Aalto University
School of Science
Master's Programme in Mathematics and Operations Research

Lauri Nyman

Solving Systems of 3 Equations in 3 Variables via Resultants

Master's Thesis
Espoo, February 22, 2021

Supervisor: Professor Vanni Noferini
Advisor: Professor Vanni Noferini

Aalto University
School of Science
Master's Programme in Mathematics and Operations Research

ABSTRACT OF MASTER'S THESIS

Author: Lauri Nyman

Title:
Solving Systems of 3 Equations in 3 Variables via Resultants

Date: February 22, 2021 **Pages:** v + 51

Major: Applied Mathematics **Code:** SCI3053

Supervisor: Professor Vanni Noferini

Advisor: Professor Vanni Noferini

Solving systems of equations is a common problem in mathematical and physical applications. The solutions of an equation can be interpreted to be roots of a function, and hence solving a system of equations corresponds to finding the common roots of the associated functions. The system of functions can be approximated with polynomials, and the resulting multidimensional polynomial root-finding problem can be solved with the Cayley resultant method which transforms the root-finding problem into a polynomial eigenvalue problem. The goal of this thesis is to implement and analyse the accuracy of a functioning root-finding algorithm that solves a system of three equations in three variables by using the Cayley resultant method. To reach this goal, we first discuss background material about polynomial interpolation techniques, matrix polynomials, polynomial eigenvalue problems, Chebyshev polynomials, and multidimensional resultants. We explain how the Cayley resultant method transforms the system of polynomial equations into a polynomial eigenvalue problem, and consider the conditioning of this transformed problem in relation to the conditioning of the root-finding problem. We describe in detail how the root-finder is implemented in practice, and characterize the accuracy of the implementation by solving example problems as well as running the algorithm in different arithmetic precisions. These numerical examples demonstrate in practice the deterioration of the conditioning caused by the Cayley resultant method, as well as the inability of the implementation to handle high-degree inputs accurately in a manageable amount of time. We address the poor numerical accuracy by showing how Newton's method can be used to polish inaccurate roots, and explain how future implementations employ domain subdivision in order to handle high-degree inputs accurately. We conclude that while the Cayley resultant method can lead to poor numerical stability, the implementation can still yield highly accurate solutions for a wide range of problems with the help of additional post-processing of the roots via Newton's method.

Keywords: Cayley resultant method, Chebyshev basis, root-finding

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Master's Programme in Mathematics and Operations Research

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Lauri Nyman		
Työn nimi:	Kolmen kolmimuuttujaisen yhtälön yhtälöryhmän ratkaisu resultanttien avulla		
Päiväys:	22. helmikuuta 2021	**Sivumäärä:**	v + 51
Pääaine:	Applied Mathematics	**Koodi:**	SCI3053
Valvoja:	Professori Vanni Noferini		
Ohjaaja:	Professori Vanni Noferini		

Yhtälöryhmien ratkaiseminen on yleinen ongelma matemaatiikan ja fysiikan sovelluskohteissa. Yhtälön ratkaisu voidaan tulkita funktion nollakohdaksi, ja näin ollen yhtälöryhmän ratkaiseminen vastaa funktioryhmän yhteisten nollakohtien löytämistä. Ryhmä funktioita voidaa approksimoida polynomien avulla, ja näiden polynomien nollakohdat voidaan ratkaista Cayley-resultanttimenetelmän avulla, joka muuttaa ongelman nollakohtien etsinnästä polynomien ominaisarvo-ongelmaksi. Työn tavoitteena on toteuttaa toimiva nollakohtien ratkaisija, joka perustuu Cayley-resultanttimenetelmään, ja analysoida toteutuksen antamien ratkaisujen tarkkuutta. Toteutus pyrkii ratkaisemaan yhteiset nollakohdat kolmen kolmimuuttujaisen yhtälön yhtälöryhmälle. Päämäärän saavuttamiseksi työssä käydään läpi taustatietoa polynomien interpoloinnista, matriisipolynomeista, polynomien ominaisarvo-ongelmista, Chebyshev-polynomeista ja moniulotteisista resultanteista. Työssä selitetään, miten Cayley-resultanttimenetelmä muuttaa ongelman nollakohtien löytämisestä polynomien ominaisarvo-ongelmaksi, ja verrataan muunnetun ongelman kuntolukua alkuperäisen ongelman kuntolukuun. Työssä käydään yksityiskohtaisesti läpi, miten ratkaisin on toteutettu käytännössä, ja analysoidaan, millä tarkkuudella toteutus löytää erilaisten esimerkkiyhtälöryhmien nollakohdat. Lisäksi tarkastellaan aritmeettisen tarkkuuden vaikutusta laskettujen ratkaisujen tarkkuuteen. Näiden numeeristen esimerkkien avulla empiirisesti osoitetaan Cayley-resultanttimenetelmän aiheuttama heikkeneminen ongelman kuntoluvussa sekä toteutuksen kyvyttömyys ratkaista korkean asteen yhtälöryhmien nollakohtia tarkasti kohtuullisessa aikamäärässä. Työssä selitetään, miten toteutus käyttää Newtonin menetelmää tarkentamaan epätarkasti laskettuja nollakohtia ja miten tulevissa toteutuksissa käytetään määrittelyjoukon osittamista mahdollistamaan korkea-asteisten yhtälöryhmien nollakohtien ratkaiseminen hyvällä tarkkuudella. Loppupäätelmä on, että vaikka Cayley-resultanttimenetelmä johtaa heikentyneeseen numeeriseen stabiilisuuteen, voi toteutus laskea erilaisten yhtälöryhmien nollakohdat hyvinkin tarkasti hyödyntäen Newtonin menetelmää nollakohtien jälkikäsittelyssä.

Asiasanat:	Cayley-resultanttimenetelmä, Chebyshev-kanta, yhtälöiden ratkaiseminen
Kieli:	Englanti

Contents

Chapter 1

Introduction

Root-finding can be defined as the act of finding either zeros of univariate functions or common zeros of several multivariate functions. The problem of root-finding is ubiquitous in natural sciences and engineering, and for this reason is of high interest in a wide range of applications. There are different methods for root-finding such as homotopy continuation methods [2, 25], resultant methods [1, 3, 6], and contouring algorithms such as marching triangles [15]. In this work, we use the *Cayley resultant method* to find the common zeros of three trivariate functions. More specifically, the aim is to find the common zeros of three trivariate complex analytic functions in some real interval. It is the real roots that often have physical meaning in real-world applications, and hence they are the roots of interest in many contexts. For this reason, this thesis focuses on finding the real roots and employs tools that fit this purpose (e.g. Chebyshev interpolation, see Section 2.1).

The Cayley resultant method has been used to solve the bivariate version of this problem in [19]. This thesis aims to expand on this work and implement a root-finding algorithm in the trivariate setting. To our knowledge, no robust implementation of the trivariate Cayley resultant method has been published at the time of writing. We anticipate that this work will continue after the thesis and hopefully lead to a research paper.

1.1 Problem statement

In this thesis, we consider the problem of finding the common real roots of three trivariate functions f, g, and h. Any root-finding problem in some real interval can be transformed into a root-finding problem in the interval

$[-1, 1]$ through a change of variables. Hence, with no loss of generality we limit each variable to belong to the interval $[-1, 1]$, where the functions f, g, and h are possibly complex-valued and assumed to be analytic. In other words, the problem is to find all triplets $(x, y, z) \in \Omega = [-1, 1]^3$ s.t.

$$\begin{pmatrix} f(x, y, z) \\ g(x, y, z) \\ h(x, y, z) \end{pmatrix} = 0, \tag{1.1}$$

where the functions f, g, and h are analytic and possibly complex-valued in Ω. We further assume that the solution set is zero-dimensional; that is, the solutions to the problem are isolated. The Cayley resultant method cannot be used to characterize a higher-dimensional solution set, which becomes clear in Section 2.3 in which the Cayley resultant method is discussed. For many real-world applications, zero-dimensionality of the solution set is not a strong assumption and hence does not undermine the utility of a Cayley resultant method based root-finding algorithm.

The aim of the thesis is to use the Cayley resultant method to create a functional root-finding algorithm in MATLAB that can solve the given problem as accurately as possible.

1.2 Structure of the thesis

This thesis consists of six chapters: introduction, background, methods, implementation, numerical results, and conclusion. The chapter following the introduction is the background chapter, in which the relevant theory for solving the root-finding problem is explained. This includes theory about polynomial interpolation techniques, *matrix polynomials* and *polynomial eigenvalue problems*, *Chebyshev polynomials* and the *Chebyshev basis*. Importantly, this chapter explains the *Cayley resultant matrix* and how *multidimensional resultants* transform a system of polynomial equations into a polynomial eigenvalue problem.

In the methods chapter, we explain how the root-finding problem can be solved with the theory laid out in the previous chapter. While doing this, the chapter outlines a plan for a practical implementation of the root-finding algorithm. Here, we also consider the conditioning of the root-finding problem as well as how the proposed approach with the Cayley resultant maps the original problem to another one which is mathematically equivalent but may have a different condition number. The chapter following the methods chapter is the implementation chapter, which explains how the root-finding

algorithm was eventually implemented with a focus on the practical aspects of the implementation.

In the numerical results chapter, we provide numerical results that characterize the accuracy of the implementation. We demonstrate the problems with the accuracy of the root-finder by considering example problems in which the Cayley resultant approach results in a higher condition number than what the original root-finding problem has. To address the issue with inaccuracy, we show how inaccurate roots can be polished via Newton's method. We further demonstrate the need for future work in handling high-degree input functions as well as run the algorithm in different arithmetic precisions to explore how varying degrees of precision affect the accuracy of the output.

Finally, we conclude the thesis by summarising the aim, the findings and the future work in the conclusion chapter.

Chapter 2

Background

This chapter aims to deliver the required theoretical tools to use the Cayley resultant method to solve the root-finding problem in (1.1). The topics that require covering can be categorized under three subjects: polynomial interpolation, matrix polynomials, and resultant methods. We follow this structure in the presentation of the theoretical background, and divide this chapter into three sections, one to address each of the aforementioned subjects.

2.1 Polynomial interpolation

The goal is to solve the problem in (1.1) by using the Cayley resultant method, which will be explained in detail in Section 2.3. We will note already here that resultant methods can be used to find the shared roots of polynomials. Therefore, we need to transform the problem in (1.1) into polynomial form. To do this, we employ polynomial interpolation in order to approximate the functions in (1.1) by trivariate polynomials. The problem in (1.1) then becomes as follows: find all triplets $(x, y, z) \in \Omega$ s.t.

$$\begin{pmatrix} p_f(x, y, z) \\ p_g(x, y, z) \\ p_h(x, y, z) \end{pmatrix} = 0, \tag{2.1}$$

where p_f, p_g and p_h are polynomial approximations of the functions f, g and h, respectively. One way to approximate the system with polynomials is via Lagrange interpolation.

2.1.1 Lagrange interpolation

This section explains Lagrange interpolation in the form of two theorems. Theorem 2.1.1 defines Lagrange interpolation, while Theorem 2.1.2 gives a result for the accuracy of the Lagrange interpolant.

Theorem 2.1.1 (Lagrange interpolation [5, 7])
Let f be a real function that is continuous on some real interval $[a, b]$. For any $n+1$ points $x_1 < x_2 < ... < x_{n+1} \in [a, b]$, there exists a unique polynomial P_n of degree less than or equal to n such that

$$P_n(x_i) = f(x_i), \quad i = 1, ..., n + 1.$$

This polynomial is called the polynomial interpolant of f of order n, and is given by

$$P_n(x_i) = \sum_{i=1}^{n+1} f(x_i) L_i(x),$$

where $L_i(x)$ are the fundamental Lagrange interpolation polynomials given by

$$L_i(x) = \prod_{j=1, j\neq i}^{n+1} \frac{x - x_j}{x_i - x_j}.$$

The interpolant P_n defined as in Theorem 2.1.1 agrees with the function f at every interpolation point, but not necessarily anywhere else. Various sets of points can be used in performing Lagrange interpolation, and the set of points that is used determines the resulting polynomial interpolant for the function. However, not all choices of interpolation points result in an equally good polynomial approximation. Here, by a "good" interpolant we mean a polynomial interpolant that does not differ much from f in the uniform norm in the interval $[a, b]$. In other words, the "best" interpolant would minimize the quantity

$$\|R_n\| = \sup\{|f(x) - P_n(x)| : x \in [a, b]\}.$$

In fact, we can characterize this quantity via Theorem 2.1.2.

Theorem 2.1.2 (Error of Lagrange interpolant [5, 7])
Let f be as in Theorem 2.1.1. If f is $n+1$ times differentiable in (a,b), then for all $x \in [a, b]$ and for all choices of n+1 interpolation points $x_1 < x_2 < ... < x_{n+1} \in [a, b]$, there exists $\xi \in (a, b)$ s.t. the error between f and the

polynomial interpolant of order n is

$$R_n(x) = f(x) - P_n(x) = \frac{f^{n+1}(\xi)}{(n+1)!}\prod_{j=1}^{n+1}(x - x_j). \tag{2.2}$$

For equidistant interpolation points, the error R_n tends to exhibit large oscillations near the endpoints of the interval. This is referred to as *Runge's phenomenon*, and can lead to a very high value for the maximal discrepancy $\|R_n\|$.

The problem of finding the set of $n+1$ interpolation points that minimize the interpolation error $\|R_n\|$ is a difficult problem to solve in general as the minimizing set of nodes depends on the function f. Instead, we attempt to decrease the intensity of Runge's phenomenon by choosing the interpolation points such that

$$\sup\left\{\left|\prod_{j=1}^{n+1}(x - x_j)\right| : x \in [a, b]\right\} \tag{2.3}$$

is minimized.

2.1.2 Chebyshev polynomials and Chebyshev nodes

The goal of reducing Runge's phenomenon by minimizing (2.3) can be achieved by interpolating the function at the roots of the *Chebyshev polynomials*.

