CHALMERS

GÖTEBORGS UNIVERSITET

Lösning av polynomekvationer

En undersökning av Sturmkedjemetoden, argumentprincipmetoden och kompanjonmatrismetoden

Examensarbete för kandidatexamen i matematik vid Göteborgs universitet
Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Kári Kristjansson
Markus Bengtsson
Tim Johansson Nero

Institutionen för matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2018

Lösning av polynomekvationer

En undersökning av Sturmkedjemetoden, argumentprincipmetoden och kompanjonmatrismetoden

Examensarbete för kandidatexamen i matematik vid Göteborgs universitet

Kári Kristjansson

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk fysik vid Chalmers

Tim Johansson Nero

Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk matematik vid Chalmers

Markus Bengtsson

Handledare: Mårten Wadenbäck
Examinator: Maria Roginskaya, Marina Axelson-Fisk

Institutionen för matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2018

Populärvetenskaplig presentation

Polynom är ett koncept som introduceras tidigt i varje matematikers utbildning. Samtidigt som den mystiska variabeln x upptäcks för första gången införs den allra enklaste sortens polynomekvationer, nämligen de av grad ett. Lite senare i utbildningen kommer studenten att stöta på polynomekvationer av grad två som snabbt löses med hjälp av den förhållandevis enkla pq-formeln. Stärkt av sin framgång rusar den unga matematikern snabbt mot det svårare konceptet av kvadratkompletering, vilken blir nådastöten för den stackars andragradsekvationen. Till synes ostoppbar fortsätter studenten sin erövring av polynomekvationer och inte heller de av grad tre är längre säkra. Fjärdegradspolynomen faller sedan och överkommen av hybris frågar sig studenten nu om det finns något polynom som kan sätta sig emot dennes nästan obegränsade kunskap.

Det visar sig att polynomekvationer av grad fem eller högre inte kan lösas med någon allmän formel och vår student får känna på sitt första nederlag. Slagen, men inte besegrad fortsätter studenten sin odyssé och går sakta från sina hemkära exakta numeriska värden till den ovissa sidan av matematiska approximationer. Någonstans på vägen får studenten sällskap av en ny vapendragare, ett så kallat matematisk beräkningsprogram vid namn Matlab. Med den nya medhjälparen vid sin sida och färsk kunskap om approximativa lösningar tänds en gnista hopp i studentens hjärta.

Snabbt inser studenten att det har öppnats en hel värld av möjligheter med oändliga sätt att bekämpa kriget mot polynomekvationerna. Från en uppsjö av olika metoder så är det tre tillvägagångsätt som fångar studentens intresse. Ovillig att helt överge bekvämligheterna som bara de reella talen kan erbjuda utses som första metod Sturmkedjemetoden. Beväpnad med kunskap om de komplexa talen väljer studenten att nyttja argumentpricipmetoden för att föra Sturmkedjemetodens fackla vidare till det komplexa planet. Sist men inte minst intresserar sig studenten för hur Matlab löser polynomekvaktioner och kommer över kompanjonmatrismetoden.

Besatt i att inte bara slå utan förgöra polynomekvationer försöker studenten att ta reda på vilken metod som fungerar bäst. Alla tre metoderna jämförs med avseende på för- och nackdelar. Med hjärtat i halsgropen väntar studenten på resultatet av jämförelsen. Tänk om kriget mot polynomekvationerna kunde vinnas med en approximativ metod. Det skulle vara mycket användbart för alla som skulle kunna tänka sig vilja lösa polynomekvationer av grad fem eller högre.

Efter teoretisk analys, algoritmskrivande, dataprogrammering och testande av de tre olika metoderna visade det sig att de alla har sina fördelar och sina nackdelar. Sturmkedjemetoden är användbar om endast reella lösningar efterfrågas, argumentprincipmetoden fungerar bra för de flesta polynom men behöver lång beräkningstid vid höga gradtal. Kompanjonmatrismetoden löser det flesta fallen, är snabb och därför bra om man ska beräkna många polynom. Vilken metod som passar bäst beror således helt och hållet på vilket problem som ska lösas.

Sammanfattning

Polynomekvationen är ett grundläggande matematiskt begrepp men det är inte möjligt att hitta en exakt representation av nollställena för gradtal större än fyra. Trots att det inte går att hitta exakta lösningar till dessa polynomekvationer kan man med olika metoder ofta uppnå en god approximation till nollställena. Det finns mer eller mindre enkla sätt att approximera rötterna till ett polynom av hög grad och i denna rapport utforskar vi tre av dessa metoder där respektive metod är baserad på: Sturmkedjor, argumentprincipen eller kompanjonmatrisen

Genom litteraturstudier bekantade vi oss med ämnesområdet som i grunden är teoretiskt. Befintliga numeriska metoder för approximativa lösningar av polynom analyserades och bevisades matematiskt. Vi testade sedan olika lösningsmetoder inklusive Sturmkedjemetoden, argumentprincipmetoden samt kompanjonmatrismetoden. Vi testade sedan beräkningsprogrammet *roots* i Matlab som använder kompanjonmatrismetoden för att hitta rötter till polynomekvationer.

De tre metoder som vi testade har alla sina fördelar och nackdelar. Vi kunde inte utse en bästa metod för att hitta nollställen till alla former av polynomekvationer. Ska man beräkna många polynom och det inte spelar någon roll att reella lösningar kan få en liten imaginärdel kan kompajonmatrismetoden vara intressant. Om endast reella lösningar efterfrågas och polynomen har heltalskoefficienter är Sturmkedjemetoden mycket användbar. Argumentprincipenmetoden fungerar bra för de flesta rötter men när polynomets gradtal stiger så behöver den betydligt längre beräkningstid än de andra metoderna. Vilken metod som passar bäst beror således på vilket problem som ska lösas.

Abstract

Although a polynomial equation is a basic mathematical concept it is not possible to find an exact representation of solutions to polynomials of a degree greater than four. This problem can be circumvented by the use of numerical approximations of the roots. Several approximation methods are available for polynomials of higher degrees and in this study we investigated three different methods including those based on Sturm sequences, the argument principle and companion matrices.

We got acquainted with the field, which is highly theoretical, through literature studies. Existing numerical methods for approximative solutions of polynomial equations were analysed and verified mathematically. We then tested different methods for solving polynomial equations including those based on Sturm sequences, the argument principle and companion matrices. We also tested the calculation program *roots* used by Matlab and based on the companion matrix method to find the roots of polynomal equations.

The three methods that were tested had all their own advantages and drawbacks. It was not possible to appoint the best method for solving all forms of polynomial zeros. The method based on companion matrices was the fasted and solved most forms of polynomial equations, but some of the real roots had a small imaginary part. The method based on Sturm sequences was highly applicable if only real root solutions were requested and the coefficients of the polynomial equations were only integrals. The method based on the argument principle worked well on most polynomial equations, but when the polynomial degree increased the calculation time for this method increased rapidly. Thus, which method is most applicable is highly dependent on the type of problem that is to be solved.

Innehåll

1 Inledning **1**
- 1.1 Syfte . . . 1
- 1.2 Precisering av frågeställningen . . . 1
- 1.3 Avgränsningar . . . 2

2 Polynomekvationer och relevanta satser **2**
- 2.1 Definition av ett polynom . . . 2
- 2.2 Polynomekvationers egenskaper . . . 2

3 Metoder för att lösa polynomekvationer **4**
- 3.1 Sturmkedjemetoden . . . 4
  - 3.1.1 Konstruktion av en Sturmkedja . . . 5
  - 3.1.2 Variation av en Sturmkedja . . . 5
  - 3.1.3 Antalet reella nollställen . . . 6
- 3.2 Argumentprincipmetoden . . . 7
  - 3.2.1 Argumentprincipen . . . 7
  - 3.2.2 Antalet komplexa nollställen . . . 9
- 3.3 Kompanjonmatrismetoden . . . 9
  - 3.3.1 QR-algoritmen . . . 11
  - 3.3.2 QR-faktorisering . . . 11
  - 3.3.3 Algoritm Householder QR-faktorisering . . . 13

4 Algoritmer för programimplementering **13**
- 4.1 Algoritmer för Sturmkedjemetoden . . . 14
- 4.2 Algoritmer för argumentprincipmetoden . . . 16
- 4.3 Algoritm för kompanjonmatrismetoden . . . 18
- 4.4 Tillämpning av algoritmer i matlab . . . 18

5 Resultat **19**
- 5.1 Sturmkedjemetoden . . . 19
- 5.2 Argumentprincipmetoden . . . 20
- 5.3 Kompanjonmatrismetoden . . . 20

6 Diskussion **23**
- 6.1 Sturmkedjemetoden . . . 23
- 6.2 Argumentprincipmetoden . . . 23
- 6.3 Kompanjonmatrismetoden . . . 24
- 6.4 Jämförelse av metoder och fördjupning i underfrågor . . . 24
- 6.5 Konklusion . . . 25

A Appendix **27**
- A.1 Matlab kod . . . 27
  - A.1.1 Kod för utförande av Sturmkedjormetoden . . . 27
  - A.1.2 Kod för kompanjonmatrismetoden . . . 30
  - A.1.3 Kod för argumentprincipen . . . 31
  - A.1.4 Kod för test av funktioner . . . 32

Förord

Denna rapport ingår som ett kandidatarbete i matematiska vetenskaper för Göteborgs universitet och Chalmers Tekniska Högskola. Studien har gjorts i samarbete mellan en student från Göteborgs universitet (KK) och två studenter från Chalmers Tekniska Högskola. Den ene studenten på Chalmers läser civilingenjörsprogrammet Teknisk matematik (MB) och den andre civilingenjörsprogrammet Teknisk fysik (TJN). Under arbetet har det förts en loggbok över gruppens aktivitet, den enskilde individens bidrag samt möten med handledare och representanter från fackspråk. Nedan följer en lista över vilka kapitel och avsnitt som respektive författare har bidragit med.

1. Populärvetenskaplig presentation – Markus Bengtsson
2. Sammanfattning – Tim Johansson Nero
3. Förord – Kári Kristjansson
4. Inledning – Kári Kristjansson
5. Definition av ett polynom – Kári Kristjansson
6. Polynomekvationers egenskaper – Tim Johansson Nero
7. Konstruktion av en Sturmkedja – Kári Kristjansson & Tim Johansson Nero
8. Variation av en Sturmkedja – Tim Johansson Nero
9. Antal reella nollställen – Markus Bengtsson
10. Exempel 3.2 – Kári Kristjansson
11. Argumentprincipmetoden – Markus Bengtsson
12. Kompanjonmatrismetoden – Kári Kristjansson
13. Algoritmer för programimplementering – Hela gruppen
14. Resultat – Hela gruppen
15. Diskussion – Hela gruppen

Arbetet har även innefattat dataprogrammering och nedan ges en lista över vem som skrivit vilken kod. Programmeringskoderna redovisas i Appendix A.

1. Sturm.m – Markus Bengtsson & Tim Johansson Nero
2. variation.m – Markus Bengtsson & Tim Johansson Nero
3. isol.m – Kári Kristjansson
4. approx.m – Kári Kristjansson
5. GCD.m – Tim Johansson Nero
6. MultRoot.m – Tim Johansson Nero
7. Argp.m – Markus Bengtsson
8. ImgIsol.m – Markus Bengtsson
9. ArgPSolve.m – Markus Bengtsson
10. KomMatMet.m – Kári Kristjansson & Tim Johansson Nero
11. TestSturm.m – Tim Johansson Nero

Vi vill tacka vår handledare Mårten Wadenbäck för hans skickliga vägledning och stöd under kandidatarbetet.

1 Inledning

Ett vanligt förekommande problem inom matematiken och dess tillämpningar är att bestämma lösningarna till polynomekvationer. En naturlig fråga är om en ekvation alltid är lösbar. Algebrans fundamentalsats garanterar att varje polynom av grad ett eller större har minst ett komplext nollställe. Ekvationer av grad två kan ju alltid på ett effektivt sätt lösas med pq-formeln eller med hjälp av kvadratkompletering. I boken *Algebra och geometri* [VE11] berättar Anders Vretbland och Kerstin Ekstig hur människor genom tiderna har intresserat sig av att lösa polynomekvationer. Redan hos babylonierna fanns det kunskap om detta och de klarade av att lösa andragradsekvationer som hade positiva tal som lösningar. Antikens greker tog detta vidare och funderade över om det fanns liknande formler som kunde lösa ekvationer av grad tre, fyra och så vidare. Det skulle dock dröja ända fram tills 1500-talet innan man lyckades hitta formler för lösningen av ekvationer av högre grad än två.

