	Ukázka principu argumentu	37
Obrázek 5.1	Ukázka algoritmu QPmR	41
Obrázek 5.2	Ukázka odhadu kořene metody QPmR	43
Obrázek 5.3	Řešení algoritmu QPmR	44
Obrázek 5.4	Rozdělení oblasti metody QPmR v2	46
Obrázek 5.5	Rozdělení oblasti v algoritmu aQPmR.....	49
Obrázek 6.1	Marching squares algoritmus	52
Obrázek 6.2	Marching squares tabulka.....	52
Obrázek 6.3	Procházení grafu (Graph traversal)	55
Obrázek 7.1	Ukázka příkazu knihovny OpenMP	58
Obrázek 7.2	Ukázka Solution Exploreru ve VS	59
Obrázek 7.3	Nastavení optimalizace ve VS	60
Obrázek 7.4	Nastavení podpory OpenMP ve VS.....	60
Obrázek 8.1	Popis algoritmu QPmR, část 1	62
Obrázek 8.2	Popis algoritmu QPmR, část 2	63
Obrázek 8.3	Popis algoritmu QPmR, část 3	63
Obrázek 8.4	Popis algoritmu QPmR, část 4	64
Obrázek 8.5	Popis algoritmu QPmR, část 5	64
Obrázek 8.6	Popis algoritmu QPmR, část 6	65
Obrázek 8.7	Popis algoritmu QPmR, část 7	65
Obrázek 8.8	Popis algoritmu QPmR, část 8	66
Obrázek 8.9	Volání metody QPmR.....	66
Obrázek 8.10	Kontrola kořenů v QPmR v2 1	68
Obrázek 8.11	Kontrola kořenů v QPmR v2 2	68
Obrázek 9.1	Menu aplikace	70
Obrázek 9.2	Nastavení výpočtu	71
Obrázek 9.3	Ukázka chybného zadání.....	72
Obrázek 9.4	Ukázka zadání QPmR v C++	73

Obrázek 9.5	Ukázka zadání kvazi-polynomu v C++	73
Obrázek 9.6	Ukázka zadání kvazi-polynomu v C++	74
Obrázek 10.1	Ukázka zadání QPmR v2	75
Obrázek 10.2	Ukázka matlab výstupu QPmR v2	76
Obrázek 10.3	Ukázka C++ výstupu QPmR v2	76
Obrázek 10.4	Ukázka C++ výstupu, příklad 2	77
Obrázek 10.5	Grafický výstup QPmR v1 v C++	78

SEZNAM TABULEK

Tabulka 10.1	Výpočetní časy u příkladu 1	75
Tabulka 10.2	Výpočetní časy u příkladu 2	77
Tabulka 10.3	Výpočetní časy u příkladu 3	78

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

LTI Linear time-invariant

TDS Time delay-systems

SEZNAM PŘÍLOH

PŘÍLOHA P I: QPmR

PŘÍLOHA P II: QPmR_app

PŘÍLOHA P I: QPMR

Složka QPmR obsahuje projekt C++ aplikace ve Visual Studiu s implementací metod QPmR a QPmR v2. Složka obsahuje příslušné soubory Visual Studia, .hpp a .cpp soubory, složky data, examples a libs. Ve složce data jsou textové soubory pro čtení výstupních dat nebo zápis vstupních dat. Ve složce examples jsou příklady složek data. Složka libs obsahuje knihovnu gnuplot.

PŘÍLOHA P II: QPMR_APP

Složka QPmR_app obsahuje konzolovou .exe aplikaci s implementací metod QPmR a QPmR v2. Dále jsou v této složce složky data potřebné pro čtení výstupních dat a zápis vstupních dat, a složka libs obsahující knihovnu gnuplot.