Definition 2.1.1 (Chebyshev polynomial of the first kind)
The Chebyshev polynomials of the first kind are defined as

$$T_n(x) = \cos\left(n\cos^{-1}(x)\right), \quad x \in [-1, 1], \quad n \in \mathbb{N}.$$

The set of Chebyshev polynomials $\{T_0, T_1, ...\}$ is a *degree-graded* polynomial basis of $\mathbb{C}[x]$. The term degree-graded means that T_j has degree j for all $j \in \mathbb{N}$. Although not evident from the definition, this result readily follows from the following proposition.

Proposition 2.1.3 (Polynomial representation of the Chebyshev polynomials [11])
For $n \geq 2$, the Chebyshev polynomials T_n satisfy the recurrence relation

$$T_n(x) = 2xT_{n-1} - T_{n-2}(x),$$

where $T_0(x) = 1$ and $T_1(x) = x$.

The roots of the Chebyshev polynomials of the first kind are referred to as *Chebyshev nodes*. The Chebyshev nodes of degree n are the roots of the nth degree Chebyshev polynomial, and are given as [11]

$$x_k = \cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, ..., n.$$

We denote the set of the Chebyshev nodes of degree n by H_n.

Theorem 2.1.4 (Smallest possible uniform norm [11])
Let

$$x_k = \cos\left(\frac{2k-1}{2(n+1)}\pi\right), \quad k = 1, ..., n+1,$$

be the elements of H_{n+1}. *The monic polynomial* $\widetilde{T}_{n+1}$ *defined by*

$$\widetilde{T}_{n+1}(x) = \prod_{k=1}^{n+1}(x - x_k)$$

has the smallest possible uniform norm in $[-1, 1]$ *in the sense that*

$$||\widetilde{T}_{n+1}||_\infty \leq ||p_{n+1}||_\infty$$

for any other monic polynomial p_{n+1} *of degree* $n + 1$. *Moreover,*

$$||\widetilde{T}_{n+1}(x)||_\infty = 2^{-n}.$$

It follows from Theorem 2.1.4 that interpolating at the Chebyshev nodes minimizes (2.3) and yields the desired polynomial interpolant in the interval $[-1, 1]$. From this result, it would be straightforward to derive the desired interpolation points in any interval $[a, b]$ by using an affine transformation that maps the Chebyshev nodes in the interval $[-1, 1]$ into the interval $[a, b]$. The resulting interpolation nodes would be $y_k = \frac{b-a}{2}x_k + \frac{a+b}{2}$, where x_k are the Chebyshev nodes. In this thesis however, we only consider polynomial interpolation in the interval $[-1, 1]$ as this makes the mathematical treatment of the topic as well as the practical implementation easier without losing in generality.

2.1.3 Optimality of interpolation nodes

The aim of reducing Runge's phenomenon by minimizing (2.3) was motivated by the problem of finding the set of $n + 1$ interpolation points that minimize the interpolation error $\|R_n\|$ of (2.2) for a given function f. This led to

the choice of the Chebyshev nodes as the interpolation nodes. Although the Chebyshev nodes do minimize (2.3), it is not clear how well they minimize the interpolation error $\|R_n\|$ of (2.2). The following example demonstrates that the Chebyshev nodes are not the minimizer of the interpolation error for all functions f.

Example 2.1.5
Let us approximate the function $f(x) = e^x$ with a degree 2 polynomial. Interpolating at the Chebyshev nodes $H_3 = \{-\frac{\sqrt{3}}{2}, 0, \frac{\sqrt{3}}{2}\}$ yields a polynomial interpolant whose error measured as the uniform norm in the interval $[-1, 1]$ is approximately 0.0565. Interpolating at competitor nodes $S_3 = \{-0.88, 0, 0.88\}$ yields a polynomial interpolant whose error is approximately 0.0519. In other words, the Chebyshev nodes do not minimize the interpolation error for the function f.

A related problem is to find the set of $n+1$ interpolation points that minimize the maximum possible interpolation error $\|R_n\|$ over all continuous functions f. As the goal of this work is to create a generic root-finding algorithm that can handle different user-specified input functions, minimizing the worst-case error is a suitable approach as well.

Theoretically, minimizing the worst-case error over all continuous functions corresponds to minimizing the *Lebesgue constant*. Although the minimizing nodes for the worst-case error are not known, it is known that the Lebesgue constant grows with the degree of the interpolant n as per $O(\log n)$ when the minimizing nodes are used [24, Section 2.5], and that the Chebyshev nodes yield similar logarithmic growth [24, Section 2.2]. The Chebyshev nodes are not necessarily the minimizing nodes as the constant factor can be significantly larger, but they provide a near-optimal choice of interpolation points that have significant practical benefits (see Section 2.1.4), and their use is thus well justified.

2.1.4 Interpolation in the Chebyshev basis

There exists a computationally efficient way of performing Lagrange interpolation at the Chebyshev nodes which utilizes the Fast Fourier Transform (FFT). This will naturally lead to a representation of the polynomial interpolant in the Chebyshev basis as opposed to the monomial basis.

2.1.4.1 Univariate case

Theorem 2.1.6 (Interpolation in the Chebyshev basis [17])
Using the Chebyshev nodes of degree n as interpolation points, the resulting polynomial interpolant of f is given in the Chebyshev basis as

$$P_n(x) = \sum_{i=0}^{n}{}' c_i T_i(x),$$

where the prime in the notation means that the first term in the summation is halved. The coefficients in the sum can be computed as

$$c_i(x) = \frac{2}{n+1} \sum_{k=0}^{n} f(x_k) T_i(x_k).$$

This is equivalent to a discrete Fourier transform of a function g given by

$$g(\theta) = f(\cos\theta).$$

Using the same definitions as in Theorem 2.1.6, the theorem tells us that we can find the coefficients c_i in the Chebyshev expansion by computing the coefficients in the discrete Fourier series of the transformed function g. This can be done in a computationally efficient way via FFT that has a computational complexity of $O(n \log n)$. In this way, we can perform polynomial interpolation in the Chebyshev nodes efficiently.

2.1.4.2 Multivariate case

Sometimes it is required to perform a polynomial interpolation for a multivariate function. To extend the concept of interpolating a function in the Chebyshev basis into higher dimensions, we need *multidimensional Chebyshev nodes* that are defined through a Cartesian product construction of standard Chebyshev nodes.

Definition 2.1.2 (Multidimensional Chebyshev nodes)
Let $n \geq 2$ and let H_{m_1}, H_{m_2}, ..., H_{m_n} be sets of Chebyshev nodes of degrees $m_1, m_2, \ldots, m_n$, respectively. The set of n-dimensional Chebyshev nodes associated with $H_{m_1}, H_{m_2}, \ldots, H_{m_n}$ is the Cartesian product $H_{m_1} \times H_{m_2} \times \cdots \times H_{m_n}$ which is denoted by $H_{m_1 \times m_2 \times \cdots \times m_n}$.

Theorem 2.1.6 naturally extends to the multivariate setting, where the multivariate function is interpolated at multidimensional Chebyshev nodes [9]. Performing n-dimensional interpolation at n-dimensional Chebyshev nodes

can be done by using the n-dimensional FFT (`fftn` in MATLAB) to efficiently find the coefficients in the n-dimensional Chebyshev expansion of the function.

Theorem 2.1.4 also extends to the multidimensional case in a straightforward manner. In other words, using multidimensional Chebyshev nodes as interpolation points minimizes the multidimensional analogue of (2.3) and gives the desired polynomial interpolant for an n-variate function in the domain $[-1, 1]^n$. The interpolating polynomial will have a degree less than or equal to $m_k - 1$ in the kth variable, $1 \leq k \leq n$.

2.2 Matrix polynomials and polynomial eigenvalue problems

In order to understand in Section 2.3 how the Cayley resultant method transforms a system of polynomial equations into a polynomial eigenvalue problem, we need to understand some key properties of matrix polynomials and polynomial eigenvalue problems.

In this writing, a matrix polynomial of degree l refers to a matrix-valued function $A : \mathbb{C} \to \mathbb{C}^{m\times n}$ s.t. $A(x) = \sum_{j=0}^{l} A_j x^j$ where $A_0, ..., A_l \in \mathbb{C}^{m\times n}$. If $A(x)$ is a square matrix polynomial and the leading coefficient matrix A_l is the identity matrix, we call $A(x)$ a *monic* matrix polynomial. The set of $m \times n$ matrix polynomials over the complex field in an indeterminate x is denoted by $\mathbb{C}[x]^{m\times n}$.

Matrix polynomials can equivalently be seen as matrices with polynomial elements. In this writing, we use the terms "matrix polynomial" and "polynomial matrix" interchangeably.

Proposition 2.2.1 (Existence of inverse)
For a square matrix polynomial $A(x) \in \mathbb{C}[x]^{n\times n}$, there exists an inverse matrix polynomial $A^{-1}(x) \in \mathbb{C}[x]^{n\times n}$ s.t. $A(x)A^{-1}(x) = I$ if and only if the determinant of $A(x)$ is a nonzero constant.

Proof. Let $A(x), A^{-1}(x) \in \mathbb{C}[x]^{n\times n}$ s.t. $A(x)A^{-1}(x) = I$. As the determinant is multiplicative, this implies that $\det A(x) \det A^{-1}(x) = 1_R$. By applying a Laplace expansion, it can be seen that the determinant of a matrix can be given as a sum of products of the elements. As all elements of $A(x)$ belong to $\mathbb{C}[x]$, it follows that $\det A(x)$ also belongs to $\mathbb{C}[x]$. Moreover, by a similar reasoning we have that $\det A^{-1}(x) \in \mathbb{C}[x]$. This implies that $\det A(x)$ is a

unit of $\mathbb{C}[x]$. The only units of $\mathbb{C}[x]$ are the nonzero constants. This proves one direction of the claim.

For the other direction, assume that $\det A(x)$ is a nonzero constant. Then, $(\det A(x))^{-1}$ adj $A(x)$ is an inverse of $A(x)$. □

Proposition 2.2.1 a well known result, the proof of which has previously been discussed for example in [20, p. 16].

Example 2.2.2
Consider the two matrix polynomials

$$A(x) = \begin{bmatrix} 1 & x \\ 0 & 1 \end{bmatrix} \in \mathbb{C}[x]^{2\times 2},$$
$$B(x) = \begin{bmatrix} x & 1 \\ 0 & x \end{bmatrix} \in \mathbb{C}[x]^{2\times 2}.$$

The determinant of $A(x)$ is 1 which is a nonzero constant; hence, $A(x)$ is invertible. One can check that the matrix polynomial $A^{-1}(x)$ given by

$$A^{-1}(x) = \begin{bmatrix} 1 & -x \\ 0 & 1 \end{bmatrix} \in \mathbb{C}[x]^{2\times 2}$$

satisfies $A(x)A^{-1}(x) = A^{-1}(x)A(x) = I$ and is thus the inverse of the matrix polynomial $A(x)$.

The determinant of $B(x)$ is x^2 which is not a nonzero constant. Hence, $B(x)$ is not invertible.

Definition 2.2.1 (Eigenvalue)
Let $A(x) \in \mathbb{C}[x]^{m\times n}$ and let $\mathbb{C}(x)$ be the field of rational functions over $\mathbb{C}$. We call $\lambda \in \mathbb{C}$ a finite eigenvalue of $A(x)$ if

$$\operatorname{rank}_{\mathbb{C}} A(\lambda) < \operatorname{rank}_{\mathbb{C}(x)} A(x).$$

The discussion of infinite eigenvalues of a matrix polynomial is omitted as only finite eigenvalues are of interest to us in the root-finding problem.

Example 2.2.3
Consider again the two matrix polynomials

$$A(x) = \begin{bmatrix} 1 & x \\ 0 & 1 \end{bmatrix} \in \mathbb{C}[x]^{2\times 2},$$
$$B(x) = \begin{bmatrix} x & 1 \\ 0 & x \end{bmatrix} \in \mathbb{C}[x]^{2\times 2}.$$

First, we note that

$$\text{rank}_{\mathbb{C}(x)} A(x) = \text{rank}_{\mathbb{C}(x)} B(x) = 2.$$

The determinant of $A(x)$ is 1 which is a nonzero constant. Hence, $A(x)$ is full rank for all $x \in \mathbb{C}$. This implies that for all values of $z \in \mathbb{C}$ it holds that

$$\text{rank}_{\mathbb{C}} A(z) = \text{rank}_{\mathbb{C}(x)} A(x),$$

and the matrix polynomial $A(x)$ has no finite eigenvalues.

For the matrix polynomial $B(x)$, it holds that

$$B(0) = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \in \mathbb{C}[x]^{2\times 2},$$

which is of rank 1. Hence,

$$\text{rank}_{\mathbb{C}} B(0) < \text{rank}_{\mathbb{C}(x)} B(x),$$

and 0 is an eigenvalue of $B(x)$.

Definition 2.2.2 (Equivalence)
Two matrix polynomials $A(x) \in \mathbb{C}[x]^{m\times n}$ and $B(x) \in \mathbb{C}[x]^{m\times n}$ are called equivalent if there exist matrix polynomials $E(x) \in \mathbb{C}[x]^{m\times m}$ and $F(x) \in \mathbb{C}[x]^{n\times n}$ with constant nonzero determinants s.t.

$$A(x) = E(x)B(x)F(x).$$

If two matrix polynomials $A(x)$ and $B(x)$ are equivalent, we write $A(x) \sim B(x)$.