Det var italienarna Scipione Ferro (1465–1526) och Niccolò Fontana, kallad Tartaglia (1499–1557), som oberoende av varandra löste tredjegradsekvationen [VE11]. Tartaglia anförtrodde lösningen i hemlighet till Girolamo Cardano (1501–1576) som dock avslöjade det hela genom en publikation. Ännu en italienare, Ludvico Ferraro (1522–1565), visade sedan att en ekvation av fjärde graden kunde reduceras till ett problem för en tredjegradsekvation, varvid den blev lösbar.

För ekvationer av femte graden eller högre kunde man inte komma fram till en lösning. På 1820-talet visade norrmannen Niels Henrik Abel (1802–1832) och fransmannen Évariste Galois (1811–1832) var för sig att för en allmän ekvation av grad fem eller högre går det inte att hitta en formel för lösningarna [VE11]. Däremot så finns det metoder för att approximera fram lösningar, och det är i många sammanhang tillräckligt bra.

I detta arbete kommer vi att studera lösning av polynomekvationer av grader upp till och lika med 50. För polynom av grad fyra eller lägre finns givna formler och metoder för att hitta alla komplexa och reella rötter. Polynomekvationer av grad fem eller högre saknar lösningsformler, men med hjälp av numeriska metoder går det att finna approximativa rötter med godtycklig precision [VE11].

1.1 Syfte

Syftet med denna rapport är att analysera och matematiskt bevisa ett par befintliga metoder för att lösa polynomekvationer av grad fem eller högre, samt att testa vilken metod som mest effektivt approximerar lösningen till en given polynomekvation.

1.2 Precisering av frågeställningen

I vårt arbete har vi valt att fokusera på följande huvudfrågeställning:

1. Vi vill beskriva, undersöka och jämnföra de tre metoderna Sturmkedjemetoden, argumentprincipmetoden och kompanjonmatrismetoden för att bestämma nollställen till polynomekvationer.

Vi har även valt att studera följande underfrågor:

1. Hur kan man på ett effektivt sätt säkert hitta <u>alla</u> lösningar till en polynomekvation $f(z) = 0$?
2. Hur många av nollställena kommer att vara reella?
3. Hur många nollställen kommer finns i intervallet $[a, b]$?
4. Vad händer om något nollställe har multiplicitet större än ett?
5. Vilken cirkel $|z| \leq R$ i det komplexa planet kommer med all säkerhet innehålla alla nollställen till $f(z)$?

1.3 Avgränsningar

Vi har begränsat rapporten till att behandla polynomekvationer i en variabel. För Sturmkedjor har endast polynom med reella koefficienter använts, men för argumentprincipen och kompanjonmatrismetoden behandlades även polynomen med komplexa koefficienter. De polynom som vi har studerat är på formen:

$$p(x) = \sum_{i=0}^{n} a_i x^i, \text{ där } a_i \in \mathbb{R} \text{ eller } a_i \in \mathbb{C}. \tag{1}$$

2 Polynomekvationer och relevanta satser

För att utföra närmare studier av polynom och lösningar av polynomekvationer krävs först att deras utformning och grundläggande egenskaper redovisas. I detta kapitel kommer vi att definiera polynom och beskriva några av deras egenskaper. Vi kommer även att redogöra för de viktigaste satserna av betydelse för polynom.

2.1 Definition av ett polynom

Ett polynom är ett matematiskt uttryck, som består av ett antal givna, reella eller komplexa tal a_0, a_1,..., a_n och en variabel x, och har formen

$$p(x) = a_0 + a_1 + a_2x^2 + \cdots + a_nx^n. \tag{2}$$

Talen a_0, a_1,..., a_n kallas för polynomets koefficienter. Man säger att polynomet $p(x)$ är reellt om dess koefficienter är reella, respektive komplext om koefficienterna är komplexa. Om p och q är två polynom, säger man att p och q är lika om, och endast om, deras koefficienter är lika. Om alla koefficienter a_k i (2) är noll så så får polynomet beteckningen 0 och kallas för nollpolynomet. Om $p(x) = a_0$, det vill säga om alla a_k med k ≥ 1 är noll, är polynomet konstant och bestäms av talet a_0. Om p inte är nollpolynomet och n är det största talet för vilket $a_n \neq 0$, säger man att p har graden n och betecknas $n = \operatorname{grad} p$. Nollpolynomet saknar grad. För alla andra polynom p är $\operatorname{grad} p$ ett heltal ≥ 0 och ett konstant polynom p har $\operatorname{grad} p = 0$. Summan $p + q$ och produkten pq av två polynom ger nya polynom.

Om p är ett polynom, så kallas ekvationen

$$p(x) = 0, \tag{3}$$

för en polynomekvation. En lösning eller rot[1] till ekvation (3) är ett tal α så att $p(\alpha) = 0$. Man säger då också att talet α är ett nollställe till polynomet p.

2.2 Polynomekvationers egenskaper

Enligt satsen om att summan av kontinuerliga funktioner är en kontinuerlig funktion [PB10] ses att ett polynom av en variabel är en kontinuerlig funktion $\forall x \in (-\infty, \infty)$ då alla dess termer x^n, $n = 0, 1, ...$ är kontinuerliga. Kontinuiteten medför att den enda möjligheten för ett polynom att växla tecken är om det passerar $p(\alpha) = 0$ för något α tillhörandes dess definitionsmängd, se sats 2.1. Detta är en viktig egenskap hos polynom och används i Sturmkedjemetoden, se kapitel 3.1.

Sats 2.1. Satsen om mellanliggande värden[2]. *Antag att f är kontinuerlig i det slutna intervallet $a \leq x \leq b$ och antingen $f(a) > 0$ och $f(b) < 0$ eller $f(a) < 0$ och $f(b) > 0$ då existerar en punkt $x_0 \in [a, b]$ så att $f(x_0) = 0$.*

Antag ett polynom $p(x)$ av $\operatorname{grad} n$ då existerar minst ett komplext nollställe till $p(x)$ enligt sats 2.2.

[1] *Anmärkning:* Allmänt är en rot $\neq$ nollställe. En rot är ett tal som löser en ekvation och ett nollställe är ett tal som ger polynomekvationens värde lika med noll

[2] Sats gjord utifrån information från [PB10]

Sats 2.2. Algebrans fundamentalsats[3]**.** *Varje komplex polynomekvation $p(z) = 0$, där* $\operatorname{grad} p \geq 1$*, har minst en komplex rot.*

Följdsats 2.2.1. *Om* $\operatorname{grad} p = n$ *så har ekvationen $p(z) = 0$ exakt n rötter, om varje rot räknas efter dess multiplicitet.*

Antag ett polynom $p(x)$ av grad $n > 1$ har två nollställen i α respektive β, $\alpha \neq \beta$, då existerar minst ett nollställe till $p'(x)$, enligt sats 2.3, mellan α och β.

Sats 2.3. Medelvärdessatsen[4]**.** *Antag att f är kontinuerlig i det slutna intervallet $a \leq x \leq b$ och deriverbar i det öppna intervallet $a < x < b$. Då finns minst en punkt ξ, $a < \xi < b$ sådan att*

$$f(b) - f(a) = f'(\xi)(b - a).$$

En begränsning att $f(b) = f(a) = 0$ i medelvärdessatsen ger Rolles sats, sats 2.4

Sats 2.4. Rolles Sats[5]**.** *Antag $a \neq b$ och $f(a) = f(b) = 0$ då finns det minst en punkt ξ, $a < \xi < b$ där*

$$f'(\xi) = 0.$$

Om ett polynom av grad n har ett nollställe är det möjligt att faktorisera $p(x)$ enligt sats 2.5.

Sats 2.5. Faktorsatsen[6]**.** *Antag att talet α är en rot till polynomekvationen $p(z) = 0$. Då är polynomet $z - \alpha$ en faktor i $p(z)$, vilket innebär att*

$$p(z) = (z - \alpha)q(z), \tag{4}$$

för något polynom $q(z)$, där $\operatorname{grad} q = \operatorname{grad} f - 1$.
Om m är det lägsta antalet faktoriseringar som måste utföras så att $q(\alpha) \neq 0$, vilket innebär $p(z) = (z - \alpha)^m q(z)$ säger vi att polynomet p har en rot av multiplicitet m i punkten $z = \alpha$.

Ett polynom av grad n har maximalt n stycken nollställen i det komplexa planet. Antag att funktionen $p(x)$ är ett polynom av grad n, då existerar ett komplex nollställe α enligt 2.2. Sats 2.5 hävdar att det är möjligt att faktorisera $p(z)$ som $(z - \alpha) \cdot g(z)$ där z är en komplex variabel och $g(z)$ är ett polynom. Med samma resonemang är det möjligt att visa att polynomet $p(z)$ har maximalt n stycken enkla nollställen i det komplexa planet och att $p(z)$ kan skrivas som

$$p(z) = p_0 \prod_{i=1}^{n} (z - \alpha_i), \tag{5}$$

där p_0 är en konstant och α_i är de enkla nollställena till $p(x)$. Om nollställena har multiplicitet m kan $p(x)$ skrivas som

$$p(z) = p_0 \prod_{i=1}^{j} (z - \alpha_i)^{m_i}, \tag{6}$$

där p_0 är en konstant, α_i är nollställena $p(x)$ och m_i är nollställenas multiplicitet. I denna formel är $n = \sum_{i=1}^{j} m_i$ det maximala antalet rötter till polynomet p.

Enligt definitionen för polynom är alla nollställen för alla tänkbara polynom ändliga. Det finns alltså något värde r så att $|\alpha_i| < r$ för alla lösningar α_i till polynomekvationen $p(z) = 0$. Alla nollställen ligger då innanför en cirkel med radie r centrerad i origo[7]. Följande sats ger ett sätt att hitta en sådan cirkel.

[3]Sats hämtad från [PB10]
[4]Sats hämtad från [PB10]
[5]Sats hämtad från [PB10]
[6]Modifierad från [PB10]
[7]*Anmärkning:* Då alla nollställen är reella fås istället ett intervall $-r \leq z \leq r$

Sats 2.6. Största absolutbelopp av nollställe till polynom[8]**.** *Alla nollställen till polynomet p(x) ligger i cirkeln*

$$|x| \leq \max\left(\left|\frac{a_{n-1}}{a_n}\right| + 1, \left|\frac{a_{n-2}}{a_n}\right| + 1, ..., \left|\frac{a_1}{a_n}\right| + 1, \left|\frac{a_0}{a_n}\right| \right).$$

Antag att vi har två tal a och b. Det är möjligt att bilda kvoten mellan dessa två tal som $\frac{a}{b} = q + \frac{r}{b}$ där q är heltalskvoten mellan a och b och r är resten. Frågan är nu om det är möjligt att för två godtyckliga polynom skriva dessa på samma sätt som a och b? Det är möjligt om de två polynomen uppfyller vissa specifika krav enligt sats 2.7.

Sats 2.7. Polynomdivision[9]**.** *Sätt $p(x)$ och $s(x)$ till två polynom och låt* $\operatorname{grad} p \geq \operatorname{grad} s \geq 1$. *Då existerar två polynom $q(x)$ och $r(x)$ så att*

$$\operatorname{grad} r < \operatorname{grad} s,$$

och

$$\frac{p(x)}{s(x)} = q(x) + \frac{r(x)}{s(x)}.$$

Detta kan skrivas om till

$$p(x) = s(x)q(x) + r(x),$$

där $q(x)$ brukar kallas för kvotpolynomet och $r(x)$ för restpolynomet.

Utgående från sats 2.7 är det möjligt att definiera största gemensamma delaren för två polynom.

Definition 2.1. Största gemensamma delare (sgd)[10]**.** *Man säger att ett polynom h är största gemenasamma delare (sgd) till två polynom p och s om h delar både p och s samt att för varje polynom d som delar både p och s gäller även att d delar h.*

För att hitta *sgd* kan man använda sig av Euklides algoritm.

3 Metoder för att lösa polynomekvationer

Genom litteraturstudier har vi bekantat oss med ämnesområdet som i grunden är teoretiskt. Befintliga numeriska metoder för approximativa lösningar av polynom har analyserats och bevisats matematiskt. Vi har sedan testat olika lösningsmetoder inklusive Sturmkedjemetoden, argumentprincipmetoden samt kompanjonmatrismetoden. Beräkningsprogrammet Matlab använder kompanjonmatrismetoden i sin funktion *roots* som hittar rötter till polynomekvationer. Slutligen har vi på olika sätt utvärderat vilken metod som mest effektivt approximerar lösningen till en given polynomekvation. Då all nödvändig teori behandlats och olika tester har utförts har metoderna implementerats i form av Matlab-program som kan lösa en godtycklig polynomekvation i en variabel utgående från någon av metoderna.