In particular, the finite eigenvalues of equivalent matrix polynomials are the same. To see this, first note that the square matrix polynomials $E(x)$ and $F(x)$ of Definition 2.2.2 are full rank over $\mathbb{C}(x)$ as their determinants are not identically zero. Hence, $A(x)$ and $B(x)$ are the same rank over $\mathbb{C}(x)$. Moreover, for all $z \in \mathbb{C}$ the matrices $E(z)$ and $F(z)$ are full rank as their determinants are nonzero, and hence $A(z)$ and $B(z)$ have the same rank for all $z \in \mathbb{C}$. It then follows from Definition 2.2.1 that the finite eigenvalues of the matrix polynomials $A(x)$ and $B(x)$ coincide.

By Theorem 2.2.4 below, every matrix polynomial is equivalent with a specific type of diagonal matrix polynomial called the *Smith normal form*.

Theorem 2.2.4 (Smith normal form [13, Theorem S1.1])
For every matrix polynomial $A(x) \in \mathbb{C}[x]^{m\times n}$, there exist matrix polynomials $E(x) \in \mathbb{C}[x]^{m\times m}$ and $F(x) \in \mathbb{C}[x]^{n\times n}$ with constant nonzero determinants s.t.

$$E(x)A(x)F(x) = S(x),$$

where $S(x) \in \mathbb{C}[x]^{m\times n}$ is a diagonal matrix polynomial with $r \leq n$ non-zero entries:

$$S(x) = \begin{bmatrix} s_1(x) & & \dots & \dots & 0 \\ & \ddots & & & \\ \vdots & & s_r(x) & & \vdots \\ \vdots & & & 0 & \vdots \\ & & & & \ddots \\ 0 & & \dots & \dots & 0 \end{bmatrix}.$$

Moreover, each diagonal element of $S(x)$ divides its successor, that is, $s_i(x) \mid s_{i+1}(x)$ for all $i = 1, 2, \dots, r-1$. Such a matrix polynomial $S(x)$ is called the Smith normal form of $A(x)$.

Example 2.2.5
Consider the matrix polynomial

$$A(x) = \begin{bmatrix} 1 & x \\ x & 2x^2 \end{bmatrix} \in \mathbb{C}[x]^{2\times 2}.$$

The matrix polynomial $A(x)$ can be expressed in terms of its Smith normal form as

$$A(x) = \begin{bmatrix} 1 & 0 \\ x & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & x^2 \end{bmatrix} \begin{bmatrix} 1 & x \\ 0 & 1 \end{bmatrix},$$

where the matrix polynomials

$$\begin{bmatrix} 1 & 0 \\ x & 1 \end{bmatrix}, \begin{bmatrix} 1 & x \\ 0 & 1 \end{bmatrix}$$

have constant nonzero determinants. This being the case, it also holds that the matrix polynomial $A(x)$ is equivalent with its Smith normal form

$$S_P(x) = \begin{bmatrix} 1 & 0 \\ 0 & x^2 \end{bmatrix}.$$

The Smith normal form can be used to prove Corollary 2.2.6 relating to the finite eigenvalues of a *regular* matrix polynomial, that is, a square matrix polynomial whose determinant is not identically zero.

Corollary 2.2.6
The finite eigenvalues of a regular matrix polynomial $A(x)$ are the zeros of its determinant $\det A(x)$.

Proof. Let $A(x) \in \mathbb{C}[x]^{n\times n}$ be a regular matrix polynomial, and let $S(x)$ be its Smith normal form. By using the multiplicative property of the determinant, it follows that

$$\det A(x) = \det E(x) \det S(x) \det F(x)$$

for some matrix polynomials $E(x)$ and $F(x)$ s.t. $\det E(x) = c_1 \in \mathbb{C}$ and $\det F(x) = c_2 \in \mathbb{C}$. The equation can then be expressed as

$$\det A(x) = c_1 c_2 \prod_{i=1}^{n} s_i(x).$$

As $A(x)$ is a regular matrix polynomial and hence its determinant is not identically zero, it follows that its Smith normal form $S(x)$ cannot have any zero diagonal elements. Hence, $S(x)$ is full rank over $\mathbb{C}(x)$, that is, $\text{rank}_{\mathbb{C}(x)} S(x) = n$. The eigenvalues of $S(x)$ are those values of $\lambda \in \mathbb{C}$ for which

$$\text{rank}_{\mathbb{C}} S(\lambda) < \text{rank}_{\mathbb{C}(x)} S(x) = n.$$

This happens if and only if $\det S(\lambda) = 0$. Hence, the eigenvalues of $S(x)$ are the zeros of $\det S(x)$. The finite eigenvalues of equivalent matrices coincide, and hence the finite eigenvalues of the regular matrix polynomial $A(x)$ are those of its Smith normal form $S(x)$, that is, the zeros of $\det S(x)$. Moreover, as it holds that

$$\det A(x) = c_1 c_2 \det S(x),$$

the zeros $\det S(x)$ are the zeros of $\det A(x)$. Hence, the finite eigenvalues of a regular matrix polynomial $A(x)$ are the zeros of $\det A(x)$. □

Corollary 2.2.7
$\lambda \in \mathbb{C}$ is a finite eigenvalue of a regular matrix polynomial $A(x)$ if and only if there exists a nonzero vector v such that $A(\lambda)v = 0$.

Proof. Let $A(x) \in \mathbb{C}[x]^{n\times n}$ be a regular matrix polynomial, and let $\lambda \in \mathbb{C}$. By Corollary 2.2.6, the finite eigenvalues of $A(x)$ are the zeros of $\det A(x)$. It is widely known that $\det A(\lambda) = 0$ if and only if the columns of $A(\lambda)$ are

linearly dependent. In other words, $\det A(\lambda) = 0$ if and only if there exist $\gamma_i \in \mathbb{C},\ i = 1, 2, \ldots, n$ such that

$$\sum_{i=1}^{n} \gamma_i a_{*,i}(\lambda) = 0,$$

where $a_{*,i}(\lambda)$ denotes the ith column of $A(\lambda)$, and

$$v := \begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_n \end{bmatrix} \neq 0.$$

Moreover,

$$\sum_{i=1}^{n} \gamma_i a_{*,i}(\lambda) = 0 \iff A(\lambda) \begin{bmatrix} \gamma_1 \\ \vdots \\ \gamma_n \end{bmatrix} = 0.$$

This shows that $\det A(\lambda) = 0$ if and only if there exists a nonzero $v \in \mathbb{C}^n$ s.t. $A(\lambda)v = 0$. In other words, λ is a finite eigenvalue of $A(x)$ if and only if there exists a nonzero $v \in \mathbb{C}^n$ s.t. $A(\lambda)v = 0$. □

The preceding corollaries give two characterizations for a finite eigenvalue of a regular matrix polynomial that are simpler to work with than the generic definition given by Definition 2.2.1.

Definition 2.2.3 (Linearization)
Let $A(x) \in \mathbb{C}[x]^{n\times n}$ with $\deg A(x) > 1$. A linear matrix polynomial $L(x) \in \mathbb{C}[x]^{(n+p)\times(n+p)}$ with $\deg L(x) = 1$ is called a linearization of the matrix polynomial $A(x)$ if

$$L(x) \sim \begin{bmatrix} A(x) & 0 \\ 0 & I_p \end{bmatrix}.$$

From the fact that eigenvalues of equivalent matrix polynomials coincide, it follows that the eigenvalues of a matrix polynomial $A(x)$ coincide with the eigenvalues of its linearization. In this way, we can characterize the eigenstructure of a higher degree matrix polynomial $A(x)$ by a linear matrix polynomial of larger size.

Let $A(x) = \sum_{j=0}^{l} A_j x^j \in \mathbb{C}[x]^{n\times n}$ be a matrix polynomial of degree l. Its linearization is of degree one and can thus be represented as $L(x) = L_1 x - L_0$.

One example of a linearization of $A(x)$ is given by defining L_1 and L_0 as [12]

$$L_1 = \begin{bmatrix} I_{(l-1)n} & 0 \\ 0 & A_l \end{bmatrix}$$

and

$$L_0 = \begin{bmatrix} 0 & I & \dots & 0 & 0 \\ 0 & 0 & \dots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & I \\ -A_0 & -A_1 & \dots & -A_{l-2} & -A_{l-1} \end{bmatrix}.$$

If we define

$$\widetilde{A}_{1:l-1} = \begin{bmatrix} A_1 & \dots & A_{l-2} & A_{l-1} \end{bmatrix},$$

then we can write the linearization more compactly as

$$L_1 x - L_0 = \begin{bmatrix} I_{(l-1)n} & 0 \\ 0 & A_l \end{bmatrix} x - \begin{bmatrix} 0_{(l-1)n \times n} & I_{(l-1)n} \\ -A_0 & -\widetilde{A}_{1:l-1} \end{bmatrix}. \tag{2.4}$$

The linearization $L_1 x - L_0$ in (2.4) is called the *companion pencil* of the matrix polynomial $A(x)$.

The intention is to find the eigenvalues of the matrix polynomial $A(x)$ by finding the eigenvalues of its linearization. The eigenvalues of the linearization $L_1 x - L_0$ can be computed by solving the generalized eigenvalue problem $L_0 v = L_1 x v$. This can be done in a *backward stable* manner (see Section 3.4) by using the QZ-algorithm implemented in MATLAB's `eig` function [26].

In some situations, it is needed to find a linearization of a matrix polynomial expressed in the Chebyshev basis (see Section 2.1.2). We can express the matrix polynomial $A(x)$ in the Chebyshev basis as $A(x) = \sum_{j=0}^{l} A'_j T_j(x)$. It is possible to construct a linearization of $A(x)$ expressed in terms of the coefficient matrices A'_j. One example of this type of linearization is [28]

$$\begin{aligned} L'_1 x - L'_0 = &\begin{bmatrix} A'_l & 0 \\ 0 & I_{(l-1)n} \end{bmatrix} x \\ &-\frac{1}{2} \begin{bmatrix} -A'_{l-1} & I_n - A'_{l-2} & -A'_{l-3} & \dots & -A'_0 \\ I_n & 0 & I_n & & \\ & \ddots & \ddots & \ddots & \\ & & I_n & 0 & I_n \\ & & & 2I_n & 0 \end{bmatrix}. \end{aligned} \tag{2.5}$$

The linearization $L_1'x - L_0'$ in (2.5) is called the *colleague pencil* of the matrix polynomial $A(x)$.

In this work, we will mostly use the colleague pencil for linearization. It was recently proved for scalar polynomials, which can be seen as 1×1 matrix polynomials, that linearizing via the colleague pencil yields a backward stable method to compute the roots [18]. At the time of writing, we are not aware whether this result extends to computing the eigenvalues of matrix polynomials of larger size.

2.3 Resultant methods

Hidden variable resultant methods are a class of methods to solve polynomial root-finding problems. Our goal is to solve the trivariate problem given in (2.1); hence, we will motivate the use of resultant methods in the context of this trivariate system.

2.3.1 The use of resultants in root-finding

The first step in hidden variable resultant methods is *hiding* the last variable, which is done by considering a trivariate polynomial $p(x, y, z)$ as a bivariate polynomial $p(x, y)$ with z-dependent coefficients. If p has maximal degree n in each variable, then

$$p(x, y, z) = \sum_{i_1, i_2, i_3 = 0}^{n} \alpha_{i_1 i_2 i_3} x^{i_1} y^{i_2} z^{i_3} = \sum_{i_1, i_2 = 0}^{n} \alpha_{i_1 i_2}(z) x^{i_1} y^{i_2} = p[z](x, y).$$

In principle, any variable could be hidden, but to simplify the exposition, we will always hide the last variable.

We will first find all $z^* \in \mathbb{C}$ s.t. the bivariate polynomials $p_f[z^*]$, $p_g[z^*]$ and $p_h[z^*]$ of (2.1) have at least one common root. Then, we find all $y^* \in \mathbb{C}$ s.t. the univariate polynomials $p_f[y^*, z^*]$, $p_g[y^*, z^*]$ and $p_h[y^*, z^*]$ have at least one common root. The univariate root-finding problems can then be treated as polynomial eigenvalue problems of 1×1 matrix polynomials, which can be solved through the linearization given in (2.4) or (2.5). Linearization transforms the problem into a generalized eigenvalue problem which can be solved by using a standard QZ-algorithm. In this way, we can find the solutions one component at a time.

The remaining question then is, how can we find all $z^* \in \mathbb{C}$ s.t. the bivariate polynomials $p_f[z^*]$, $p_g[z^*]$ and $p_h[z^*]$ have at least one common root, or find

all $y^* \in \mathbb{C}$ s.t. the univariate polynomials $p_f[y^*, z^*]$, $p_g[y^*, z^*]$ and $p_h[y^*, z^*]$ have at least one common root? To do this, we need a *multidimensional resultant* [10, Ch. 13].

Let us denote the ring of d-variate polynomials with complex coefficients of maximal degree n in variables $x_1, ..., x_d$ by $\mathbb{C}_n[x_1, ..., x_d]$.

Definition 2.3.1 (Multidimensional resultant as defined in [21])
Let $d \geq 2$ and $n \geq 0$. A functional $T : (\mathbb{C}_n[x_1, ..., x_{d-1}])^d \to \mathbb{C}$ is a multidimensional resultant if, for any set of d polynomials $q_1, ..., q_d \in \mathbb{C}_n[x_1, ..., x_{d-1}]$, $T(q_1, ..., q_d)$ is a polynomial in the coefficients of $q_1, ..., q_d$ and $T(q_1, ..., q_d) = 0$ if and only if $\exists x^* \in \tilde{\mathbb{C}}^{d-1}$ s.t.

$$\begin{pmatrix} q_1(x^*) \\ \vdots \\ q_d(x^*) \end{pmatrix} = 0,$$

where $\tilde{\mathbb{C}}$ denotes the extended complex plane (see Section 2.3.3 for explanation of roots at infinity).

As we are considering a system of three polynomials, we are interested in the case $d = 3$ in Definition 2.3.1. Notice that $p_f[z], p_g[z], p_h[z] \in \mathbb{C}_n[x, y]$ for some $n \in \mathbb{N}$. If, for some multidimensional resultant $T : (\mathbb{C}_n[x, y])^3 \to \mathbb{C}$ it holds that $T(p_f[z], p_g[z], p_h[z]) = 0$, then $\exists (x^*, y^*) \in \tilde{\mathbb{C}}^2$ s.t.

$$\begin{pmatrix} p_f[z](x^*, y^*) \\ p_g[z](x^*, y^*) \\ p_h[z](x^*, y^*) \end{pmatrix} = 0.$$

In other words, the zeros of the three-dimensional resultant give us the desired values for z^*. As we are looking for solutions in $\Omega = [-1, 1]^3$, we can discard all zeros outside Ω. In a similar way, we can use a two-dimensional resultant to find the desired values for y^*.

However, using multidimensional resultants in the described way leads to a practical problem: it can be very numerically unstable to directly compute the zeros of a multidimensional resultant. For numerical stability, it is recommended to consider a matrix whose determinant is a multidimensional resultant (see [21]). Such a matrix is called a *multidimensional resultant matrix*. After hiding a variable, the entries of the multidimensional resultant matrix become univariate polynomials in the hidden variable. The zeros of the resultant are then the eigenvalues of the multidimensional resultant matrix polynomial, which can be computed by using a more numerically stable method such as the QZ-algorithm.

In the introduction to this thesis, we made a remark that the Section 2.3 would shed light on why the solution set to the problem is assumed to be zero-dimensional. The fact that the eigenvalues of a specific type of matrix polynomial are used to yield the desired roots makes this evident. As the amount of eigenvalues of a matrix polynomial of a finite size is always finite, the method clearly cannot capture the solutions when the roots are not isolated.

In the next section, we show how to construct a specific type of multidimensional resultant matrix, called the *Cayley resultant matrix*.

2.3.2 The Cayley resultant

The Cayley resultant [4] can be defined through the *Cayley function*.

Definition 2.3.2 (Cayley function as defined in [21])
Let $q_1, ..., q_d \in \mathbb{C}_n[x_1, ..., x_{d-1}]$ be polynomials that all have the same degree in each variable. The Cayley function associated with polynomials $q_1, ..., q_d$ is a multivariate polynomial in $2d-2$ variables, denoted by $f_{Cayley} = f_{Cayley}(q_1, ..., q_d)$, and is given by

$$f_{Cayley} = \left(\prod_{i=1}^{d-1}(s_i - t_i)\right)^{-1} \det \begin{pmatrix} q_1(s_1, s_2, ..., s_{d-1}) & \dots & q_d(s_1, s_2, ..., s_{d-1}) \\ q_1(t_1, s_2, ..., s_{d-1}) & \dots & q_d(t_1, s_2, ..., s_{d-1}) \\ \vdots & \ddots & \vdots \\ q_1(t_1, t_2, ..., t_{d-1}) & \dots & q_d(t_1, t_2, ..., t_{d-1}) \end{pmatrix}.$$

By applying the Laplace expansion for the Cayley function, we can see that the degree of f_{Cayley} is the same in both s_k and t_{d-k} for all $1 \leq k \leq d-1$. Let us denote the degree corresponding to both s_k and t_{d-k} by τ_k. Based on the Laplace expansion, it also holds that $\tau_k \leq kn - 1$ for all $1 \leq k \leq d-1$. Let $\{\phi_0, \phi_1, ...\}$ be a degree-graded polynomial basis (i.e. ϕ_j has degree j for all j) of $\mathbb{C}[x]$. The Cayley function can be expanded as

$$f_{Cayley} = \sum_{i_1=0}^{\tau_1} \cdots \sum_{i_{d-1}=0}^{\tau_{d-1}} \sum_{j_1=0}^{\tau_{d-1}} \cdots \sum_{j_{d-1}=0}^{\tau_1} A_{i_1,...,i_{d-1},j_1,...,j_{d-1}} \prod_{k=1}^{d-1} \phi_{i_k}(s_k) \prod_{k=1}^{d-1} \phi_{j_k}(t_k), \tag{2.6}$$

where A is a tensor of size $(\tau_1+1) \times \cdots \times (\tau_{d-1}+1) \times (\tau_{d-1}+1) \times \cdots \times (\tau_1+1)$.

The *Cayley resultant matrix* is defined as an unfolding of the tensor A in the following way (see [23, Sec. 2.3] for an unfolding of a tensor).

Definition 2.3.3 (Cayley resultant matrix as defined in [21])
Let $\{\phi_0, \phi_1, ...\}$ *be a degree-graded basis of* $\mathbb{C}[x]$*, and let* $q_1, ..., q_d \in \mathbb{C}_n[x_1, ..., x_{d-1}]$ *have the same degree in each variable. The Cayley resultant matrix associated with* $q_1, ..., q_d$ *with respect to the basis* $\{\phi_0, \phi_1, ...\}$ *is the* $\left(\prod_{k=1}^{d-1}(\tau_k + 1)\right) \times \left(\prod_{k=1}^{d-1}(\tau_k + 1)\right)$ *matrix formed by the unfolding of the tensor* A *in (2.6). This matrix is denoted by* R_{Cayley}*.*

As for any other resultant matrix, it holds that the determinant of the Cayley resultant matrix, $\det R_{Cayley}$, is identically zero if and only if the system of polynomial equations

$$\begin{pmatrix} q_1(x_1, ..., x_{d-1}) \\ \vdots \\ q_d(x_1, ..., x_{d-1}) \end{pmatrix} = 0$$

has a solution in $\tilde{\mathbb{C}}^{d-1}$ (i.e. infinite roots are also registered).

2.3.3 Roots at infinity

To understand what it means to have a root at infinity, we need to consider *homogeneous coordinates*. The set of homogeneous coordinates corresponding to a point $(x_1, x_2, ..., x_{d-1}) \in \mathbb{C}^{d-1}$ is any set of d coordinates of the form $(x_1X, x_2X, ..., x_{d-1}X, X)$, where $X \in \mathbb{C}\backslash\{0\}$.

Next, we will see how homogeneous coordinates and the roots of a polynomial are connected. For a polynomial $p(x_1, x_2, ..., x_{d-1})$ of total degree n, we can define a homogeneous polynomial in d variables as

$$\tilde{p}(x_1, x_2, ..., x_{d-1}, X) = X^n \; p\left(\frac{x_1}{X}, \frac{x_2}{X}, ..., \frac{x_{d-1}}{X}\right).$$

The polynomial $\tilde{p}$ has the same degree n in each term. It is straightforward to see that if $x^* = (x_1^*, x_2^*, ..., x_{d-1}^*)$ is a root of p, then any set of homogeneous coordinates corresponding to x^* is a root of $\tilde{p}$. The converse is also true: if $\tilde{x}^* = (\tilde{x}_1^*, \tilde{x}_2^*, ..., \tilde{x}_{d-1}^*, X^*)$ is a root of $\tilde{p}$, where $X^* \neq 0$, then $\left(\frac{\tilde{x}_1^*}{X^*}, \frac{\tilde{x}_2^*}{X^*}, ..., \frac{\tilde{x}_{d-1}^*}{X^*}\right)$ is a root of p. However, it is also possible that $\tilde{p}$ has a nonzero root $\tilde{x}^* = (\tilde{x}_1^*, \tilde{x}_2^*, ..., \tilde{x}_{d-1}^*, X^*)$ with $X^* = 0$. In such a case, we say that the polynomial p has a *root at infinity* associated with the root $\tilde{x}^* = (\tilde{x}_1^*, \tilde{x}_2^*, ..., \tilde{x}_{d-1}^*, 0)$ (or any of its nonzero scalar multiples) of the homogeneous polynomial $\tilde{p}$. We say that two or more polynomials share a root at infinity if their associated homogeneous polynomials share a nonzero root with dth component equal to zero.

Example 2.3.1
Consider the bivariate polynomial $p(x, y) = x^2 + y^2 - 1$ *whose roots define the unit circle. The corresponding homogeneous polynomial is* $\tilde{p}(x, y, z) = x^2 + y^2 + z^2$*. It is easy to verify that the point* $(1, i, 0)$ *is a root of* $\tilde{p}$*. Hence,* p *has a root at infinity.*

Chapter 3

Methods

With the help of the tools presented in Chapter 2, we now propose a method to solve the problem in (1.1). The details of each step of the method are explained in this chapter while keeping in mind the ultimate aim of creating a MATLAB function that computes the solutions to the problem. At the end of the chapter, we consider how the presented method affects the conditioning of the root-finding problem. We begin the chapter by giving an overall idea of the proposed method in the form of an outline.

3.1 Method outline

First, we interpolate the system in the Chebyshev basis to receive a polynomial approximation. Then, we hide the z-variable and compute the Cayley resultant matrix polynomial for the polynomial system. We compute the eigenvalues of this matrix polynomial through linearization. These eigenvalues correspond to the z-components of the solutions. We substitute these back into the polynomials, and apply the Cayley resultant method now to the resulting bivariate systems. In this way, we compute the shared roots one component at a time.

3.2 Polynomial interpolation

As explained in Section 2.3, the Cayley resultant can be used to solve a polynomial root-finding problem. Hence, we will first transform the system of functions into a system of polynomials by using the Lagrange interpolation, which will yield a polynomial approximation of the system (this choice is in

fact temporary; see Section 4.2). The problem in (1.1) then becomes the one given in (2.1), where p_f, p_g and p_h are the polynomial approximations of the functions f, g and h, respectively.

The orders of the polynomial interpolants p_f, p_g and p_h will be user-specified. The user will input maximal degrees n_f, n_g and n_h for the polynomials p_f, p_g and p_h, respectively. The interpolation nodes are then the three-dimensional Chebyshev nodes $H_{(n_f+1)\times(n_f+1)\times(n_f+1)}$, $H_{(n_g+1)\times(n_g+1)\times(n_g+1)}$, and $H_{(n_h+1)\times(n_h+1)\times(n_h+1)}$ for p_f, p_h, and p_g, respectively.

Polynomial interpolation at the Chebyshev nodes provides a reasonably good interpolant, as explained in Section 2.1.3. An efficient implementation of this is through the FFT, which yields the polynomial expansion in the Chebyshev basis, as explained in Theorem 2.1.6. To perform this in practice, we use the `fftn` function in MATLAB.

Now, p_f, p_g and p_h belong to $\mathbb{C}_n[x, y, z]$ for some n. In order to use the Cayley resultant method, the degrees need to be the same in both x and y of each polynomial (see Definition 2.3.3). Notice that we do not have this condition for z as this will be the first hidden variable. In the scope of this thesis, this limitation is addressed by adding small perturbations to the polynomials. In other words, we add to p_f the monomials x^{n_f}, y^{n_f} and z^{n_f} each multiplied by some very small but nonzero value, and we do the same for p_g and p_h (now with monomials $x^{n_g}, y^{n_g}, z^{n_g}$ and $x^{n_h}, y^{n_h}, z^{n_h}$, respectively). By abuse of notation, we call these perturbed polynomials p_f, p_g and p_h, and now they have the same degree in both x and y (and in z, which was not required; the perturbation was done in each variable for even treatment of the variables). These polynomials represent a slightly different system; hence, the solutions only approximate those of the original polynomial problem.

For future implementations, we will try to remove the condition on the degrees of x and y in order to be able to deal with any possible input without the need for perturbation of the polynomials. This requires more thorough understanding of the Cayley resultant matrix and its construction. For example, the suggestions in [16] might remove the need to have such a restriction on the degree structures of the polynomials.

3.3 Solving the polynomial root-finding problem

We can construct the Cayley resultant matrix polynomial for the polynomial system in (2.1) by first hiding the variable z. In this case $d = 3$, and the Cayley function is a polynomial in four variables:

$$f_{Cayley} = \frac{1}{(s_1 - t_1)(s_2 - t_2)} \det \begin{pmatrix} p_f[z](s_1, s_2) & p_g[z](s_1, s_2) & p_h[z](s_1, s_2) \\ p_f[z](t_1, s_2) & p_g[z](t_1, s_2) & p_h[z](t_1, s_2) \\ p_f[z](t_1, t_2) & p_g[z](t_1, t_2) & p_h[z](t_1, t_2) \end{pmatrix}, \tag{3.1}$$

which can be expanded as

$$f_{Cayley} = \sum_{i_1=0}^{\tau_1} \sum_{i_2=0}^{\tau_2} \sum_{j_1=0}^{\tau_2} \sum_{j_2=0}^{\tau_1} A_{i_1,i_2,j_1,j_2}[z] T_{i_1}(s_1) T_{i_2}(s_2) T_{j_1}(t_1) T_{j_2}(t_2),$$

where $A[z]$ is a tensor of size $(\tau_1 + 1) \times (\tau_2 + 1) \times (\tau_2 + 1) \times (\tau_1 + 1)$ with entries in $\mathbb{C}[z]$.

In practice, we perform this expansion by interpolating the multivariate polynomial f_{Cayley} in an efficient way via FFT. By our earlier observations in Section 2.3, the degree of f_{Cayley} in both s_1 and t_2 is $\tau_1 \leq n - 1$, and the degree in both s_2 and t_1 is $\tau_2 \leq 2n - 1$, where n is the maximum of the degrees n_f, n_g and n_h. In order to interpolate f_{Cayley}, it is required to consider the degree in the hidden variable z as well. By applying the Laplace expansion, it is quite straightforward to see that the degree of f_{Cayley} in z is $\tau_3 \leq 3n$. In fact, the values for all of τ_1, τ_2 and τ_3 can be readily computed from n_f, n_g and n_h, and one sees that

$$\begin{aligned} \tau_1 &= \max\{n_f, n_g, n_h\} - 1, \\ \tau_2 &= \max\{n_f + n_g, n_f + n_h, n_g + n_h\} - 1, \\ \tau_3 &= n_f + n_g + n_h. \end{aligned}$$

Interpolating f_{Cayley} in the five-dimensional Chebyshev nodes $H_{(\tau_1+1)\times(\tau_2+1)\times(\tau_2+1)\times(\tau_1+1)\times(\tau_3+1)}$ interpolates f_{Cayley} exactly; in other words, the polynomial interpolant is equal to f_{Cayley}. Lagrange interpolation at the Chebyshev nodes can be performed via FFT (see Section 2.1.4). In this way, we are able to compute the tensor $A[z]$ efficiently.