3.1 Sturmkedjemetoden

I många fall är endast de reella lösningarna till en polynomekvation av intresse. Ett sätt att hitta alla reella nollställen till ett polynom med reella koefficienter är att använda Sturmkedjor. Herbert S. Wilf utforskar Sturmkedjor och deras användningsområden i sin bok *Mathematics for the Physical Sciences* [Wil62], varifrån vi har hämtat olika definitioner och satser som används i detta kapitel. Idén bakom Sturmkedjor är att utgå ifrån en funktion $f(x)$ som endast ändrar tecken i funktionens nollställen, vilket uppfylls av polynom med enkla nollställen. Wilf inför vissa begränsningar, enligt definition 3.1, på funktionerna som måste uppfyllas för att sekvensen av funktioner ska vara en Sturmkedja.

[8] Sats hämtad och översatt från [Wil62]
[9] Sats hämtad från [VE11]
[10] Sats hämtad från [VE11]

Definition 3.1. Sturmkedjor. *En Sturmkedja är en följd funktioner $f_1(x), ..., f_n(x)$ som är kontinuerliga på ett intervall $x \in (a, b)$ och uppfyller följande axiom:*

$$f_{k-1}(x_0) \text{ och } f_{k+1}(x_0) \text{ är nollskilda och av olika tecken då } x_0 \text{ är ett nollställe till } f_k(x) \quad (7)$$

$$f_n(x) \neq 0 \text{ för alla } x \in (a, b) \quad (8)$$

Då polynom är kontinuerliga funktioner för alla x är en sekvens polynom $p_1(x), ..., p_n(x)$ som uppfyller definition 3.1 en Sturmkedja. Om $p(x)$ har fler än ett nollställe på intervallet (a, b) existerar, enligt Rolles sats, sats 2.4, minst ett nollställe mellan a och b för dess derivata p'.

Definition 3.1 innebär att om ett nollställe existerar i sekvensen $p_1(x), ..., p_n(x)$ så måste de två omkringliggande funktionerna ha motsatt tecken samt att den sista funktionen $p_n(x)$ inte växlar tecken. I Sturmkedjan är det möjligt att använda godtyckliga polynom förutsatt att de uppfyller definition 3.1 men att hitta dessa generella polynom blir snabbt omöjligt. En Sturmkedja kan konstrueras på ett enkelt sätt med följande metod som tas upp i kapitel 3.1.1.

3.1.1 Konstruktion av en Sturmkedja

Börja med ett polynom p med enkla nollställen och sätt $p_1(x) = p(x)$, $p_2(x) = p_1'(x)$. Enligt sats 2.7 kan p_1 och p_2 skrivas som $p_1(x) = a_1(x)p_2(x) + r_1(x)$ där $a_1(x)$ är kvoten och $r_1(x)$ resten. Nästa term i polynomsekvensen sätts till $p_3(x) = -r_1(x)$ vilket ger för ett allmänt sturmkedjepolynom $p_{k-1} = a_{k-1}p_k - p_{k+1}$. Antag att α är ett nollställe till polynomet p_k, vilket innebär att $p_k(\alpha) = 0$, då ger ekvationen innan att $p_{k-1}(\alpha) = -p_{k+1}(\alpha)$ som därmed uppfyller det första kravet i definition 3.1. Detta leder till att ett polynom av grad n utan multiplicitet kommer att ha en Sturmkedja bestående av maximalt $n + 1$ polynom, eftersom graden på sturmkedjepolynomen $p_k(x)$ minskar stegvis efter varje polynomdivision 2.7.

Däremot om $p(x)$ har nollställen med multiplicitet kommer $p(x)$ och $p'(x)$ att få värdet noll i dessa punkter, vilket bryter mot det första axiomet i definitionen av Sturmkedjor 3.1. Det finns dock ett sätt att kringgå det här problemet. Det är möjligt att konstruera en Sturmkedjan som kan användas även i detta fall efter vissa omskrivningar. Från ett polynom av grad n med multiplicitet fås en kedja som består av n eller färre polynom och där sista polynomet i kedjan inte är konstant. Låt $p_1(x), ..., p_m(x)$ vara denna kedja där $m < n + 1$, då kommer $p_m(x)$ vara största gemensamma delare, se definition 2.1, till $(p_1(x), ..., p_{m-1}(x))$ och varje polynom i $(p_1(x), ..., p_m(x))$ kan divideras med $p_m(x)$. Slutligen fås då en ny kedja som bildar en Sturmkedja och har formen:

$$\frac{p_1(x)}{p_m(x)}, \frac{p_2(x)}{p_m(x)}, ..., \frac{p_{m-1}(x)}{p_m(x)}, 1.$$

Detta överensstämmer inte helt med konstruktionen av Sturmkedjor beskriven ovan eftersom vanligtvis är $p_1(x) = p(x)$ och $p_2(x) = p'(x)$. Polynomet $p_2(x)/p_m(x)$ är dock inte derivatan av $p_1(x)/p_m(x)$. Det spelar emellertid ingen roll som vi kommer att se i beviset av sats 3.1.

3.1.2 Variation av en Sturmkedja

För att hitta antalet och placeringen av de reella nollställena till ett polynom $p_1(x)$ med hjälp av Sturmkedjan $p_1(x), ..., p_n(x)$ på intervallet (a, b) måste Sturmkedjans variation hittas. Variationen för Sturmkedjan definieras enligt definition 3.2

Definition 3.2. Variation. *Låt $x_0 \in R$, då definieras variationen av sekvensen $p_1(x), ..., p_n(x)$ i punkten x_0 som antalet teckenväxlingar mellan + och − då sekvensen nedan genomgås i ordning från vänster till höger.*

$$(\text{sgn}(p_1(x_0)), \text{sgn}(p_2(x_0)), ..., \text{sgn}(p_n(x_0)))$$

Exempel 3.1. Beräkna variationen av Sturmkedjan för polynomet $p(x) = 4x^4 + 2x^2 - 1$ i punkten $x = -2$.

Det givna polynomet har Sturmkedjan $(4x^4 + 2x^2 - 1, 16x^3 + 4x, 1 - x^2, -5x, -1)$ som i punkten $x = -2$ ändrar tecken enligt $(+, -, -, +, -)$. Kedjan ändrar tecken mellan den första och andra, tredje och fjärde och fjärde och femte polynomet och variationen blir 3 i punkten $x = -2$, alltså blir $V(-2) = 3$.

3.1.3 Antalet reella nollställen

Det återstår nu att slutligen hitta alla reella nollställen till en given polynomekvation. I följande sats visas att Sturmkedjor och deras variation kan användas för att hitta dessa nollställen.

Sats 3.1. *Antalet reella nollställen till polynomet $p(x)$ inom intervallet $x \in [a, b]$ är skillnaden i variationen mellan dessa punkter, det vill säga $V(a) - V(b)$.*

Bevis. Betrakta variationen $V(x)$ i en punkt $x \in [a, b]$. Låt $x = a$, notera värdet på $V(a)$ och låt sedan x gå mot b. Om för något x variationen ändras måste någon funktion $p_k(x)$ byta tecken. För denna punkt x_0 gäller då att $p_k(x_0) = 0$. På grund av definitionen för Sturmkedjor står klart att $k \neq n$. Av definitionen är också givet att kringliggande funktioner $p_{k-1}(x)$ och $p_{k+1}(x)$ har olika tecken kring x_0. Detta innebär att då nollstället x_0 för p_k, $k > 1$, passeras sker ingen ändring av variationen, vilket illustreras i följande exempel:

vänster om x_0		
$p_{k-1}(x)$	$p_k(x)$	$p_{k+1}(x)$
$+$	$+$	$-$
$+$	$-$	$-$
$-$	$+$	$+$
$-$	$-$	$+$

höger om x_0		
$p_{k-1}(x)$	$p_k(x)$	$p_{k+1}(x)$
$+$	$-$	$-$
$+$	$+$	$-$
$-$	$-$	$+$
$-$	$+$	$+$

Således kan variationen enbart ändras i de punkter där $p_1(x) = p(x) = 0$. Låt nu x_0 vara ett nollställe till $p_1(x)$. Då $p_2(x) = p'(x)$ finns endast följande möjligheter för Sturmkedjans värden:

vänster om x_0	
$p_1(x)$	$p_2(x)$
$+$	$-$
$-$	$+$

höger om x_0	
$p_1(x)$	$p_2(x)$
$-$	$-$
$+$	$+$

Nu ses att variationen minskar med 1 då ett nollställe till $p(x)$ passeras, och endast då. Således blir antalet nollställen som passeras på ett intervall (a, b) lika med skillnaden i variation mellan ändpunkterna, det vill säga $V(a) - V(b)$. Notera att variationen är avtagande vid varje övergång och har därmed sitt största värde i punkten $x = a$.

□

Det är möjligt att använda detta för att ta fram ett litet intervall runt ett nollställe.

Exempel 3.2. Hitta antalet reella lösningar till polynomet $p(x) = 4x^4 + 2x^2 - 1$ med hjälp av Sturmkedjor.

Lösning. Enligt avsnitt 3.1.1 ovan finns det en metod för att beräkna Sturmkedjan. Vi börjar med att sätta $p_1(x) = 4x^4 + 2x^2 - 1$. Nästa steg är att sätta $p_2(x)$ till $p_1'(x)$ och ta bort gemensamma faktorer, alltså $p_1'(x) = 16x^3 + 4x \;\rightarrow\; p_2(x) = 4x^3 + x$. Nu ges resten av polynomen i Sturmkedjan av $-\text{rest}[\frac{p_k(x)}{p_{k+1}(x)}]$, så $p_3(x) = -\text{rest}[\frac{p_1(x)}{p_2(x)}]$:

$$
\begin{array}{r|l}
4x^4 + 2x^2 - 1 & 4x^3 + x \\ \cline{2-2}
- \quad 4x^4 + x^2 & \qquad x \\ \hline
x^2 - 1 &
\end{array}
$$

$r_1 = x^2 - 1 \;\rightarrow\; p_3(x) = 1 - x^2$. Vi upprepar samma procedur, $p_4(x) = -\text{rest}[\frac{p_2(x)}{p_3(x)}]$:

$$
\begin{array}{r|l}
4x^3 + x & 1 - x^2 \\ \cline{2-2}
- \quad 4x^3 - 4x & -4x - 1 \\ \hline
5x &
\end{array}
$$

$r_2 = 5x \;\rightarrow\; p_4(x) = -5x$. Proceduren upprepas igen, $p_5(x) = -\text{rest}[\frac{p_3(x)}{p_4(x)}]$:

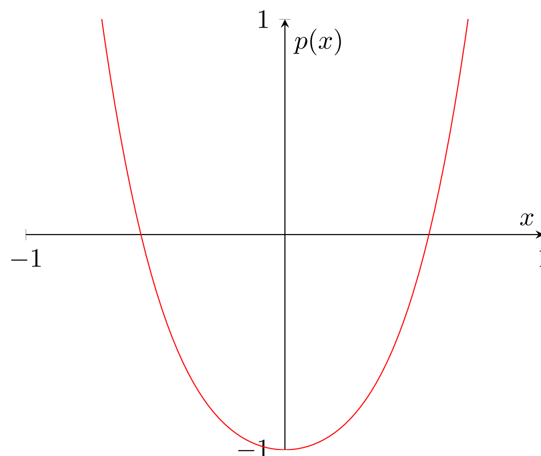
$$
\begin{array}{r|l}
- \quad -x^2+1 & -5x \\
x^2 & -(1/5)x \\
\hline
1 &
\end{array}
$$

$r_3 = 1 \quad \rightarrow \quad p_5(x) = -1$. Sturmkedjan ges nu av $(p_1(x), p_2(x), p_3(x), p_4(x), p_5(x))$.

Tabell 1: *Visar variationen av tecken i Sturmkedjan för olika x-värden.*

x	$p_1(x)$	$p_2(x)$	$p_3(x)$	$p_4(x)$	$p_5(x)$	$V(x)$
$-\infty$	+	−	−	+	−	3
−1	+	−	0	+	−	3
0	−	0	+	0	−	2
1	+	+	0	−	−	1
∞	+	+	−	−	−	1

Tabell 1 visar Sturmkedjans värden i punkterna $x = -\infty, -1, 0, 1, \infty$, notera att polynomet $p(x)_1 = 4x^4 + 2x^2 - 1$ har två reella nollställen mellan $(-1, 1)$. Det första nollstället ligger mellan $(-1, 0)$ och det andra ligger mellan $(0, 1)$ vilket kan ses tydligt från ändringen i variation vid dessa punkter. Figur 1 visar polynomet $p(x) = 4x^4 + 2x^2 - 1$ som illustrerar att metoden fungerar.



Figur 1: *Polynomet $4x^4 + 2x^2 - 1$ nära dess reella nollställen*

3.2 Argumentprincipmetoden

Betrakta ett polynom $p(x)$ av grad n som enligt följdsatsen till algebrans fundamentalsats, sats 2.2.1, har exakt n stycken nollställen i det komplexa planet. Tidigare löstes polynomekavationer med reella koefficienter och nollställen med hjälp av Sturmkedjor, men denna metod kan ej tillämpas på komplexa polynom. Till exempel så är variationen inte definierad för komplexa termer och det finns ingen enkel generalisering för variationen av komplexa polynom. Sturmkedjor kan således nyttjas till att hitta reella lösningar till en polynomekvation men för att hitta alla lösningar krävs att andra metoder används och en av dem vi ska titta närmare på är argumentprincipen.

3.2.1 Argumentprincipen

Antag att vi har ett polynom $p(z) = \sum_{j=0}^{n} a_j z^j$ där z är en komplex variabel $z = x + iy$ och $a_j \in \mathbb{C}$. Ett sätt att hitta nollställen till detta polynom är att fastställa hur många nollställen som ligger i ett givet område i det komplexa planet. I boken *A First Course in Complex Analysis* [Bec+02] presenteras möjligheten att göra detta genom införandet av de satser och definitioner som återges

i detta kapitel. Polynomet p är en kontinuerlig och differentierbar funktion som saknar poler i hela det komplexa planet. Nu införs två nödvändiga definitioner för funktioner.

Definition 3.3. Holomorf. *En funktion $f : G \to \mathbb{C}$ är holomorf i en punkt z om den är differentierbar i en öppen cirekelskiva centrerad i z. Om f är differentierbar för alla punkter på ett öppen mängd $E \subseteq G$ sägs f vara holomorf på E.*

Definition 3.4. Meromorf. *En funktion f är meromorf på ett område G om f är holomorf i G utom i funktionens poler.*

Polynomet p är alltså holomorft i alla punkter z_0 på det komplexa planet och därmed även meromorft på samma område. Slutligen presenteras en sista nödvändig definition.

Definition 3.5. Homotopi. *Antag att $\gamma_0(t)$ och $\gamma_1(t)$, $0 \leq t \leq 1$ är slutna kurvor i området $G \subseteq \mathbb{C}$. Då är γ_0 G-homotop till γ_1, vilket vi skriver $\gamma_0 \sim_G \gamma_1$, om det finns en kontinuerlig funktion $h : [0,1]^2 \to G$ så det gäller för alla $s, t \in [0,1]$ att*

$$\begin{aligned} h(t,0) &= \gamma_0(t), \\ h(t,1) &= \gamma_1(t), \\ h(0,s) &= h(1,s). \end{aligned}$$

Två kurvor är alltså homotopa om de kan kontinuerligt deformeras från den ena till den andra. Om en kurva γ i området G är homotop till punkten noll, $\gamma \sim_G 0$, kallas γ nollhomotop. Nu kan den titulära satsen presenteras

Sats 3.2. Argumentprincipen. *Antag att f är meromorf innanför γ, då γ är en positivt orienterad, enkel, sluten, styckvis kontinuerlig kurva som inte passerar något nollställe eller pol för f, och $\gamma \sim_G 0$. Låt $Z(f,\gamma)$ vara antalet nollställen till f innanför γ räknat efter multiplicitet och $P(f,\gamma)$ vara antalet poler innanför γ räknat efter ordning. Då gäller att*

$$\frac{1}{2\pi i} \int_\gamma \frac{f'}{f} dz = Z(f,\gamma) - P(f,\gamma).$$

Bevis. Vi nöjer oss med att bevisa satsen då f är ett godtyckligt polynom med alla nollställen innanför γ. Varje polynom $p(z)$ kan enligt faktorsatsen, sats 2.5, uttryckas som

$$p(z) = k(z - z_1)^{n_1}(z - z_2)^{n_2}...(z - z_j)^{n_j},$$

där n_i betecknar multipliciteten hos nollstället z_i. Vidare blir den logaritmiska derivatan

$$\frac{p'}{p} = \frac{n_1(z - z_1)^{n_1 - 1}...(z - z_j)^{n_j} + ... + n_j(z - z1)^{n_1}...(z - z_j)^{n_j - 1}}{(z - z_1)^{n_1}(z - z_2)^{n_2}...(z - z_j)^{n_j}} = \frac{n_1}{z - z_1} + ... + \frac{n_j}{z - z_j}.$$

Nu kan den ursprungliga integralen beräknas enligt följande

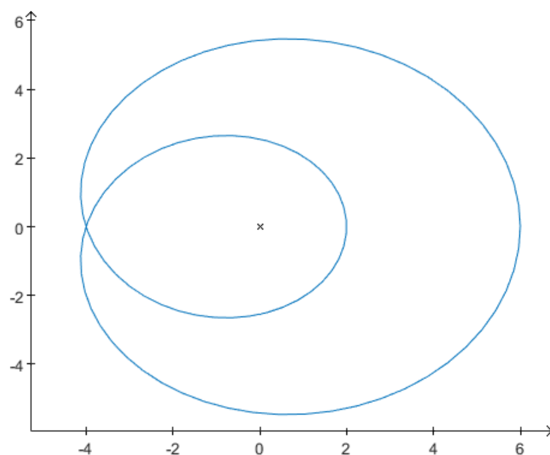
$$\int_\gamma \frac{p'}{p} dz = n_1 \int_\gamma \frac{dz}{z - z_1} + ... + n_j \int_\gamma \frac{dz}{z - z_j} = 2\pi i(n_1 + ... + n_j).$$

Då polynom saknar poler är detta ekvivalent med uttrycket i satsen. □

Namnet kommer från det faktum att integralen av den logaritmiska derivatan längs med en kurva γ, $\int_\gamma \frac{f'}{f} dz$ kan ses som skillnaden i argumentet av f då z följer γ. Detta kan illustreras med ett exempel.

Exempel 3.3. Hitta antalet nollställen till polynomet $p(z) = z^2 + z$ som ligger i en cirkel med radie 2 centrerad i origo med hjälp argumentprincipen.

Lösning. Då p kan faktoriseras till $p = z(z + 1)$ ses lätt att polynomet har nollställena $z = 0$ och $z = 1$, vilka båda ligger i cirkeln. Om nu cirkeln beskrivs med hjälp av vinkeln θ fås $z(\theta) = 2e^{\theta i}$ och $p(z(\theta)) = 4e^{2\theta i} + 2e^{\theta i} = 4\cos 2\theta + 2\cos\theta + i(4\sin 2\theta + 2\sin\theta)$ är polynomets avbildning på cirkeln. Avbildningens kurva då $0 \leq \theta \leq 2\pi$ kan ses i Figur 2



Figur 2: Avbildningen av polynomet $p(z) = z^2 + z$ på cirkeln $x^2 + y^2 = 4$. Notera att den vertikala axeln är imaginär.

Från figuren ses att kurvan omlöper origo två gånger, det vill säga argumentet ändras med 4π, vilket tyder på att det finns två nollställen till polynomet i den givna cirkeln. Nu undersöks om detta överensstämmer med resultatet som fås med hjälp av argumentprincipen. Med γ satt till cirkeln med radie 2 och $\frac{p'}{p} = \frac{2z+1}{z^2+z}$ fås

$$\frac{1}{2\pi i}\int_\gamma \frac{p'}{p}dz = \frac{1}{2\pi i}\int_0^{2\pi} \frac{p'(z(\theta))}{p(z(\theta))}z'(\theta)d\theta = \frac{1}{2\pi i}\int_0^{2\pi} \frac{i(1+4e^{i\theta})}{1+2e^{i\theta}}d\theta = 2.$$

Argumentprincipen och den grafiska tolkningen ger alltså båda det korrekta svaret att det finns två nollställen i cirkeln.

3.2.2 Antalet komplexa nollställen

Likt hur variationen av en Sturmkedja gav antalet nollställen på ett intervall ger argumentprincipen antalet nollställen på ett område i det komplexa planet inneslutet av kurvan γ. Nu kvarstår att söka i området efter mer exakta lösningar till ett givet polynom.

Enligt argumentprincipen kan γ väljas som en godtycklig enkel, sluten, styckvis kontinuerlig kurva som innesluter alla nollställen till polynomet. En kvadrat med sidan R centrerad i origo är en sådan kurva om R väljs tillräckligt stort. För att utföra sökningen efter nollställenas position i kvadraten delas den upp i sina fyra kvadranter. Nu används argumentprincipen på varje ny kvadrat och om integralen har ett värde större än noll finns nollställen i den mindre kvadraten. För varje kvadrat där nollställe hittas delas även denna i kvadranter och processen upprepas tills dess att en kvadrat med nollställen hittas vars sida r är tillräckligt liten. Denna kvadrat ses då som en approximation av ett nollställe till polynomet. Då en sådan djupsökning har utförts i alla fyra kvadranter till den ursprungliga kvadraten har alla nollställen till polynomet hittats.

Den första kvadratens utformning garanterar att inget nollställe ligger på dess rand. Det finns dock inga garantier att detta stämmer för alla kvadrater under sökningen. Om ett nollställe x_0 till polynomet ligger på randen till en kvadrat fås att $p(x_0) = 0$ och $\frac{p'(x_0)}{p(x_0)}$ blir odefinierat. Således kan antalet nollställen inom kvadraten ej fastställas. Denna komplikation kan hanteras genom att flytta kvadraten då en odefinierad integral påträffas i sökningen och istället använda integralen för den nya kurvan.

3.3 Kompanjonmatrismetoden

I linjär algebra kan man hitta nollställen till ett polynom genom att skapa en matris och beräkna dess egenvärden och egenvektorer. Micheal T. Heath beskriver metoden i sin bok *Scientific Computiong An Intorductory Survey* [Hea02]. En $n \times n$ matris $\mathbf{A}$ har egenvärden λ_i och egenvektorer

$\mathbf{v}_i$ om $\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i$. En $n \times n$ matris har maximalt n egenvärden och n egenvektorer. Att beräkna egenvärdena är likvärdigt med att hitta lösningen till $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$, som ger upphov till ett polynom där egenvärdena λ_i är nollställena till polynomet. Ekvationen $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$ kallas för det karakteristiska polynomet. Matrisen som skapas kan ses som en kompanjon till polynomet och kallas därför kompanjonmatris.

Låt $p(x) = x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0$; då ges kompanjonmatris för p av

$$\mathbf{K}(p) = \begin{bmatrix} 0 & 0 & \dots & -a_0 \\ 1 & 0 & \dots & -a_1 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 1 & -a_{n-1} \end{bmatrix}. \tag{9}$$

Rötterna för det karakteristiska polynomet $\det(x\mathbf{I} - \mathbf{K}) = 0$ är nollställena för polynomet p och egenvärdena för kompanjonmatrisen. Elementen i den sista kolonnen i kompanjonmatrisen är polynomets koefficienter med motsatt tecken, ettor längs sub-diagonalen och nollor på resterande platser. Således är nollställena för p egenvärdena för $\mathbf{K}(p)$.

För ett polynom p definieras kompanjonmatrisen så att det karakteristiska polynomet $\det(x\mathbf{I} - \mathbf{K}) = p(x)$.