The decision to compute f_{Cayley} through polynomial interpolation as opposed to an analytical calculation was done based on two qualities that are

important for any numerical method: efficiency and accuracy. Polynomial interpolation is efficient by virtue of FFT, and we know that polynomial interpolation will yield an exact polynomial interpolant given enough interpolation nodes. Hence, polynomial interpolation is a satisfactory choice for expanding the Cayley function. It may also be possible to compute the Cayley function analytically, but as of now, we are not aware of how to do this. For a future implementation, the prospect of expanding the Cayley function analytically will be considered in more detail.

There is one practical problem in the interpolation of f_{Cayley}, however. Whenever it happens that $s_1 = t_1$ or $s_2 = t_2$ in (3.1), the denominator becomes zero and f_{Cayley} is not defined. This must happen for some of the interpolation points in $H_{(\tau_1+1)\times(\tau_2+1)\times(\tau_2+1)\times(\tau_1+1)\times(\tau_3+1)}$ when the Chebyshev nodes $H_{(\tau_1+1)}$ and $H_{(\tau_2+1)}$ overlap. However, we know that it is possible to divide the determinantal expression in (3.1) by $(s_1 - t_1)(s_2 - t_2)$, and to be precise, f_{Cayley} in fact denotes this resulting polynomial. Hence, f_{Cayley} should be continuous even at points where $s_1 = t_1$ or $s_2 = t_2$. We can compute the values at these points by using the L'Hospital's rule on (3.1) and then evaluating the resulting expression in these problematic points. With this additional step, we are able to perform the interpolation and compute the tensor $A[z]$. The Cayley resultant matrix polynomial $R_{Cayley}[z]$ is then the $((\tau_1 + 1)(\tau_2 + 1)) \times ((\tau_1 + 1)(\tau_2 + 1))$ matrix with entries in $\mathbb{C}[z]$ formed by the unfolding of the tensor $A[z]$.

The Cayley resultant matrix polynomial $R_{Cayley}[z]$ is singular if and only if $\exists(x^*, y^*) \in \tilde{\mathbb{C}}^2$ s.t.

$$\begin{pmatrix} p_f[z](x^*, y^*) \\ p_g[z](x^*, y^*) \\ p_h[z](x^*, y^*) \end{pmatrix} = 0.$$

Hence, the eigenvalues of the matrix polynomial $R_{Cayley}[z]$ give all the z-components of the solutions to (2.1). Since we use the Chebyshev basis for $R_{Cayley}[z]$, it is more convenient to compute the eigenvalues through the colleague pencil given in (2.5). The computation of the eigenvalues is done by applying the QZ-algorithm (the `eig` function in MATLAB) to the generalized eigenvalue problem corresponding to this linearization.

Let S_z be the set that contains all the z-components of the solutions. In other words, $S_z = \{z^* \in \Omega : R_{Cayley}[z^*] = 0\}$. We will substitute all $z^* \in S_z$ back into the equations and next hide the variable y. In other words, we will consider the following problem: for all $z^* \in S_z$, find all the values of y such

that

$$\begin{pmatrix} p_f[z^*, y](x) \\ p_g[z^*, y](x) \\ p_h[z^*, y](x) \end{pmatrix} = 0$$

has a solution. We can solve the y-components in a very similar way as the z-components. This time, we need to consider three subproblems of finding all the values of y such that the following three equations have a solution:

$$\begin{pmatrix} p_f[z^*, y](x) \\ p_g[z^*, y](x) \end{pmatrix} = 0; \quad \begin{pmatrix} p_f[z^*, y](x) \\ p_h[z^*, y](x) \end{pmatrix} = 0; \quad \begin{pmatrix} p_g[z^*, y](x) \\ p_h[z^*, y](x) \end{pmatrix} = 0.$$

All of these problems can be solved with the Cayley resultant matrix in a familiar way, only this time with $d = 2$ which simplifies the construction of the resultant matrix. Let then the set $S_{z,y}$ consist of the tuples (z^*, y^*) that solve all the above equations. In other words, for all the tuples $(z^*, y^*) \in S_{z,y}$, the system

$$\begin{pmatrix} p_f[z^*, y^*](x) \\ p_g[z^*, y^*](x) \\ p_h[z^*, y^*](x) \end{pmatrix} = 0 \tag{3.2}$$

has a solution. Now, we have three univariate polynomials left whose roots we need to find. We can treat each polynomial as a 1×1 matrix polynomial, and use the linearization in (2.5) to find their roots. All the values of x s.t. all the three polynomials in (3.2) are simultaneously zero then give us the x-coordinates of the solutions. The final solution set $S_{z,y,x}$ then contains all the triplets (z^*, y^*, x^*) s.t.

$$\begin{pmatrix} p_f(x^*, y^*, z^*) \\ p_g(x^*, y^*, z^*) \\ p_h(x^*, y^*, z^*) \end{pmatrix} = 0.$$

In other words, the elements of $S_{z,y,x}$ give the solutions to the polynomial approximation of the initial problem given in (2.1).

The resultant method may also yield infinite roots (see Definition 2.3.1). Infinite roots are not of interest to us and all computed roots corresponding to an infinite root can be ignored.

3.4 Numerical considerations

The aim of this section is to characterize the error in the output of the Cayley resultant method based root-finder. This error is defined as the difference between the computed output and the exact solution to the problem. We call this difference the *(absolute) forward error*.

When the computed solution differs from the exact solution to the problem, we can view this inaccurate solution as an exact solution to a different problem. The size of this difference in the problems is referred to as the *(absolute) backward error*. An algorithm is called *backward stable* if the norm of the backward error is bounded above by the product of machine precision and a moderate constant. By "moderate constant" we usually mean some low-degree polynomial in the input size with coefficient of order 1 in the monomial basis [14]. The numerical methods used in the root-finding algorithm are backward stable numerical methods, such as the QZ-algorithm for solving the generalized eigenvalue problem [26].

The backward error together with the condition number yields a bound on the forward error given by

$$\text{forward error} \lesssim \text{condition number} \times \text{backward error}, \tag{3.3}$$

which comes from a first order expansion for the worst-case forward error [14]. When the backward error is small, this first order expansion delivers a useful rule of thumb for how the forward error behaves. As we employ backward stable numerical methods, the backward error is small, although at least of the size of machine epsilon ϵ (2^{-52} in MATLAB). However, in order to give a bound for the forward error, we also need to consider the behaviour of the condition number.

3.4.1 Condition number

Let S and D be two Banach spaces. Given a problem whose solution $s \in S$ depends on some data $d \in D$, we can characterize how changes in d affect the value for s. Denote the change in d by δd and the resulting change in s by δs. The absolute condition number tells us the upper bound, to first order, for δs given δd. The absolute condition number is [21]

$$\lim_{\epsilon \to 0} \sup_{\|\delta d\| \leq \epsilon} \frac{\|\delta s\|}{\|\delta d\|},$$

where $\|\cdot\|$ is some chosen norm defined in S or D, depending on the argument.

A related concept is the relative condition number, which is given by [21]

$$\lim_{\epsilon \to 0} \sup_{\|\delta d\| \leq \epsilon} \frac{\|\delta s\|}{\|\delta d\|} \frac{\|s\|^{-1}}{\|d\|^{-1}}.$$

The relative condition number would give a more relevant measure of the sensitivity of the problem to perturbations as it takes into account the relative sizes of the error and the perturbation. However, it is often easier to derive results for the absolute condition number, and indeed there are more precisely formulated results for how the absolute condition number associated with the Cayley resultant method can behave (see [21]). As a result, we will primarily consider the absolute conditioning of the problem. Theorem 3.4.1 lets us characterize the absolute condition number of a *simple root* as a solution to the root-finding problem with the Jacobian matrix. A root is called simple if the Jacobian is invertible at the root [19].

Theorem 3.4.1 (Absolute condition number of a simple root (see [19]))
Let $(x^*, y^*, z^*) \in \mathbb{C}^3$ be a simple root of the polynomial root-finding problem given in (2.1). The absolute condition number of (x^*, y^*, z^*) associated with root-finding is $\|J(x^*, y^*, z^*)^{-1}\|_2$, where the Jacobian is given by

$$J(x,y,z) = \begin{bmatrix} \frac{\partial p_f}{\partial x}(x,y,z) & \frac{\partial p_f}{\partial y}(x,y,z) & \frac{\partial p_f}{\partial z}(x,y,z) \\ \frac{\partial p_g}{\partial x}(x,y,z) & \frac{\partial p_g}{\partial y}(x,y,z) & \frac{\partial p_g}{\partial z}(x,y,z) \\ \frac{\partial p_h}{\partial x}(x,y,z) & \frac{\partial p_h}{\partial y}(x,y,z) & \frac{\partial p_h}{\partial z}(x,y,z) \end{bmatrix}.$$

Definition 3.4.1 (Absolute condition number of a finite eigenvalue of a regular matrix polynomial [19, 27])
Let z^* be a finite eigenvalue of a regular matrix polynomial $R(z)$. The condition number of z^* as an eigenvalue of $R(z)$ is defined by

$$\kappa(z^*, R) = \lim_{\epsilon \to 0^+} \sup \left\{ \frac{1}{\epsilon} \min |\hat{z} - z^*| : \det(\widehat{R}(\hat{z})) = 0 \right\},$$

where the supremum is taken over the set of matrix polynomials $\widehat{R}(z)$ such that

$$\max_{z \in [-1,1]} \|\widehat{R}(z) - R(z)\|_2 \leq \epsilon.$$

Theorem 3.4.2 (Worsening of condition (see [21]))
There exist p_f, p_g and p_h in (2.1) with a simple root $(x^*, y^*, z^*) \in \mathbb{C}^3$ such that

$$\kappa(z^*, R_{Cayley}[z]) \geq \|J(x^*, y^*, z^*)^{-1}\|_2^3$$

and $\|J(x^*, y^*, z^*)^{-1}\|_2 > 1$.

By contrasting Theorem 3.4.2 with Theorem 3.4.1, we can conclude that the condition number for finding an eigenvalue of $R_{Cayley}[z]$ can be more than the cube of the condition number for finding a root of the associated root-finding problem. Hence, the Cayley resultant method transforms the root-finding problem into an eigenproblem of possibly poorer condition number.

Theorem 3.4.2 does not guarantee that this drop in conditioning takes place, but it shows that it might for some systems. For an example of such a system, let Q be a 3×3 orthonormal matrix such that $QQ^T = I$, and for $1 \leq i, j \leq 3$ let q_{ij} denote the element in position (i, j) of Q. Let $u < 1$, and consider the polynomial system

$$\begin{pmatrix} p_f(x,y,z) \\ p_g(x,y,z) \\ p_h(x,y,z) \end{pmatrix} := \begin{pmatrix} x^2 + u(q_{11}x + q_{12}y + q_{13}z) \\ y^2 + u(q_{21}x + q_{22}y + q_{23}z) \\ z^2 + u(q_{31}x + q_{32}y + q_{33}z) \end{pmatrix} = 0. \tag{3.4}$$

It is straightforward to see that $0 \in \mathbb{C}^3$ is a root of this polynomial system. The Jacobian at 0 is $J(0) = uQ$. Therefore, the absolute condition number associated with the root 0 is $\|J(0)^{-1}\| = u^{-1}$. It can be shown that the abosulte condition number for computing the eigenvalue $z^* = 0$ of the associated Cayley resultant matrix polynomial $R_{Cayley}[z]$ is $\kappa(z^*, R_{Cayley}[z]) = u^{-3}$ [21]. Hence, this type of system exhibits the same worsening of condition as shown possible by Theorem 3.4.2. We will try to replicate this behaviour in practice in Section 5.2 by testing the root-finder with systems where the transformed problem suffers from worse conditioning.

3.4.2 Missing a root

The absolute condition number tells us to first order how much small perturbations in the data can alter the solutions in the worst case. We can use the absolute condition number together with the absolute backward error to give a bound for the absolute forward error as expressed in (3.3). The absolute backward error is at least of the order of the machine precision ϵ (2^{-52} in MATLAB). Hence, if the condition number is larger than ϵ^{-1}, the error in the computed root can be of order 1 or higher. In this case, the root can be mapped outside the interval $[-1, 1]$ which would result in missing the root entirely.

We saw in Section 3.4.1 that the condition number of the root-finding problem can get raised to the third power when it is transformed into an eigenvalue problem of the Cayley resultant matrix polynomial. Hence, it can happen that the condition number of some root z^* associated with the root-finding

problem is naturally lower than ϵ^{-1}, and becomes higher than this after raising to the third power. In such a case, the root z^* might be missed entirely, while the original condition number guarantees that this would not happen in a numerically stable root-finding algorithm. If the condition number of the original problem were higher than ϵ^{-1} to begin with, one could not blame the method for missing the root. However, it is inappropriate to miss a root when the condition number of the original problem does not permit this.

Even if the solution is not missed entirely, its computed value might be significantly inaccurate due to poor conditioning. To address this problem, we will employ the Newton's method to polish the computed roots, although more sophisticated methods can also be used (see [19]).

3.4.3 Spurious solutions

A well-known problem with resultant based methods is the occurrence of spurious solutions. The resultant can be numerically singular even when the polynomials p_f, p_h and p_g do not share a root in the desired domain (see [19, Section 7.6]). An easy solution to this problem is to substitute the candidate solutions back into the system and leave out those that do not yield the value zero for each polynomial.

Chapter 4

Implementation

In the process of implementing the root-finder in practice, we were able to adopt the method presented in Chapter 3 to a great degree. However, often new ideas emerge and unexpected problems arise during the practical implementation of a plan, and the plan has to be modified accordingly. This chapter discusses these new ideas and how unexpected problems were addressed in the final implementation.

4.1 Choice of interpolation points for the Cayley function

When the Chebyshev nodes $H_{(\tau_1+1)}$ and $H_{(\tau_2+1)}$ overlap, the denominator of the Cayley function in (3.1) becomes zero at some of the interpolation points. The idea presented in Chapter 3 was to use L'Hospital's rule to evaluate the Cayley function at these problematic points. However, using the L'Hospital's rule in the polynomial interpolation of f_{Cayley} causes a practical problem: computing the derivatives consumes most of the computational time, and makes the root-finder impractically slow.

In this thesis, we primarily focus on the accuracy of the root-finder; hence, the discussion of the running time in this section will be done in a rather loose manner. In order to get an idea of how different implementations compare to each other, all values of running time presented in this section refer to how long it takes for the algorithm to run on the same computer that we can access at the time of writing.

When L'Hospital's rule is involved, computing a single component of the

solutions for a system of degree 5 already takes a high amount of time (hundreds of seconds depending on the complexity of the input and how many interpolation points overlap). This implies that solving systems of high degree is impractical if not infeasible with this type of implementation, even if there is some room for optimizing the running time.

The slowness is caused by the evaluation of the derivatives of f_{Cayley} at the points where the Chebyshev nodes H_{τ_1+1} and H_{τ_2+1} overlap. A way to circumvent this problem is to choose the interpolation points in such a way that these two sets of Chebyshev points are disjoint. In other words, we will use the set of five-dimensional Chebyshev points $H_{\kappa_1\times\kappa_2\times\kappa_2\times\kappa_1\times(\tau_3+1)}$ as interpolation points, where $\kappa_1 \geq \tau_1 + 1$ and $\kappa_2 \geq \tau_2 + 1$, and H_{κ_1} and H_{κ_2} are disjoint. As we are not reducing the amount of interpolation points, the resulting polynomial interpolant will still interpolate f_{Cayley} exactly. The resulting tensor $A[z]$ in (2.6) will be of size $\kappa_1 \times \kappa_2 \times \kappa_2 \times \kappa_1$. The tensor $\widetilde{A}[z]$ given by

$$\widetilde{A}[z](i_1, i_2, i_3, i_4) = A[z](i_1, i_2, i_3, i_4), \quad 1 \leq i_1, i_4 \leq \tau_1 + 1, \quad 1 \leq i_2, i_3 \leq \tau_2 + 1$$

is a tensor of size $(\tau_1 + 1) \times (\tau_2 + 1) \times (\tau_2 + 1) \times (\tau_1 + 1)$ which leaves out the additional zero entries of $A[z]$ caused by the addition of interpolation points. We can then define the Cayley resultant matrix polynomial as the unfolding of the tensor $\widetilde{A}[z]$ in the standard way given in Definition 2.3.3.

In order to make H_{κ_1} and H_{κ_2} disjoint, we choose $\kappa_1 = n$ and $\kappa_2 = 2n$, where n denotes the maximum of the user-specified maximal degrees n_f, n_g and n_h. Now, the Chebyshev nodes H_{κ_1} and H_{κ_2} are disjoint (proof in Appendix A), and there is no need to use the L'Hospital's rule in evaluating f_{Cayley}. When there is no need to evaluate the derivatives of the Cayley function, the root-finder is able to compute one of the components of the solutions for a system of degree 5 in a few seconds, as opposed to the hundreds of seconds that it took the previous implementation to perform the same computation.

In the new implementation of the root-finder, solving the eigenvalue problem becomes the bottleneck for higher degrees (approximately degree 5 and higher). Solving the eigenvalue problem is the bottleneck in the 2D problem solved in [19] as well. In terms of running time, this is the best we can do. Solving the eigenvalue problem is done by employing the QZ-algorithm implemented in the `eig` function in MATLAB whose running time we cannot affect. Hence, in an ideal implementation the rest of the algorithm runs faster than the `eig` function, in which case the eigenvalue problem becomes the bottleneck.

4.2 Omitting the polynomial interpolation of the system

The resultant methods can only be used to solve polynomial root-finding problems, and hence the plan proposed in Chapter 3 included finding polynomial interpolants p_f, p_g and p_h for functions f, g and h. The plan was then to use these polynomial interpolants in evaluating values of the Cayley function in order to perform a polynomial interpolation for the Cayley function itself. The key observation is that even if the Cayley function is not initially in polynomial form, its interpolant will be, which allows for the construction of the Cayley resultant matrix polynomial even if the initial form of the Cayley function is not polynomial. This allows for the use of the functions f, g and h directly in the Cayley function in place of the polynomial interpolants p_f, p_g and p_h. By using the functions f, g and h directly, we will be finding a polynomial interpolant for some function $\hat{f}_{Cayley}$ resembling the original Cayley function, which is given by

$$\hat{f}_{Cayley} = \frac{1}{(s_1 - t_1)(s_2 - t_2)} \det \begin{pmatrix} f(s_1, s_2, z) & g(s_1, s_2, z) & h(s_1, s_2, z) \\ f(t_1, s_2, z) & g(t_1, s_2, z) & h(t_1, s_2, z) \\ f(t_1, t_2, z) & g(t_1, t_2, z) & h(t_1, t_2, z) \end{pmatrix}.$$

We make sure that the polynomial interpolant of $\hat{f}_{Cayley}$ will have high enough degrees in each variable by perturbing the functions f, g and h in the same way we perturbed the polynomials p_f, p_g and p_h (see Section 3.2).

The polynomial interpolant of this type of modified Cayley function $\hat{f}_{Cayley}$ is not necessarily equal to the Cayley function constructed by using the polynomial approximations of the system functions. Both of these approaches approximate the original problem, and the remaining question is, is it preferable to have the truncation error at (1) the polynomial approximation of the system, and then proceed to interpolate the Cayley function exactly, or (2) polynomial interpolation of the Cayley function?

The motivation for skipping the polynomial interpolation of the system functions is that the fewer manipulations we perform to the input data, the better. However, we do not present mathematically formulated results that would support this decision. Hence, in future work beyond this thesis, we will need to investigate further the mathematical consequences of the decision of omitting the polynomial interpolation of the system.

For simplicity, we make the polynomial interpolant of the modified Cayley function $\hat{f}_{Cayley}$ to have the same degree structure as the Cayley func-

tion f_{Cayley} constructed through polynomial interpolants of the system functions. In other words, we choose the interpolation points for $\hat{f}_{Cayley}$ to be the same points as for f_{Cayley}: the five-dimensional Chebyshev nodes $H_{\kappa_1 \times \kappa_2 \times \kappa_2 \times \kappa_1 \times (\tau_3+1)}$. The resulting polynomial interpolant $p_{\hat{f}_{Cayley}}$ is not necessarily equal to the original function $\hat{f}_{Cayley}$, but is a polynomial approximation. We can express the interpolant $p_{\hat{f}_{Cayley}}$ as

$$p_{\hat{f}_{Cayley}} = \sum_{i_1=0}^{\kappa_1-1} \sum_{i_2=0}^{\kappa_2-1} \sum_{j_1=0}^{\kappa_2-1} \sum_{j_2=0}^{\kappa_1-1} \hat{A}_{i_1,i_2,j_1,j_2}[z] T_{i_1}(s_1) T_{i_2}(s_2) T_{j_1}(t_1) T_{j_2}(t_2),$$

and we again define the tensor $\tilde{A}[z]$ of size $(\tau_1 + 1) \times (\tau_2 + 1) \times (\tau_2 + 1) \times (\tau_1 + 1)$ through the tensor $\hat{A}[z]$ as explained in Section 4.1. We then compute the Cayley resultant matrix polynomial as the $((\tau_1 + 1)(\tau_2 + 1)) \times ((\tau_1 + 1)(\tau_2 + 1))$ matrix with entries in $\mathbb{C}[z]$ formed by the unfolding of the tensor $\tilde{A}[z]$. Then, we proceed to solve the problem similarly to what was outlined in Section 3.3.

When the functions f, g and h are low-degree polynomials, skipping the polynomial interpolation of the system does not change how the root-finder works. In order to characterize the numerical consequences of this change in practice, we consider non-polynomial systems in Section 5.3.

4.3 Solving fewer subproblems

The Cayley resultant method is used to compute the shared roots one component at a time as explained in Section 3.3. However, in order to avoid unnecessary computations, the second and third component of the solutions are computed with only one bivariate subproblem and one univariate subproblem (as opposed to three bivariate subproblems and three univariate subproblems as described in Section 3.3). The resulting solutions computed in this way might not be roots of all of the functions, but only some of them. The roots that other subproblems do not share are then eliminated by substituting every candidate solution into the original system of equations and only keeping those that actually solve the system. If the resulting value has a modulus larger than the threshold 10^{-12} for any function, the root is discarded. This also discards any other spurious solutions that the Cayley resultant method might yield.

Chapter 5

Numerical results

In this chapter, numerical results are presented. We characterize the accuracy of the root-finder by considering various example systems as well as running the algorithm in different arithmetic precisions. Some of the example systems we consider exhibit worsened conditioning of the transformed problem (see Section 3.4.1). We demonstrate the need for *domain subdivision*, and show how inaccurate roots can be polished by using Newton's method.

5.1 Example with three spheres

Consider a system of equations that describes three spheres. The first sphere is centered around the point $(\frac{1}{2}, \frac{1}{2}, 0)$ and has radius $\frac{1}{\sqrt{2}}$. The second sphere is centered around the point $(-\frac{1}{2}, \frac{1}{2}, 0)$ and has radius $\frac{1}{\sqrt{2}}$, and the third sphere is centered around the point $(0, 0, 0)$ and has radius $\frac{1}{2}$. This system of equations is given by

$$\begin{pmatrix} f(x,y,z) \\ g(x,y,z) \\ h(x,y,z) \end{pmatrix} := \begin{pmatrix} (x - \frac{1}{\sqrt{2}})^2 + (y - \frac{1}{\sqrt{2}})^2 + z^2 - \frac{1}{2} \\ (x + \frac{1}{\sqrt{2}})^2 + (y - \frac{1}{\sqrt{2}})^2 + z^2 - \frac{1}{2} \\ x^2 + y^2 + z^2 - \frac{1}{4} \end{pmatrix} = 0,$$

and represents a root-finding problem, albeit a rather simple one as the functions are low-degree polynomials. It is straightforward to check that the only intersection points of these spheres are $(0, \frac{1}{2}, \frac{\sqrt{3}}{4})$ and $(0, \frac{1}{2}, -\frac{\sqrt{3}}{4})$. The root-finder is able to find both roots, and the error in the computed roots is of order 10^{-15}. The error is computed as the distance, given by the 2-norm, between the numerically computed root and the analytically computed root. The accuracy in the computed roots seems promising as the error is not much larger than the machine epsilon ϵ (roughly 10^{-16} in MATLAB).

5.2 Examples of weakened conditioning

In this section, we analyse three systems of equations where the transformed problem has an increased condition number as explained in Section 3.4.1. Each system is constructed based on (3.4) and hence has $(0, 0, 0)$ as a shared root, while the eigenvalue problem associated with finding the z-component of the root has a condition number $\kappa = u^{-3}$.

The condition number for the original root-finding problem is $\kappa = u^{-1}$, and hence for some values of u it happens that the condition number of the transformed problem is higher than ϵ^{-1}, the inverse of the machine epsilon, while the condition number of the original problem is not. This can lead to the method missing a root when it should not (see Section 3.4.2), which is undesirable behaviour. The aim of this section is to see whether this happens in practice.

Let the following set of functions be called System 1:

$$\begin{cases} f_1(x,y,z) &= x^2 + u\frac{1}{\sqrt{3}}(x+y+z) \\ g_1(x,y,z) &= y^2 + u\left(x\sqrt{\frac{2}{3}} - y\frac{1}{\sqrt{6}} - z\frac{1}{\sqrt{6}}\right) \\ h_1(x,y,z) &= z^2 + u\frac{1}{\sqrt{2}}(y-z). \end{cases}$$

Let the following set of functions be called System 2:

$$\begin{cases} f_2(x,y,z) &= x^2 + ux \\ g_2(x,y,z) &= y^2 + u\left(y\frac{1}{\sqrt{2}} + z\frac{1}{\sqrt{2}}\right) \\ h_2(x,y,z) &= z^2 + u\left(y\frac{1}{\sqrt{2}} - z\frac{1}{\sqrt{2}}\right), \end{cases}$$

and finally, let the following set of functions be called System 3:

$$\begin{cases} f_3(x,y,z) &= x^2 + u\left(\frac{x\sqrt{3}}{2} + y\frac{\sqrt{3}}{4} + z\frac{1}{4}\right) \\ g_3(x,y,z) &= y^2 + u\left(-x\frac{1}{2} + y\frac{3}{4} + z\frac{\sqrt{3}}{4}\right) \\ h_3(x,y,z) &= z^2 + u\left(-y\frac{1}{2} + z\frac{\sqrt{3}}{2}\right). \end{cases}$$

In numerical computations, $(0, 0, 0)$ might be an unusually easy root to find. Hence, we shift the variables x, y and z by x_0, y_0 and z_0, respectively, drawn from a uniform distribution in the interval $[-1, 1]$. We evaluate the accuracy of the implementation by computing the z-component of this shifted root (x_0, y_0, z_0). We do no compute all the components of the root for practical reasons: the running time is smaller and the code is easier to manage.

The error in the z-component of the root was computed for all $u \in \{10^{-i} : i = 0, 1, \ldots, 16\}$. This was repeated 10^4 times with new x_0, y_0 and z_0 generated in every iteration. The averages of the error for each value of u were computed. The results are shown in Figure 5.1.