Vi visar att $p(x) = \det(x\mathbf{I} - \mathbf{K})$ med hjälp av induktion. För $n = 1$ är $\mathbf{K} = [-a_0]$ och $\det(x\mathbf{I} - \mathbf{K}) = \det([x + a_0]) = x + a_0$. Antag nu att det är sant för $n-1$ och visa att det är sant för n. Vi använder oss av kofaktor utveckling på översta raden.

$$\det(x\mathbf{I} - \mathbf{K}) = \begin{vmatrix} x & 0 & \dots & 0 & a_0 \\ -1 & x & \dots & 0 & a_1 \\ 0 & -1 & \dots & 0 & a_2 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & \dots & -1 & x + a_{n-1} \end{vmatrix}$$

$$= x \begin{vmatrix} x & 0 & \dots & 0 & a_1 \\ -1 & x & \dots & 0 & a_2 \\ \vdots & \vdots & \ddots & & \vdots \\ 0 & 0 & \dots & -1 & x + a_{n-1} \end{vmatrix} + (-1)^{n+1} a_0 \begin{vmatrix} -1 & x & \dots & 0 \\ 0 & -1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & -1 \end{vmatrix}.$$

Enligt induktionsantagandet är den första determinanten

$$x^{n-1} + a_{n-2}x^{n-2} + \dots + a_2 x + a_1. \tag{10}$$

Den andra determinanten är $(-1)^{n-1}$ eftersom det är en $(n-1) \times (n-1)$ triangulär matris och då blir determinanten produkten av matrisens diagonalelement. Alltså får vi

$$\begin{aligned} \det(x\mathbf{I} - \mathbf{K}) &= x(x^{n-1} + a_{n-1}x^{n-2} + \dots + a_2 x + a_1) + (-1)^{n+1} a_0 (-1)^{n-1} \\ &= x^n + a_{n-1}x^{n-1} + \dots + a_1 x + a_0 \\ &= p(x). \end{aligned} \tag{11}$$

Genom induktion har vi visat att påståendet är sant för ett polynom av grad n.

Varje metod som beräknar egenvärdena för $\mathbf{K}(p)$ (ekvation 9) är en metod för att beräkna nollställen för $p(x)$ och tvärtom. Matlabs polynomlösare *roots* beräknar nollställena till ett polynom via kompanjonmatrisen. En metod för att beräkna egenvärdena till kompanjonmatrisen är QR-algoritmen.

3.3.1 QR-algoritmen

QR-algoritmen är en metod för att räkna ut egenvärdena till en kvadratisk matris $\mathbf{A}$. Heath beskriver metoden i sin bok [Hea02], varifrån vi har inhämtat exempel 3.4 och 3.5. Idén bakom algoritmen baseras på QR-faktorisering, vilket beskrivs mer ingående i avsnitt 3.3.2. I korthet innebär metoden faktorisering av matrisen $\mathbf{A}$: $\mathbf{A} = \mathbf{QR}$ där $\mathbf{Q}$ är en ortogonal matris, $\mathbf{Q}^{-1} = \mathbf{Q}^T$, $\mathbf{QQ}^T = \mathbf{I}$, och $\mathbf{R}$ är en övertriangulär matris. Om $\mathbf{A}$ skulle vara komplex är $\mathbf{Q}$ en unitär matris istället för en ortogonal sådan. Därefter multipliceras $\mathbf{Q}$ och $\mathbf{R}$ i omvänd ordning: $\mathbf{A}_0 = \mathbf{RQ}$. QR-faktoriseringen upprepas sedan för $\mathbf{A}_0$: $\mathbf{A}_0 = \mathbf{Q}_1\mathbf{R}_1$ och $\mathbf{Q}_1$ och $\mathbf{R}_1$ multipliceras i omvänd ordning: $\mathbf{A}_1 = \mathbf{R}_1\mathbf{Q}_1$.

Algoritm 1: QR-algoritmen [Hea02]

```
1 A_0 = A
2 for k = 1, 2, ... do
3 |  Beräkna QR-faktoriseringen
4 |  Q_k R_k = A_{k-1}
5 |  A_k = R_k Q_k
6 end
```

Exempel 3.4. Illustrering av QR-algoritmen.

För att illustrera QR-algoritmen tillämpar vi den på en symmetrisk matris

$$\mathbf{A} = \begin{bmatrix} 2.9766 & 0.3945 & 0.4198 & 1.1159 \\ 0.3945 & 2.7323 & -0.3097 & 0.1129 \\ 0.4198 & -0.3097 & 2.5675 & 0.6079 \\ 1.1159 & 0.1129 & 0.6079 & 1.7231 \end{bmatrix},$$

som har egenvärden $\lambda_1 = 4$, $\lambda_2 = 3$, $\lambda_3 = 2$, $\lambda_4 = 1$. Om vi QR-faktoriserar och sedan skapar den omvända produkten, RQ, får vi

$$\mathbf{A}_1 = \begin{bmatrix} 3.7703 & 0.1745 & 0.5126 & -0.3934 \\ 0.1745 & 2.7675 & -0.3872 & 0.0539 \\ 0.5126 & -0.3872 & 2.4019 & -0.1241 \\ -0.3934 & 0.0539 & -0.1241 & 1.0603 \end{bmatrix}.$$

Diagonalen närmar sig egenvärdena och de andra värdena i matrisen minskar i storlek. Fortsätter vi iterera ett par gånger får vi

$$\mathbf{A}_2 = \begin{bmatrix} 3.9436 & 0.0143 & 0.3046 & 0.1038 \\ 0.0143 & 2.8737 & -0.3362 & -0.0285 \\ 0.3046 & -0.3362 & 2.1785 & 0.0083 \\ 0.1038 & -0.0285 & 0.0083 & 1.0042 \end{bmatrix},$$

och

$$\mathbf{A}_3 = \begin{bmatrix} 3.9832 & -0.0356 & 0.1611 & -0.0262 \\ -0.0356 & 2.9421 & -0.2432 & 0.0098 \\ 0.1611 & -0.2432 & 2.0743 & 0.0047 \\ -0.0262 & 0.0098 & 0.0047 & 1.0003 \end{bmatrix}.$$

Diagonalen är nu ännu närmare egenvärdena och de andra värdena i matrisen har fortsatt minska i storlek. Endast några iterationer till skulle behövas för att beräkna egenvärdena med full precision som visas.

3.3.2 QR-faktorisering

QR-faktorisering är en ortogonal transformation till en triangulär form [Hea02]. Om $\mathbf{A}$ är en $n \times n$ matris, så kan $\mathbf{A}$ skrivas på formen

$$\mathbf{A} = \mathbf{QR},$$

där $\mathbf{Q}$ är en ortogonal $n \times n$ matris och $\mathbf{R}$ är en övertriangulär $n \times n$ matris. Det finns flera sätt att beräkna QR-faktoriseringen. En populär metod är att använda Householdertransformation, som är en matris av formen

$$\mathbf{H} = \mathbf{I} - 2\frac{\mathbf{v}\mathbf{v}^T}{\mathbf{v}^T\mathbf{v}},$$

där $\mathbf{v}$ är en nollskild vektor. Matrisen $\mathbf{H}$ är både symmetrisk och ortogonal, så $\mathbf{H} = \mathbf{H}^T = \mathbf{H}^{-1}$. Ta en vektor $\mathbf{a} = (a_1, a_2, \ldots, a_n)$, då väljs $\mathbf{v}$ så att alla komponenter till $\mathbf{a}$ försvinner utom den första,

$$\mathbf{Ha} = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \alpha \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \alpha \mathbf{e}_1.$$

Använder vi formeln för $\mathbf{H}$, får vi

$$\alpha \mathbf{e}_1 = \mathbf{Ha} = \big(\mathbf{I} - 2\frac{\mathbf{v}^T\mathbf{v}}{\mathbf{v}\mathbf{v}^T}\big)\mathbf{a} = \mathbf{a} - 2\mathbf{v}\frac{\mathbf{v}^T\mathbf{a}}{\mathbf{v}\mathbf{v}^T},$$

således blir

$$\mathbf{v} = (\mathbf{a} - \alpha \mathbf{e}_1)\frac{\mathbf{v}^T\mathbf{v}}{2\mathbf{v}^T\mathbf{a}}.$$

Skalärfaktor är dock inte relevant eftersom den försvinner om vi använder formeln för $\mathbf{H}$, så vi kan skriva

$$\mathbf{v} = \mathbf{a} - \alpha \mathbf{e}_1.$$

För att bevara normen måste $\alpha = \pm \|\mathbf{a}\|_2$ och tecknet bör väljas efter $\alpha = -\operatorname{sgn}(a_1)\|\mathbf{a}\|_2$ så att α och a_1 inte tar ut varnadra.

För att få en bättre förståelse för transformationen från vektorn $\mathbf{a}$ till $\pm \|\mathbf{a}\|_2 \mathbf{e}_1$ kan man föreställa sig det geometriskt. Vektorn $\mathbf{a}$ transformeras till den första koordinataxeln där alla andra komponeterer är noll. Normen bevaras genom att spegla $\mathbf{a}$ genom någon av de två hyperplanen. Hyperplanen är två vinkelräta strålar som delar vinklarna mellan $\mathbf{a}$ och $\mathbf{e}_1$ mitt i tur (bisektriser). Ett sådant hyperplan ges av $\operatorname{span}(\mathbf{v})^\perp = \{x : v^T x = 0\}$ där $\mathbf{v}$ är en nollskild vektor. Vektorn $\mathbf{v}$ är parallell med $\mathbf{a} - \alpha \mathbf{e}_1$ där $\alpha = \pm \|\mathbf{a}\|_2$ ($\pm$ beror på vilket hyperplan $\mathbf{a}$ speglas genom). Den ortogonala projektionen på $\operatorname{span}(\mathbf{v})$ ges av $\mathbf{P} = \mathbf{v}(\mathbf{v}^T\mathbf{v})^{-1}\mathbf{v}^T = (\mathbf{v}\mathbf{v}^T)/(\mathbf{v}^T\mathbf{v})$ och projektionen på $\operatorname{span}(\mathbf{v})^\perp$ ges av $\mathbf{I}-\mathbf{P}$. Således är $(\mathbf{I}-\mathbf{P})\mathbf{a} = \mathbf{a} - \mathbf{v}(\mathbf{v}^T\mathbf{a})/(\mathbf{v}^T\mathbf{v})$ projektionen av $\mathbf{a}$ på $\operatorname{span}(\mathbf{v})^\perp$, men för att nå den första koordinataxeln behöver vi gå dubbelt så långt. Därför ges transformationen vi behöver av $\mathbf{H} = \mathbf{I} - 2\mathbf{P} = \mathbf{I} - 2(\mathbf{v}\mathbf{v}^T)/(\mathbf{v}^T\mathbf{v})$.

Exempel 3.5. Illustrering av Householder-transformation.

För att ge exempel på konstruktionen just beskriven, använder vi vektorn

$$\mathbf{a} = \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix}.$$

Vi väljer vektorn $\mathbf{v}$ enligt

$$\mathbf{v} = \mathbf{a} - \alpha \mathbf{e}_1 = \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix} - \alpha \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix} - \begin{bmatrix} \alpha \\ 0 \\ 0 \end{bmatrix},$$

där $\alpha = \pm \|\mathbf{a}\|_2 = \pm 5$. Eftersom a_1 är positiv väljer vi $\alpha = -5$. Vi får

$$\mathbf{v} = \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix} - \begin{bmatrix} -5 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 8 \\ 4 \\ 0 \end{bmatrix}.$$

För att kontrollera att transformationen stämmer kan vi beräkna

$$\mathbf{Ha} = \mathbf{a} - 2\frac{\mathbf{v}^T\mathbf{a}}{\mathbf{v}\mathbf{v}^T}\mathbf{v} = \begin{bmatrix} 3 \\ 4 \\ 0 \end{bmatrix} - 2\frac{40}{80}\begin{bmatrix} 8 \\ 4 \\ 0 \end{bmatrix} = \begin{bmatrix} -5 \\ 0 \\ 0 \end{bmatrix},$$

här ser vi att normen är bevarad.

3.3.3 Algoritm Householder QR-faktorisering

Hittills har vi visat hur man konstruerar en Householder transformation för att transformera en given vektor. Mer generellt för en n-vektor $\mathbf{a}$, så kan $\mathbf{a}$ delas till en partition av två vektorer

$$\mathbf{a} = \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \end{bmatrix},$$

där $\mathbf{a}_1$ är en $(k-1)$-vektor, $1 \leq k < n$. Om vi väljer Householder-vektorn enligt

$$\mathbf{a} = \begin{bmatrix} 0 \\ \mathbf{a}_2 \end{bmatrix} - \alpha\mathbf{e}_k,$$

där $\alpha = -\operatorname{sgn}(a_1)\,\|\mathbf{a}\|_2$, då kommer Householder transformationen att ta bort de $n-k$ sista komponenterna i $\mathbf{a}$. Använder vi Householder transformationen upprepade gånger för $k = 1, \ldots, n$ kan vi ta bort subdiagonalen och elementen under i en $(n \times n)$ matris $\mathbf{A}$, genom att transformera en kolonn i taget från vänster till höger, och på det sättet reducera matrisen till övertriangulär form. QR-faktorisering genom Householdertranformation sammanfattas i Algoritm 2 där den j:te kolonnen i $\mathbf{A}$ ges av $\mathbf{a}_j$.