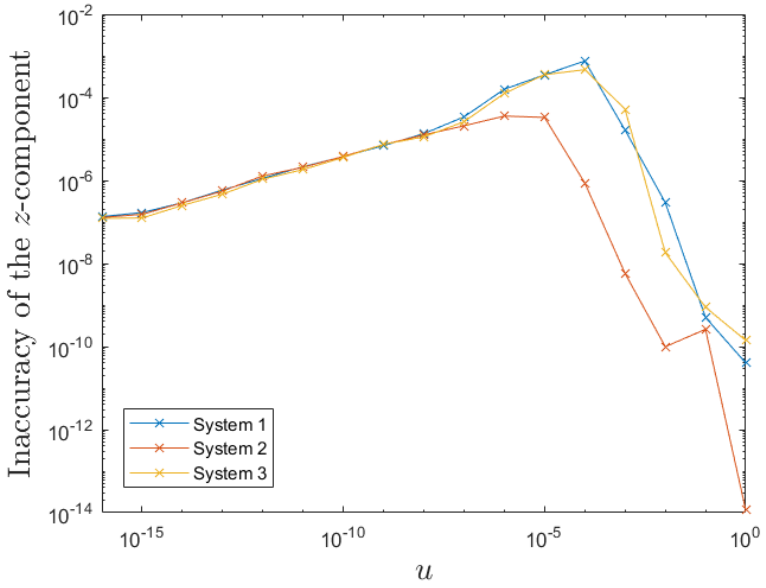

Figure 5.1: Inaccuracy of the z-component of the root (x_0, y_0, z_0) as a function of the parameter u for the shifted Systems 1, 2, and 3.

The behaviour of error in Systems 1 and 3 is almost identical. System 2 follows a similar trend with slight variation in the height and location of the inaccuracy peak. In all of the three systems, the inaccuracy in the computed root increases as the value for u decreases, up until u reaches approximately the value 10^{-4} for Systems 1 and 3, and the value 10^{-5} for System 2. At these inaccuracy peaks, the error is approximately 10^{-3} for Systems 1 and 3, and $10^{-4.5}$ for System 2. When the value for u decreases even further, the error starts decreasing and is of order 10^{-7} when $u = 10^{-16}$ for all systems.

The fact that the inaccuracy starts to stagnate and stops growing as u decreases can be curious considering that the condition number $\kappa = u^{-3}$ keeps increasing. One possible explanation for this stagnation behaviour can be attributed to the quadratic terms in the systems. The condition number is based on a first-order approximation of the system in the vicinity of the solution, and the derivative of the quadratic terms become zero at the solution (x_0, y_0, z_0). Hence, the quadratic terms do not influence the condition num-

ber, although in practice, they do contribute to the system significantly when the perturbation has a similar order of magnitude as ϵ and when u becomes small enough. However, this explanation is merely a tentative guess, and more effort will be put into finding the correct explanation of this behaviour in the future.

With small values of u, the condition number becomes larger than the inverse of the backward error, ϵ^{-1}, which causes the bound for the forward error to be of order 1 or higher. It was expected that this could cause the root to be entirely missed. However, we are not able to observe this behaviour in Figure 5.1 as the inaccuracy in the root is of order 10^{-3} at worst. In this case, it seems that merely a high condition number does not guarantee that the solution is inaccurate. Therefore, it still remains an open question whether the algorithm misses roots for some systems.

However, the condition number of the original root-finding problem is u^{-1}, and so the root-finding problem should not exhibit as high inaccuracy as 10^{-3} when $u = 10^{-4}$. This claim follows from the upper bound for the forward error which can be computed as backward error times condition number as expressed in (3.3). The backward error is approximately the machine epsilon $\epsilon \approx 10^{-16}$, and so for $u = 10^{-4}$, the upper bound becomes $\epsilon u^{-1} \approx 10^{-12}$, which is much smaller than the observed error of order 10^{-3}. Hence, this example shows that the conditioning is indeed worse in the transformed problem (the absolute conditioning in the transformed problem is u^{-3}, and for that reason at $u = 10^{-4}$ we have an upper bound $\epsilon u^{-3} \approx 10^{-4}$ which is close to the error we observe).

5.3 Non-polynomial example, domain subdivision and Newton's method

In this section, the aim is to compare the numerical accuracy of a root-finder that omits the polynomial interpolation of the system, and a root-finder that first finds polynomial interpolants for f, g and h and then computes f_{Cayley} by using these interpolants (see Section 4.2). We also demonstrate the need for domain subdivision, and show how inaccurate roots can be polished via Newton's method.

5.3.1 Omitting the system interpolation

Let us consider the following set of functions:

$$\begin{cases} f(x,y,z) &= \cos(2\pi x)\cos(2\pi y)\cos(2\pi z) \\ g(x,y,z) &= y \\ h(x,y,z) &= x^2+y^2+z^2-1. \end{cases}$$

The polynomial interpolation of the functions is done by utilising the `chebfun` package in MATLAB, which can be used to represent every function as a trivariate polynomial expressed in the Chebyshev basis to roughly 15 digits of relative accuracy in the domain $[-1,1]^3$. For the function f, this requires a polynomial interpolant that has a degree 28 in each x, y and z.

The original idea was to interpolate the Cayley function exactly based on the degrees of the polynomial approximations of the system functions. The degree structure of the polynomial interpolant of the Cayley function is then determined by the maximum of the degrees of the polynomial approximations, denoted by n (see Section 4.1). For large degrees, solving the eigenvalue problem takes an impractically high amount of time. For this problem, $n = 6$ is the highest value for the maximal degree of the polynomial interpolants that allows the algorithm to solve the eigenvalue problem in a convenient amount of time. Hence, we use `chebfun` to approximate the system functions with polynomials that have a maximal degree $n = 6$ in each variable, instead of a maximal degree 28 that would be required in order to represent the functions to 15 digits of relative accuracy (this causes a large truncation error; Section 5.3.2 addresses this point). In the no-system-interpolation design, we use the same value $n = 6$ to determine the degree structure of the polynomial interpolant of the Cayley function.

It can be calculated by hand that the set of common roots to the set of functions is

$$\left\{ \left(\pm\frac{\sqrt{15}}{4}, 0, \pm\frac{1}{4}\right), \left(\pm\frac{\sqrt{7}}{4}, 0, \pm\frac{3}{4}\right), \left(\pm\frac{1}{4}, 0, \pm\frac{\sqrt{15}}{4}\right), \left(\pm\frac{3}{4}, 0, \pm\frac{\sqrt{7}}{4}\right), \right.$$
$$\left. \left(\pm\frac{\sqrt{15}}{4}, 0, \mp\frac{1}{4}\right), \left(\pm\frac{\sqrt{7}}{4}, 0, \mp\frac{3}{4}\right), \left(\pm\frac{1}{4}, 0, \mp\frac{\sqrt{15}}{4}\right), \left(\pm\frac{3}{4}, 0, \mp\frac{\sqrt{7}}{4}\right) \right\}.$$

We define the error in the output as the 2-norm of the difference between the numerically computed root and the exact root. In the no-system-interpolation

design, the errors in the numerically computed roots are in the range between 10^{-2} and 10^{-1}. These errors remain similarly distributed in the same range when we perform the same computation by first finding polynomial interpolants for f, g and h and then computing f_{Cayley} by using these interpolants. This implies that skipping the interpolation of the system does not significantly weaken or improve the accuracy of the root-finder. However, individual examples like these do not constitute reliable empirical evidence, especially in the absence of theoretical results supporting the claim. The numerical consequences of skipping the polynomial interpolation of the system is something that will be more closely analysed for the future implementations of the root-finder.

5.3.2 Domain subdivision

Using a low-degree approximant for a function that would require an interpolant of high degree will surely introduce a large truncation error. This effect is demonstrated by the fact that the difference between the function f and its degree 6 approximant is of order 1 in the uniform norm in the domain $[-1, 1]^3$. The high error in the computed roots is likely the result of this effect. This issue can be solved by subdividing the domain into small enough regions so that a local approximant of degree 6 achieves 15 digits of relative accuracy in each subdomain. These local approximants can then be used to solve roots locally. This approach is described in [19] for the bivariate problem. In order to accurately handle inputs requiring a high-degree interpolant, domain subdivision will be adopted for the trivariate root-finder in future implementations. Thus, in addition to investigating the effect of omitting the polynomial interpolation of the system, this example demonstrates that using a low-degree approximant globally can hinder the performance, and motivates the implementation of domain subdivision in the future.

5.3.3 Newton's method

For the problem under consideration, it was observed that the error is in the range between 10^{-2} and 10^{-1}. It is likely that such a high inaccuracy is caused by truncation error and that it can be reduced with domain subdivision. However, it is also possible to polish the computed solutions through Newton's method.

Newton's method is an iterative method that under some assumptions produces successively better approximations that converge to a root [8, 22]. Given some initial guess $(x_0, y_0, z_0) \in \mathbb{C}^3$, Newton's method finds the next

iterate $(x_{i+1}, y_{i+1}, z_{i+1})$ based on the current value (x_i, y_i, z_i) as

$$\begin{bmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} - \begin{bmatrix} \frac{\partial f}{\partial x}(x,y,z) & \frac{\partial f}{\partial y}(x,y,z) & \frac{\partial f}{\partial z}(x,y,z) \\ \frac{\partial g}{\partial x}(x,y,z) & \frac{\partial g}{\partial y}(x,y,z) & \frac{\partial g}{\partial z}(x,y,z) \\ \frac{\partial h}{\partial x}(x,y,z) & \frac{\partial h}{\partial y}(x,y,z) & \frac{\partial h}{\partial z}(x,y,z) \end{bmatrix}^{-1} \begin{bmatrix} f(x_i, y_i, z_i) \\ g(x_i, y_i, z_i) \\ h(x_i, y_i, z_i) \end{bmatrix}.$$

Characterization of the convergence rate as well as the conditions for convergence are given by the Newton-Kantorovich theorem. Loosely speaking, the Newton-Kantorovich theorem states that the convergence is quadratic if the initial guess is close enough to the root; the functions are sufficiently smooth; and the root is simple (i.e. the Jacobian is invertible at the root), while linear convergence is possible under milder assumptions [8, 22].

Using the inaccurate roots given by the algorithm as the initial guesses, Newton's method makes all the computed roots exact to machine precision in four iterations. This demonstrates the possibility of polishing inaccurate roots as long as they are not entirely missed, and highlights the risk of missing a root as the largest numerical concern.

5.4 Accuracy in various arithmetic precisions

In order to gain insight on the numerical stability of the algorithm, we will investigate how running the algorithm in various precisions affects the accuracy of the output. We use the shifted System 1 from Section 5.2 since we already were able to verify that the algorithm can give inaccurate roots for this system. Similar to Section 5.2, the root whose accuracy we measure is (x_0, y_0, z_0), where x_0, y_0 and z_0 are drawn from a uniform distribution in the interval $[-1, 1]$.

As the system is polynomial and hence can be interpolated exactly, the algorithm should not introduce any truncation errors. Thus, the only source of error should be rounding errors that are affected by the arithmetic precision. Hence, the error should decrease by one order of magnitude when adding one more digit.

We use variable-precision accuracy (`vpa` function in MATLAB) to alter the arithmetic precision. The computation of the Cayley function was done analytically for this example due to practical problems with using `vpa` in conjunction with the `fftn` function in MATLAB. The computation of the eigenvalues of the Cayley resultant matrix polynomial is the most numerically unstable part of the algorithm, and hence this experiment will still give insight on the numerical behaviour of the implementation.

When we compute the eigenvalues of the Cayley matrix polynomial, we multiply the eigenvalue problem by random orthonormal matrices U and V s.t. the eigenvalue problem $L_0 w = \lambda L_1 w$ becomes $UL_0Vw = \lambda UL_1Vw$, where $L_1\lambda - L_0$ is the Colleague linearization of the Cayley resultant matrix polynomial. We do this in order to prevent the eigenvectors of this eigenvalue problem from having an unnaturally simple structure that could make the eigenvalues particularly easy to compute and in this way decrease the inaccuracy in the output. One example of such a structure is having many zero-valued components.

We ran the algorithm in 16, 32 and 64 significant digits. The error in the z-component of the root was computed for all $u \in \{10^{-i} : \ i = 0, 1, \ldots, 32\}$ for each arithmetic precision. This was repeated 50 times with new shifts x_0, y_0 and z_0 and new random orthonormal matrices U and V generated in every iteration. The averages of the error for each value of u were computed for every arithmetic precision. The results are shown in Figure 5.2.

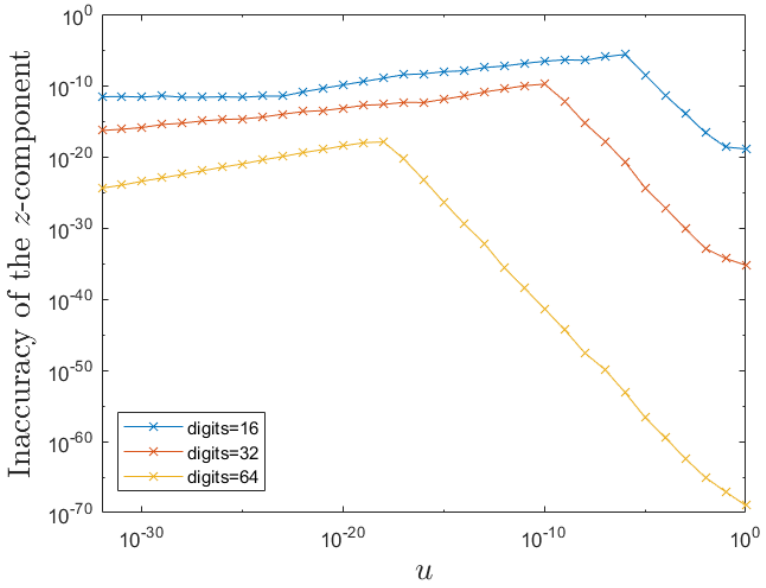

Figure 5.2: Inaccuracy of the z-component of the root as a function of the parameter u computed in three different arithmetic precisions: 16, 32 and 64 significant digits.

Increasing the amount of digits by n should improve the accuracy of the solutions by n orders of magnitude. We can see this behaviour in Figure 5.2 for $u \geq 10^{-6}$ when the error has not yet started to stagnate for any of the

arithmetic precisions. Until the stagnation starts to take place, the error increases in a cubic fashion as u decreases for each arithmetic precision. Theory predicts this behaviour as the condition number for the solution is $\kappa = u^{-3}$.

Changing the arithmetic precision affects the location and the value of the highest inaccuracy. The error starts to stagnate when it reaches the same order of magnitude that the value for u has. With 16 digits, this happens when both the error and u reach approximately the value 10^{-6}; with 32 digits, this happens when both the error and u reach approximately the value 10^{-10}; and with 64 digits, this happens when both the error and u reach approximately the value 10^{-18}. Increasing the amount of digits by 16 seems to decrease the highest inaccuracy, as well as the value for u at which this peak is reached, by four orders of magnitude.

All of the linear terms in the system are scaled by u, while the quadratic terms are not. It may be that for this reason, the quadratic terms begin to dominate when the error reaches the same order of magnitude that the value for u has. This would explain why the stagnation begins when the error reaches the value of u, and would also support the suggestion from Section 5.2 that the stagnation behaviour might be caused by the quadratic terms in the system.

As the value for u decreases after reaching the peak inaccuracy, the accuracy starts increasing significantly faster with 64 digits than for 32 digits. A similar difference in the rate at which the accuracy changes is not as noticeable between 16 and 32 digits.

5.5 Summary of numerical results

Section 5.1 gave an example of a low-degree root-finding problem that the implementation can solve close to machine precision. However, in Section 5.2 we saw that the error in the computed solutions for three different systems were significantly higher than what is permitted by the condition number of the solution to a standard root-finding problem. The behaviour of the error as a function of the parameter u seemed to be in line with the theoretical result that the condition number would get cubed as a result of using the Cayley resultant method, apart from the fact that the error started to stagnate after reaching a peak value for some value of u even though the condition number continued to increase.

The observations in Section 5.4 were consistent with the observations of Sec-

tion 5.2. It was observed that the error in the computed root of System 1 increases in a cubic manner as the parameter u decreases before reaching the peak inaccuracy for all three different arithmetic precisions. This implies that the condition number indeed is cubed, and supports the idea that the Cayley resultant method causes a worsening in the condition of the solutions. The stagnation behavior of the error occurred for all arithmetic precisions when the error reached a value close to the value of u, which is likely a feature of this particular example system, and not an intrinsic feature of the implementation.

Section 5.3 showed that skipping the system interpolation does not always have a significant impact on the accuracy of the computed solutions. This topic will be investigated further for future implementations of the root-finder. The section also demonstrated the need for domain subdivision in order to handle high-order inputs with decent accuracy. This will be included in the root-finding algorithm in the future.

It is concluded that the implementation does not exhibit ideal numerical stability as the Cayley resultant method can cause a deterioration in the conditioning. However, yielding inaccurate solutions is not a significant limitation of the algorithm as it is possible to improve the accuracy of the computed roots by applying the Newton's method as seen in Section 5.3. It is still an open question under what conditions, if any, the implementation can miss a root entirely.

Chapter 6

Conclusion

The aim of this thesis consisted of three main parts: (1) to explain how the Cayley resultant method can be used in solving trivariate root-finding problems; (2) to implement a functioning root-finder in MATLAB based on the proposed method; and (3) to characterize the numerical accuracy of the implementation through theoretical as well as practical results.

The aim (1) was reached to a satisfactory degree. Reaching (1) required us to cover background knowledge in polynomial interpolation techniques, matrix polynomials and polynomial eigenvalue problems, Chebyshev polynomials and the Chebyshev basis. It was also described how the Cayley resultant transforms a system of polynomial equations into a polynomial eigenvalue problem.

The aim (3) was reached to an adequate degree with some room for future work. We outlined how the Cayley resultant method transforms the root-finding problem into a problem of worse conditioning, and were able to demonstrate this in practice. We saw how using global low-degree approximants without employing domain subdivision can lead to a high truncation error. Further exploration is needed in trying to determine whether the algorithm can miss a root entirely under some conditions, and whether omitting the polynomial interpolation of the system is numerically beneficial.

Based on the numerical results, we conclude that the aim (2) was reached to some degree with plenty of room for future work. The implementation is able to solve some root-finding problems close to machine precision, while it can be highly inaccurate for other problems. The algorithm suffers from two major sources of error: the worsening of the conditioning of the problem caused by the Cayley resultant method, and a large truncation error for

high-degree input functions.

To increase the accuracy, the implementation introduces additional post-processing of inaccurately computed roots via Newton's method, which can greatly enhance the accuracy of the roots. The ability to polish any root that is found suggests that the largest concern for the performance of the implementation is the possibility of missing a root entirely. The problem with large truncation errors for high-degree inputs can be addressed via domain subdivision. This is an essential part of the algorithm that will be implemented in the future.

One aspect to consider for future implementations is how to handle inputs of any degree. In the current implementation, the construction of the Cayley resultant matrix requires perturbing the polynomial system with suitable monomials. For future implementations of the algorithm, we will consider alternative methods to handle any possible input.

Bibliography

[1] Barnett, S. Greatest common divisors from generalized Sylvester resultant matrices. *Linear and Multilinear Algebra, 8*, 4 (1980), 271–279.

[2] Bates, D. J., Haunstein, J. D., Sommese, A. J., and Wampler, C. W. *Numerically Solving Polynomial Systems with Bertini.* Society for Industrial and Applied Mathematics, 2013.

[3] Bézout, É. *Théorie Générale des Équations Algébrique.* PhD thesis, Pierres, Paris, 1779.

[4] Cayley, A. On the theory of elimination. *Cambridge and Dublin Mathematical Journal, 3* (1848), 116–120.

[5] Conte, S. D., and de Boor, C. *Elementary Numerical Analysis. An Algorithmic Approach*, 3 ed. McGraw-Hill, 1980.

[6] Cox, D., Little, J., and O'Shea, D. *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*, 3rd ed. Springer, 2008.

[7] Davis, P. J. *Interpolation and Approximation.* Dover Publications, 1975.

[8] Deuflhard, P. *Newton Methods for Nonlinear Problems: Affine Invariance and Adaptive Algorithms.* Springer Berlin Heidelberg, 2005.

[9] Deville, M., and Labrosse, G. An algorithm for the evaluation of multidimensional (direct and inverse) discrete Chebyshev transforms. *Journal of Computational and Applied Mathematics, 8*, 4 (1982), 293–304.

[10] Gelfand, I., Kapranov, M., and Zelevinsky, A. *Discriminants, Resultants, and Multidimensional Determinants*. Springer, Birkhäuser, 2008.

[11] Gil, A., Segura, J., and Temme, N. *Numerical Methods for Special Functions*. SIAM, 2007.

[12] Gohberg, I., Kaashoek, M. A., and Lancaster, P. General theory of regular matrix polynomials and band Toeplitz operators. *Integral Equations and Operator Theory, 11* (1988), 776–882.

[13] Gohberg, I., Lancaster, P., and Rodman, L. *Matrix Polynomials*. Academic Press, 1982.

[14] Higham, N. *Accuracy and Stability of Numerical Algorithms*, 2nd ed. Society for Industrial and Applied Mathematics, 2002.

[15] Hilton, A., Stoddart, A. J., Illingworth, J., and Windeatt, T. Marching triangles: range image fusion for complex object modelling. In *Proceedings of 3rd IEEE International Conference on Image Processing* (1996), vol. 2, pp. 381–384.

[16] Kapur, D., Saxena, T., and Yang, L. Algebraic and geometric reasoning using dixon resultants. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation* (New York, NY, USA, 1994), ISSAC '94, Association for Computing Machinery, p. 99–107.

[17] Mason, J., and Handscomb, D. C. *Chebyshev Polynomials*. Chapman and Hall/CRC, 2002.

[18] Nakatsukasa, Y., and Noferini, V. On the stability of computing polynomial roots via confederate linearizations. *Mathematics of Computation, 85*, 301 (2016), 2391–2425.

[19] Nakatsukasa, Y., Noferini, V., and Townsend, A. Computing the common zeros of two bivariate functions via Bézout resultants. *Numerische Mathematik, 129* (2015), 181–209.

[20] Noferini, V. *Polynomial Eigenproblems: a root-finding approach.* PhD thesis, Università di Pisa, 2012. Supervisors: D. A. Bini and L. Gemignani.

[21] Noferini, V., and Townsend, A. Numerical Instability of Resultant Methods for Multidimensional Rootfinding. *SIAM Journal on Numerical Analysis, 54*, 2 (2016), 719–743.

[22] Ortega, J. M. The Newton-Kantorovich Theorem. *The American Mathematical Monthly, 75*, 6 (1968), 658–660.

[23] Ragnarsson, S., and Loan, C. F. V. Block tensor unfoldings. *SIAM Journal on Matrix Analysis and Applications, 33*, 1 (2012), 149–169.

[24] Smith, S. J. Lebesgue constants in polynomial interpolation. *Annales Mathematicae et Informaticae, 33* (2006), 109–123.

[25] Sommese, A., and Wampler, C. *The Numerical Solution of Systems of Polynomials Arising in Engineering and Science*. World Scientific, 2005.

[26] Sun, J. *Stability and Accuracy: Perturbation Analysis of Algebraic Eigenproblems*. Technical Report UMINF 98.07. University of Umeå, 1998.

[27] Tisseur, F. Backward error and condition of polynomial eigenvalue problems. *Linear Algebra and its Applications, 309*, 1 (2000), 339–361.

[28] Trefethen, L. *Approximation Theory and Approximation Practice*. SIAM, 2013.

Appendix A

Disjointness of H_n and H_{2n}

Let q_1, q_2, and q_3 be trivariate polynomials that have the same degree in every variable. Let n be the maximum of these degrees. The Cayley function defined based on the polynomials q_1, q_2, and q_3 is

$$f_{Cayley} = \frac{1}{(s_1 - t_1)(s_2 - t_2)} \det \begin{pmatrix} q_1(s_1, s_2) & q_2(s_1, s_2) & q_3(s_1, s_2) \\ q_1(t_1, s_2) & q_2(t_1, s_2) & q_3(t_1, s_2) \\ q_1(t_1, t_2) & q_2(t_1, t_2) & q_3(t_1, t_2) \end{pmatrix}.$$

The degree of the Cayley function in s_1 and t_2 is less than or equal to $n-1$, and in s_2 and t_1 it is less than or equal to $2n-1$. Thus, using n points for s_1 and t_2 and $2n$ points for s_2 and t_1 interpolates the polynomial f_{Cayley} exactly.

One complication for the numerical interpolation of the Cayley function is the fact that when s_1 and t_1 or when s_2 and t_2 are equal, the determinant and the divisor $(s_1 - t_1)(s_2 - t_2)$ become zero at the same time. In such cases L'Hospital's rule would need to be employed. However, if we use Chebyshev points as the interpolation points, then we can use Proposition A.0.1 to conclude that the divisor can never be zero, and thus L'Hospital's rule is not needed. This follows from the fact that when the sets of interpolation points for s_i and t_i are disjoint, $s_i - t_i$ is always nonzero at all interpolation points.

Proposition A.0.1
Let $n \in \mathbb{Z}^+$. Let H_n be the set of n Chebyshev points, and let H_{2n} be the set of $2n$ Chebyshev points. The sets H_n and H_{2n} are disjoint.

Proof. The elements of H_n are the roots of the Chebyshev polynomial of the

first kind of degree n. These roots H_n^k are

$$H_n^k = \cos\left(\frac{2k-1}{2n}\pi\right),\ k = 1, 2, ..., n.$$

Likewise, the elements of H_{2n} are

$$H_{2n}^k = \cos\left(\frac{2k-1}{4n}\pi\right),\ k = 1, 2, ..., 2n.$$

We notice that the argument of cosine is always in the interval $[0, \pi]$. In this interval, cosine is strictly decreasing. Hence, any two sets of Chebyshev points are disjoint if and only if the corresponding sets of arguments of cosine are disjoint. Let us refer to the elements of these sets as x_n^k and x_{2n}^k, where

$$x_n^k = \frac{2k-1}{2n}\pi,\ k = 1, 2, ..., n,$$

and

$$x_{2n}^k = \frac{2k-1}{4n}\pi,\ k = 1, 2, ..., 2n.$$

We can further ignore the multiplication by π, as multiplying all elements of the two sets by a nonzero constant does not affect the disjointness of the sets. We call these new sets y_n and y_{2n} such that

$$y_n^k = \frac{2k-1}{2n},\ k = 1, 2, ..., n,$$

and

$$y_{2n}^k = \frac{2k-1}{4n},\ k = 1, 2, ..., 2n.$$

We want to show that $y_n^s \neq y_{2n}^t\ \forall s = 1, 2, ..., n;\ t = 1, 2, ..., 2n$. Let us prove this by contradiction.

Assume that $\exists s, t \in \mathbb{Z}^+$ such that $y_s^n = y_t^{2n}$. Then

$$\frac{2s-1}{2n} = \frac{2t-1}{4n}.$$

After multiplying both sides by $4n$, we have that

$$2(2s-1) = 2t-1.$$

We note that the left side is always an even number, and the right side is always odd. Hence, the equality cannot hold, and $y_n^s \neq y_{2n}^t\ \forall s, t \in \mathbb{Z}^+$. This proves that the sets y_n and y_{2n} are disjoint, which further proves that the sets of roots H_n^k and H_{2n}^k are disjoint. □