Algoritm 2: Householder QR-faktorisering [Hea02]

1 **for** $k = 1, \ldots, n$ **do**
2     $\alpha = -\operatorname{sgn}(a_{kk}\sqrt{a_{kk}^2 + \cdots + a_{nk}^2})$
3     $\mathbf{v}_k = [0 \ \ldots \ 0 \ a_{kk} \ \ldots \ a_{nk}]^T - \alpha\mathbf{e}_k$
4     $\beta_k = \mathbf{v}_k^T\mathbf{v}_k$
5   **if** $\beta_k = 0$ **then**
6   Fortsätt med nästa k
7   **for** $j = k, \ldots, n$ **do**
8             $\gamma = \mathbf{v}_k^T\mathbf{a}_j$
9             $\mathbf{a}_j = \mathbf{a}_j - (2\gamma_j/\beta_k)\mathbf{v}_k$
10   **end**
11   **end**
12 **end**

4 Algoritmer för programimplementering

Nedan följer algoritmerna som kan användas för implementering av sturmkedjemetoden, argumentprincipen samt kompanjonmatrismetoden. Ett försök har gjorts för att göra algoritmerna så tydliga som möjligt utan att gå in för mycket på variabler och bakomliggande tankar och idéer som använts i vårt program. Den första raden i varje algoritm ger namnet på den kod som vi har skrivit som kan hittas i appendix, exempelvis på rad 1 algoritm 3 står det *sturm.m* som går att hitta i appendix. På varje algoritm ges ett rekommenderat input och output till varje funktion.

4.1 Algoritmer för Sturmkedjemetoden

Här presenteras en algoritm för att beräkna sturmkedjan för ett polynom p.

Algoritm 3: Sturmkedja

1 sturm.m ;
  Input : Polynom $\mathbf{p}(x)$
  Output: Matris som innehåller Sturmkedjepolynomen
2 $\mathbf{p}_1$ = derivatan av $\mathbf{p}$
3 Polynomdividera $\mathbf{p}$ med $\mathbf{p}_1$; $\mathbf{p} = \mathbf{q} * \mathbf{p1} + \mathbf{r}$
4 $\mathbf{p}_2 = -\mathbf{r}$
5 Lägg till $\mathbf{p}_2$ som i matris
6 Upprepa tills r är av grad noll (konstant), med $\mathbf{p} = \mathbf{p}_1$ och $\mathbf{p}_1 = \mathbf{p}_2$

Variationen beräknar antal reella nollställen ett polynom p(x) har inom ett intervall. Nedan representeras algoritmen.

Algoritm 4: Variationen av en sturmkedja

1 variation.m ;
  Input : Matris $\mathbf{U}$ med Sturmkedjepolynom, Två gränspunkter a och b
  Output: Skillnaden i variation mellan punkterna a och b, $\mathbf{V}(a) - \mathbf{V}(b)$
2 Utvärdera Sturmkedjepolynomen från matrisen i punkterna a och b, $\mathbf{s}_a = \mathbf{U}(a)$, $\mathbf{s}_b = \mathbf{U}(b)$
3 **if** $\mathbf{s}_{ab}(j) * \mathbf{s}_{ab}(j+1) < 0$ **then**
4 | $\mathbf{V}_{ab} = \mathbf{V}_{ab} + 1$
5 **else if** $\mathbf{s}(i) * \mathbf{s}(i+1) == 0$ **then**
6 | $\mathbf{V}_{ab} = \mathbf{V}_{ab} + 1$
7 | hoppa över $i + 1$
8 **else**
9 | $j = j + 1$
10 **end**
11 Ta $\mathbf{V}(a) = \mathbf{V}_a$ och $\mathbf{V}(b) = \mathbf{V}_b$ och beräkna $\mathbf{V}(a) - \mathbf{V}(b)$

Algoritm 5 tar bort multipliciteten från $p(x)$. Multipliciteten tas bort från $p(x)$ för att man skall kunna utnyttja att polynomet byter tecken innan och efter nollstället, vilket krävs för Sturmkedjemetoden. Exempelvis så har x^2 noll som nollställe och multiplicitet två och byter inte tecken i intervallet $(-1, 1)$. Däremot har x samma nollställe som x^2 och byter tecken i $(-1, 1)$.

Algoritm 5: Hantering av multipla nollställen

1 MultRoot.m ;
  Input : Polynom $\mathbf{p}(x)$
  Output: Polynom $\mathbf{p1}(x)$ med endast enkla rötter
2 Hitta sgd mellan $\mathbf{p}$ och $\mathbf{p}'$ (derivatan), $SGD = sgd(\mathbf{p}, \mathbf{p}')$
3 **if** Antal element av SGD är mindre än två **then**
4 | returnera $\mathbf{p1} = \mathbf{p}$
5 **else**
6 | Beräkna och returnera $\mathbf{p1} = \mathbf{p}/\mathbf{p}'$
7 **end**

Efter att Sturmkedjan och variationen har beräknats för ett polynom p(x) i intervallet (a,b) kan varje reellt nollställe i (a,b) isoleras. Nedan ges algoritm 6 som isolerar de reella nollställena för p(x).

Algoritm 6: Isolering av nollställen [Kal18]

```
1  isol.m ;
   Input  : Polynom p(x), mätintervall (a, b)
   Output: E, A
2  E = [ ], A = [ ];
3  r = Antal nollställen inom (a, b);
4  if r = 0 then
5  |  returnera E = [ ], A = [ ]
6  end
7  if r = 1 then
8  |  returnera E = [ ], A = (a, b)
9  end
10 W = [a, b, r];
11 while W ≠ [ ] do
12 |  Ta bort översta elementet [c, d, r] i W
13 |  m = (c + d)/2
14 |  if p(m) = 0 then
15 |  |  Placera m i E
16 |  |  p(x)/(x − m)
17 |  end
18 |  r = Antal nollställen inom(c, m)
19 |  if r = 1 then
20 |  |  placera (c, m) i A
21 |  else
22 |  |  placera (c, m, r) i W för vidare arbetning
23 |  end
24 |  r = Antal nollställen inom(m, d)
25 |  if r = 1 then
26 |  |  placera (m, d) i A
27 |  else
28 |  |  placera (m, d, r) i W för vidare arbetning
29 |  end
30 end
```

När varje reellt nollställe till $p(x)$ har isolerats i ett intervall så kan algoritm 7 användas för att minska intervallen.

Algoritm 7: Approximering inom tolerans [Kal18]

```
1  approx.m ;
   Input  : Polynom p(x), mätintervall (a, b), tolerans ϵ
   Output: A
2  if q(a) = 0 then
3  |  q(x)/(x − a)
4  end
5  if q(b) = 0 then
6  |  q(x)/(x − b)
7  end
8  c = a;
9  d = b;
10 m = (a + b)/2;
```

```
11 while d − c > ϵ do
12 |  if q(m) = 0 then
13 |  |  spara m
14 |  end
15 |  if sgn(q(c)) ≠ sgn(q(d)) then
16 |  |  d = m
17 |  |  m = (c + d)/2
18 |  else
19 |  |  c = m;
20 |  |  m = (c + d)/2
21 |  end
22 end
23 A = (c, d)
```

Algoritm 8 tar ett polynom och ger som utvärde rötternas multiplicitet samt de reella lösningar till polynomet som hittas med hjälp av Sturmkedjemetoden.

Algoritm 8: Hantering av indata

```
1  PolyLos.m ;
   Input  : Polynom p(x)
   Output: 2×(antal element p(x))-matris med nollställen i kolumn 1 och multiplicitet i
           kolumn 2
2  [R, p1] = MultRoot(p(x)), Ta bort multipla nollställen, R som är det största intervall som
     innehåller alla nollställen fås av sats 2.6
3  [P, I] = approx(p1, −R, R, Tollerans)
4  Beräkning av multiplicitet ;
5  Utvidga de exakta punkterna till små intervall runt dessa punkter
6  [P2, I2] = [P, I],  p2 = p
7  while Antal element i P2 eller I2 är större än noll do
8  |  Öka multipliciteten med 1
9  |  p2 = p2′
10 |  Beräkna [P2, I2] = approx(p2, vänstra intervallgräns, höger intervallgräns, Tolerans)
11 end
```

4.2 Algoritmer för argumentprincipmetoden

Algoritmen för att hitta lösningar till komplexa polynomekvationer innefattar dels själva argumentprincipen som hittar antalet nollställen inom en kvadrat i det komplexa planet, en algoritm för att söka fram approximerade nollställen i kvadraten samt en algoritm som given ett polynom löser polynomekvationen och bestämmer varje rots multiplicitet.

För att beräkna kurvintegralen längs en kvadrat delas den upp i fyra delar. Kurvan av varje sida kan då beskrivas som en linjär funktion och integralerna blir lätta att beräkna. Integralerna summeras och antalet nollställen i kvadraten har då hittats.

Algoritm 9: Argumentprincipen

```
1  ArgP.m ;
   Input  : Funktion f(x) = p′/p logaritmiska derivatan av polynomet p, Kvadrat i ℂ
   Output: Antalet nollställen r i den givna kvadraten
```

```
2  for Varje sida av kvadraten do
3  |  I = Kurvintegralen av f längs med kvadratens sida
4  |  if Integralen odefinierad then
5  |  |  return r = odefinierad
6  |  end
7  end
8  r = Summan av integralerna över kvadratens fyra sidor
```

Då en algoritm för argumentprincipen har implementeras kan själva sökningen av nollställen utföras. Algoritm 10 söker efter alla nollställen i en given kvadrat genom att rekursivt hitta alla nollställen i mindre och mindre kvadrater placerade inom den ursprungliga kvadraten.

Algoritm 10: Sök lösningar till en polynomekvation i en kvadrat

```
1  ImgIsol.m ;
   Input  : Funktion f = p'/p logaritmiska derivatan av polynomet p, Kvadrat K i ℂ, Matris
            för alla funna nollställen A
   Output: Alla unika nollställen till polynomet i K
2  r = Antalet nollställen i K
3  while r odefinierad do
4  |  K = K + (1 + i)ϵ, ϵ liten positiv konstant
5  |  r = Antalet nollställen i nya K
6  end
7  if r = 0 then
8  |  return
9  end
10 if r = 1 then
11 |  Lägg till nollstället i A
12 else
13 |  for Alla kvadranter i K do
14 |  |  A = Alla unika nollställen i kvadranten
15 |  end
16 end
```

Slutligen implementeras en algoritm som givet ett polynom ger alla komplexa lösningar samt varje lösnings multiplicitet.

Algoritm 11: Lös komplex polynomekvation med argumentprincipen

```
1  ArgPSolve.m ;
   Input  : Polynom p(x)
   Output: Alla nollställen till polynomet och deras multiplicitet
2  [R, p_m] = MultRoot(p(x)), Ta bort multipla nollställen, R radie för största cirkel som
     innehåller alla nollställen enligt 2.6
3  A = Alla unika nollställen till p_m, multiplicitet = 1
4  p_d = p
5  while Antalet funna nollställen < graden av p do
6  |  p_d = p_d'
7  |  for Varje nollställe α till p do
8  |  |  if 0 < Antalet nollställen till p_d' i en liten kvadrat runt α then
9  |  |  |  Öka multipliciteten för α i A med 1
10 |  |  end
11 |  end
12 end
```

4.3 Algoritm för kompanjonmatrismetoden

Algoritm 12 ger en algoritm för kompanjonmatrismetoden som börjar med att sätta upp kompanjonmatrisen för polynomet och sedan hittar egenvärdena. Vi har använt matlabs kommando *eig* för att beräkna egenvärdena som baseras på QR-algoritmen [SX03].

Algoritm 12: Konstruction av kompanjonmatrisen

```
1  KomMatMet.m ;
   Input  : Polynom p(x)
   Output: Alla nollställen till polynomet
2  k = p(1) högstagradskoefficient
3  for j = 1, ..., length(p) do
4  |  p(j) = p(j)/k Delar varje koefficient i p med högstagradskoefficient
5  end
6  p(1) = [ ] ta bort högstagradskoefficient i p
7  l=lengt(p)
8  z = (zeros(l, l − 1))
9  p=transpose(p)
10 A = [z, −p]
11 for i = 1, ..., l − 1 do
12 |  A(i + 1, i) = 1 Ettor lägns sub-diagonalen
13 end
14 Beräkna egenvärden för A
```

4.4 Tillämpning av algoritmer i matlab

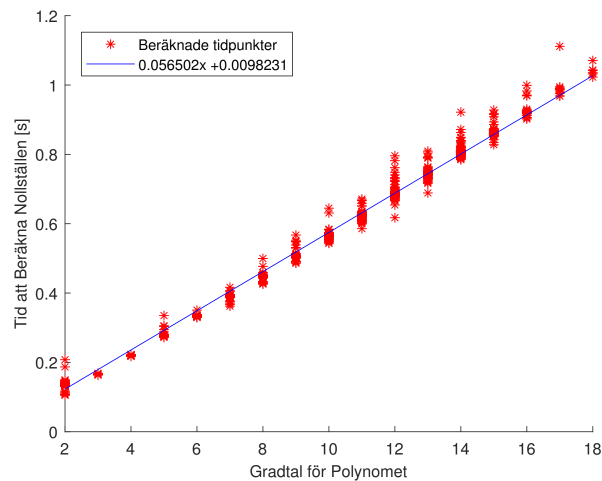
För att testa metoderna skrevs ett program i Matlab som slumpade fram polynomets nollställen och gradtal, som låg inom ett bestämt intervall, och utvecklade dessa nollställen till ett polynom. Efter detta beräknades nollställena med hjälp av antingen Sturmkedjemetoden, argumentprincipmetoden eller kompanjonmatrismetoden. Nollställena samt multipliciteten jämfördes därefter mot de nollställen som vi visste att polynomet hade och var inom en tolerans på cirka 10^{-3}. Beräkningtiden och gradtalet lades sedan i en lista som vi plottade. Notera att koden inte är fullt optimerad så att tiderna för alla metoder kan förbättras.

5 Resultat

Våra resultat avseende beräkning av ett polynoms nollställen med olika metoder presenteras nedan i tabeller eller graf som illustrerar beräkningstid versus polynomets gradtal. Graferna visar att det går att beräkna nollställena med en relativt enkel kod och att beräkningen inte tar alltför lång tid. Formen på graferna beror på hur vår kod ser ut och det är möjligt att en annorlunda implementering av metoderna har ett annat tid-grad utseende.

5.1 Sturmkedjemetoden

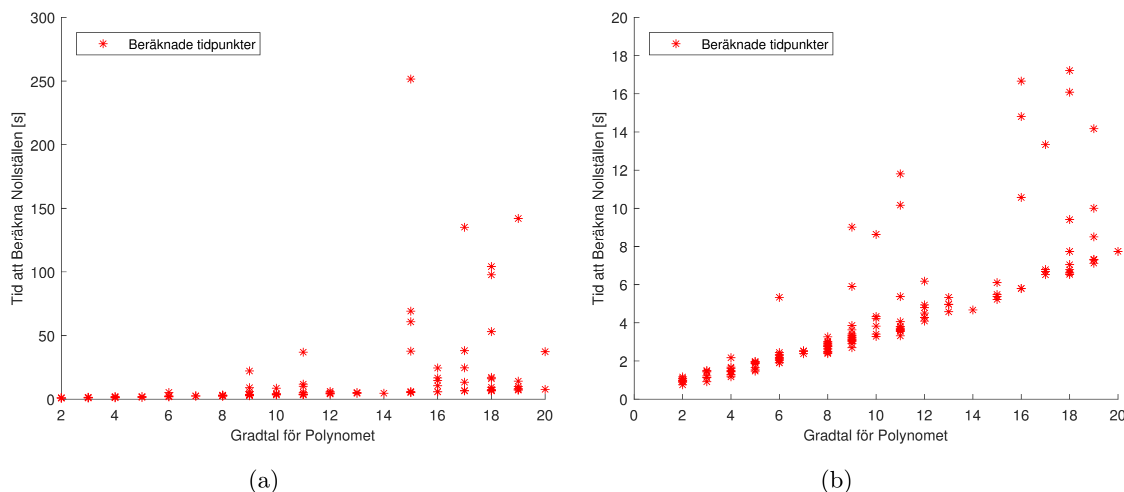
Figur 3 visar sambandet mellan tiden det tar för Sturmkedjemetoden att beräkna alla reella nollställen mot polynomets gradtal. Figuren pekar på ett relativt linjärt samband mellan tiden det tar att beräkna nollställena och gradtalet för polynomet. Figuren är baserad på cirka 1000 slumpade polynom med nollställen mellan -50 och 50 och gradtal mellan 2 och 18.



Figur 3: *Sambandet mellan tiden att beräkna nollställen med Sturmkedjemetoden och gradtal för polynomet.*

5.2 Argumentprincipmetoden

Figur 4 visar relationen mellan polynomets gradtal och tiden för att beräkna dess nollställen med argumentprincipen. Exekveringstiden för slumpmässigt utvalda nollställen, inom området $\alpha+i\beta$ där $\alpha, \beta \in [-1000, 1000]$, för 95 polynom av grad $1-11$ och 60 polynom med grad $11-20$ införs i Figur 4. Figur 4a visar att det förekommer ”outliers” med lång beräkningstid för nollställen när gradtalet för polynomet överstiger 14. I Figur 4b har vi dragit ned skalan på tidsaxeln och utelämnat en del ”outliers”, varvid ett svagt kurvlinjärt samband mellan beräkningstid och polynomets gradtal framträder.



Figur 4: *Sambandet mellan beräkningstid för nollställen ($\alpha + i\beta$ där $\alpha, \beta \in [-1000, 1000]$) enligt argumentprincipen och polynomets gradtal. Figur 4b visar en förstorad bild av Figur 4a och utelämnar en del ”outliers” ($> 20\,$s).*

I tabell 2 visas den genomsnittliga exekveringstiden för polynom av olika grader och storlek på dess nollställen.

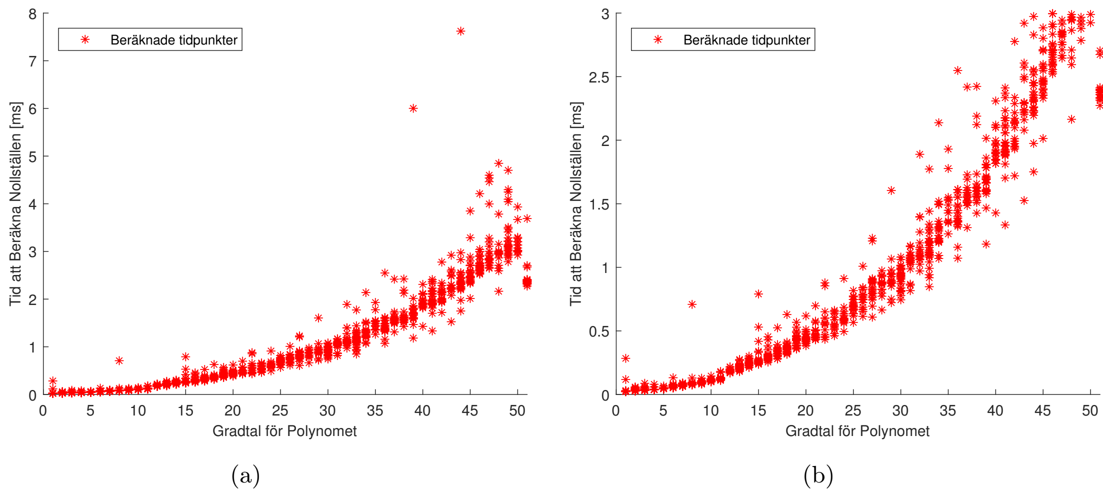
Tabell 2: *Exekveringstid för att hitta nollställen till polynom med komplexa koefficienter beroende på polynomets gradtal n och storleken av dess nollställen, mätvärden i [s]. För polynom av grad 20 väljs sidan till kvadraten innehållande alla nollställen till $R = 10^{13}$. Nollställen inom $\alpha + i\beta$ där α, β* anges i tabellen.

n	$\alpha, \beta < 10$	50	100	200	500	1000
2	2.2205	2.2531	2.2576	2.2694	2.2834	2.3116
3	2.3292	2.4496	2.5041	2.5349	2.6009	2.6542
5	2.7238	2.9775	3.1423	3.3247	3.5817	3.8910
10	4.1801	7.4823	9.6679	11.1912	14.1127	17.4760
20*	10.37779	12.14062	17.67183	32.73250	59.48764	103.0977

5.3 Kompanjonmatrismetoden

Figur 5 visar exekveringstiden med kompanjonmatrismetoden för polynom av grad 1 till 50 med slumpade nollställen inom området α+iβ där $\alpha, \beta \in [-1000, 1000]$. I Figur 5b där vi har dragit ned skalan på tidsaxeln och utelämnat beräkningstid $> 3\,$ms förefaller exekveringstiden öka exponentiellt när gradtalet på polynomen ökar.

I Tabell 3 visas den genomsnittliga exekveringstiden för Matlabs polynomlösare *roots* att hitta nollställen för polynom av olika grader och med nollställen av olika storlek.



(a) (b)

Figur 5: *Plot av tiden det tar att beräkna polynomets nollställen med kompanjonmatrismetoden gentemot dess gradtal. Polynomen är slumpade med gradtal mellan 1 och 50 samt nollställen inom området* $(\alpha + i\beta) \times 1000$ *där* $\alpha, \beta \in [-1, 1]$. *Notera att skalan på tidsaxeln är i millisekunder. Figur 5b visar en förstorad bild av figur 5a och utelämnar beräkningstid som överstiger* $3ms$.

Tabell 3: *Exekveringstid för Matlabs polynomlösare roots att hitta nollställen till polynom med komplexa koefficienter beroende på polynomets gradtal* n *och storleken av dess nollställen, mätvärden i [s]. Nollställen inom* $\alpha + i\beta$ *där* α, β *anges i tabellen.*

n	$\alpha, \beta < 10$	50	100	200	500	1000
2	0.000171	0.000191	0.000217	0.000233	0.000221	0.003374
3	0.000150	0.000159	0.000156	0.000170	0.000202	0.000484
5	0.000173	0.000156	0.000167	0.000190	0.000200	0.000638
10	0.000223	0.000217	0.000212	0.000243	0.000249	0.000290
20	0.000392	0.000389	0.000388	0.000288	0.000420	0.000481
50	0.002326	0.002642	0.002616	0.002822	0.002170	0.002708

Vi jämförde även kompanjonmatrismetoden för polynom med endast enkla nollställen med polynom som har nollställen med multiplicitet enligt nedan.

Exempel 5.1. *Antag att*

$$\begin{aligned} p_1(x) &= (x-2.2)(x-2.7)(x-3.3)(x-3.5)(x+3.7)(x-4.1) \\ &\quad (x+5.2)(x-8.4)(x+7.1)(x+5.2) \\ &= x^{10} - 3x^9 - 115.911x^8 + 329.738x^7 + 4342.45x^6 - 14281.2x^5 + 61770.2x^4 \\ &\quad + 261680x^3 + 171509x^2 - 1646320x - 1678410 \end{aligned} \tag{12}$$

och

$$\begin{aligned} p_2(x) &= (x-2.2)^3(x+3.5)^3(x-4.1)^4 \\ &= x^{10} - 13.2x^9 - 32.27x^8 + 1077.09x^7 - 2907.08x^6 - 20876.6x^5 + 118194x^4 \\ &\quad - 44077x^3 - 907691x^2 + 2216130x - 1613860 \end{aligned} \tag{13}$$

Nedanför i Tabell 4 visas beräknade resultat av *komMatMet.m*

Tabell 4: *Beräknade resultat av komMatMet.mat*

(a) *Enkla nollställen*

p_1 med enkla nollställen	
Nollställen	Beräknade resultat
2.2	2.2000
2.7	2.7000
3.3	3.3000
3.5	3.5000
4.1	4.1000
5.2	5.2000
8.4	8.4000
-3.7	-3.7000
-5.2	-5.2000
-7.1	-7.1000

(b) *Nollställen med multiplicitet*

p_2 med multiplicitet	
Nollställen	Beräknade resultat
2.2	$2.2000 + 0.0001i$
2.2	$2.2000 - 0.0001i$
2.2	$2.2001 + 0.0000i$
4.1	$4.0989 + 0.0011i$
4.1	$4.0989 - 0.0011i$
4.1	$4.1011 + 0.0011i$
4.1	$4.1011 - 0.0011i$
-3.5	$-3.5000 + 0.0000i$
-3.5	$-3.5000 + 0.0000i$
-3.5	$-3.5000 + 0.0000i$

Tabell 5 visar beräknade resultat med Matlabs polynomlösare *roots*.

Tabell 5: *Beräknade resultat av roots*

(a) *Enkla nollställen*

p_1 med enkla nollställen	
Nollställen	Beräknade resultat
2.2	2.2000
2.7	2.7000
3.3	3.3000
3.5	3.5000
4.1	4.1000
5.2	5.2000
8.4	8.4000
-3.7	-3.7000
-5.2	-5.2000
-7.1	-7.1000

(b) *Nollställen med multiplicitet*

p_2 med multiplicitet	
Nollställen	Beräknade resultat
2.2	$2.2000 + 0.0000i$
2.2	$2.2000 - 0.0000i$
2.2	$2.2000 + 0.0000i$
4.1	$4.1002 + 0.0002i$
4.1	$4.1002 - 0.0002i$
4.1	$4.0998 + 0.0002i$
4.1	$4.0998 - 0.0002i$
-3.5	$-3.5000 + 0.0000i$
-3.5	$-3.5000 + 0.0000i$
-3.5	$-3.5000 + 0.0000i$

6 Diskussion

Att bestämma nollställen till polynomekvationer är av stor betydelse inom flera matematiska tillämpningsområden. Ett viktigt problem är att det finns inga matematiska metoder för att hitta en exakt lösning till polynomekvationer av grad större än fyra utom i vissa undantagsfall. Däremot finns det flera numeriska metoder för att approximera nollställena till polynom av grad fem eller högre. Vi har i detta arbete undersökt och implementerat tre metoder för att approximativt lösa polynomekvationer utgående från Sturmkedjor, argumentprincipen samt kompanjonmatrisen. Vi har även studerat ett antal underfrågor kring polynomets nollställen (avsnitt 1.2) och avhandlat de med teoretiska resonemang (kapitel 3).

6.1 Sturmkedjemetoden

Sturmkedjemetoden kan endast lösa reella polynomekvationer med reella koefficienter, men i många fall är vi endast intresserade av dessa lösningar. Sturmkedjemetoden är således användbar i många situationer, men uppvisar ändå nackdelar vid viss implementering. Bland annat är den dålig på att hantera godtyckliga polynom med godtyckliga reella koefficienter. Problemet är att det är svårt att exakt polynomdividera och hitta största gemensamma delare (sgd) för ett allmänt polynom med koefficienter med decimaltecken. Svårigheten att hitta sgd gör att polynomets multiplicitet inte kan divideras bort. Felaktigheterna i polynomdivisionen gör att konstruktionen av Sturmkedjan inte är tillämpbar i dessa fall då den ger fel värden. Teoretiskt kan alla reella lösningar till ett polynom hittas med hjälp av Sturmkedjemetoden men på grund av datorns begränsade beräkningsintervall kommer inte extremt stora nollställen att kunna hittas. Fördelar med Sturmkedjemetoden är att den endast hittar reella lösningar, den är relativt snabb och den kan, för låga gradtal, enkelt utföras för hand.

Vid beräkning av ett slumpat polynoms nollställen med vår egna funktion *PolyLos.m* satte vi det maximala gradtalet till 18 (Figur 3). Detta för att vid gradtal större en 18 och vid nollställen med stort absolutbelopp så blev det fel i de deriveringar och polynomdivisioner som Matlab behöver utföra. Om detta fel sedan används i nästa polynomdivision blir felet större och därmed kommer metoden att ge fel värden på nollställena. Sturmkedjemetoden är därmed mest användbar för låga gradtal, under cirka 15, där det tar mindre än en sekund att beräkna varje polynoms nollställen. Tiden det tar att beräkna nollställena växer linjärt som $t = 0.056502x + 0.0098231$ beroende på gradtalet x (Figur 3).

6.2 Argumentprincipmetoden

Metoden som baseras på argumentprinicipen kan lösa komplexa polynomekvationer, vilket inte Sturmekedjemetoden förmår att göra. En fördel med argumentprincipen är dess relativt enkla grafiska representation. Teoretiskt sett kan alla nollställen till ett polynom som ligger inom ett väldefinierat godtyckligt område hittas med hjälp av argumentprincipen. Polynom av grad fem eller lägre hanteras väl, även för nollställen $0 < |z_k| < 1000$ (Tabell 2). Exekveringstiden för polynom av högre grad ökade kraftigt, varvid algoritmen blev opraktisk. Till exempel blir inte bara integralerna längs en kurva som passerar ett nollställe odefinierade, utan även kurvor som passerar nära ett nollställe resulterar i odefinierade integraler vid beräkning i Matlab. Detta försvårar sökandet då små kvadrater runt möjliga nollställen uppträder väldigt opålitligt.

Ett annat problem som uppstår vid implementeringen av argumentprincipmetoden är att för stora polynom kommer alla beräkningar att snabbt innefatta enormt stora tal. För ett polynom av grad 20 med nollställen inom $|z| < 10$ blir integralen av den första kvadraten given av sats 3.4 så stor att Matlab tolkar den som oändlig. Om en mindre kvadrat som innehåller alla nollställen är känd kan istället den nyttjas i programmet. För att lösa polynomekvationer av grad 20 användes en kvadrat med sidan $r = 10^{13}$, vilket visade sig vara den störst möjliga kvadraten som kunde användas för att hitta alla nollställen. På liknande sätt kan kvadraten väljas allt mindre för att hantera polynom av allt högre grad. För polynom av tillräckligt höga gradtal kommer den minsta kvadraten centrerad i origo som ger en definierad integral inte innehålla alla nollställen och programmet kan då anses

ha slutat fungera. Detta sker för polynom med gradtal kring 95 eller högre.

Lösningar av polynomekvationer med hjälp av argumentprincipen kan sägas fungera i allmänhet väl för ekvationer med gradtal < 20, under speciella förhållanden för gradtal från 20 till 95 och inte alls för gradtal > 95. Tekniskt sett kan alltså polynomekvationer av grad upp till 95 lösas. Exekveringstiden ökar dock snabbt i takt med gradtalet och uppnår flera minuter för polynom av grad 50 eller högre. Implementeringen av argumentprincipen kan alltså ses som ett ineffektivt sätt att lösa polynomekavtioner med högt gradtal.

6.3 Kompanjonmatrismetoden

Kompanjonmatrismetoden kan i likhet med argumentprincipen lösa komplexa polynomekvationer. Eftersom det finns flera effektiva numeriska algoritmer för att beräkna egenvärdena för en matris såsom QR-algoritmen kan nollställen till ett polynom beräknas snabbt och precist. Vår tillämpning av kompanjonmatrismetoden visade att den var snabb och att den klarade av polynom upp till grad 50 utan problem (Figur 5).

Det finns tydliga skillnader mellan polynom med enkla nollställen och polynom med multiplicitet som vi måste ta hänsyn till vid beräkningar (exempel 5.1). För polynom med enkla nollställen ger kompanjonmatrismetoden snabba och noggranna resultat (Tabell 4a), medan för polynom med multiplicitet ger beräkningarna i regel inte lika noggrant utbyte (Tabell 4b och 5b). Xiao-Ming Niu och Tetsuya Sakurai föreslår i en artikel från 2003 [SX03] att detta kan lösas genom att konstruera en ny kompanjonmatris med endast enkla egenvärden. Med deras metod kan man, istället för att beräkna alla nollställen samtidigt, beräkna polynomets distinkta nollsällen och multiplicitet separat. På detta vis tar man bort multipliciteten från polynomet, vilket resulterar i ett polynom med samma nollställen fast alla är enkla. På så sätt används komapanjonmatrismetoden för att i första led beräkna de distinkta nollställena; multipliciteten beräknas sedan i efterhand för att få mer exakta resultat.

6.4 Jämförelse av metoder och fördjupning i underfrågor

Vid beräkning av nollställen är idén bakom Sturmkedjemetoden och argumentprincipmetoden mycket lik. Båda metoder börjar med att hitta antalet nollställen till polynomet. Sturmkedjemetoden delar sedan in nollställena i intervall tills varje intervall endast innehåller ett nollställe och minskar varje intervall inom en viss tolerans. Argumentprincipen delar in nollställena i kvadrater istället för intervall och när varje nollställe är indelat i en egen kvadrat minskas kvadraten. Båda metoderna räknar systematiskt ut ett nollställe i taget. På grund av den extra dimension som argumentprincipmetoden tillämpar behöver den genomföra betydligt fler iterationer för att hitta alla nollställen. Kompanjonmatrismetoden räknar ut alla nollställen samtidigt. Sturmkedjemetoden har inga ”outliers” som drastiskt påverkar beräkningstiden som argumentvariationspricipen eller kompanjonmatrismetoden har, jämför Figur 3 med Figur 4 och Figur 5. Detta innebär att Sturmkedjemetoden är pålitlig och användbar i det intervall den kan beräkna.

I avsnitt 1.2 presenterades en rad frågeställningar angående särskilda aspekter för lösningar av polynomekvationer. Dessa har alla besvarats i rapporten och kortfattade svar till varje fråga återges i detta avsnitt. I föregående avsnitt har vi visat att det finns tre ganska effektiva metoder för att hitta alla lösningar till en polynomekvation $f(z) = 0$. Alla dessa metoder har fördelar och nackdelar. Är beräkningshastighet av intresse är kompanjonmatrismetoden den mest användbara men den har nackdelen att den kan göra om en reell rot till en komplex. Om endast reella lösningar efterfrågas kan man använda Sturmkedjor. Förutom att hitta hur många av nollställena som kommer att vara reella kan man med hjälp av sats 2.2.1 även se hur många som kommer att vara komplexa. Då Sturmkedjor kan hitta alla reella lösningar har metoden inget problem att beräkna hur många nollställen som kommer att finnas i ett definierat intervall $[a, b]$.

Om ett nollställe har multiplicitet större än ett som inte kan tas bort är Sturmkedjemetoden inte användbar. Även kompanjonmatrismetoden kan ge komplexa värden för reella rötter vilket inte

är helt optimalt. Argumentprincipen är inte lika beroende av rötternas multiplicitet och metoden hittar distinkta nollställena för ett allmänt polynom. Däremot har vår algoritm svårt att hitta nollställenas multiplicitet och behöver därför optimeras.Trots en lång beräkningstid kan argumentprincipmetoden användas med fördel för att hitta distinkta nollställen.

Vi frågade även vilken cirkel $|z| \leq R$ i det komplexa planet kommer med all säkerhet innehålla alla nollställen till $f(z)$? Enligt Sats 2.6 innehåller cirkeln i det komplexa planet med den största radien R med all säkerhet alla nollställen till $f(z)$.

6.5 Konklusion

De tre metoder som vi har utforskat för lösning av polynomekvationer av grad 5 eller högre har alla fördelar och nackdelar. Bland Sturmkedjemetoden, argumentprincipmetoden och kompanjonmatrismetoden kan vi inte utse en överlag bästa metod för att hitta nollställen till alla former av polynomekvationer. Vilken metod som passar bäst beror helt och hållet på vilket problem som ska lösas. Ska man beräkna många polynom och det inte spelar någon roll att reella lösningar kan få en liten imaginärdel kan kompajonmatrismetoden vara intressant. Om endast reella lösningar efterfrågas och polynomen har heltalskoefficienter är Sturmkedjemetoden mycket användbar. Argumentprincipmetoden fungerar bra för de flesta rötter men när polynomets gradtal stiger så behöver den betydligt längre beräkningstid än de andra metoderna.

Referenser

[Bec+02] Matthias Beck m. fl. *A First Course in Complex Analysis*. 1.53. Orthogonal Publishing, 2002, s. 23, 135–137. ISBN: 978-1944325-04-6.

[Hea02] Michael T. Heath. *Scientific Computing An Intoductory Survey*. Second. Illinois, USA: McGraw-Hill, 2002, s. 179–180. ISBN: 978-0-07-112229-0.

[Kal18] K. Kalorkoti. *Introduction to Computer Algebra*. First. 10 Crichton Street Edinburgh EH8 9AB, Scotland: School of Informatics University of Edinburgh, January 2018, s. 90–99.

[PB10] Arne Persson och Lars-Christer Böiers. *Analys i en variabel*. swe. Studentlitteratur AB, 2010. ISBN: 978-91-44-06765-0.

[SX03] Tetsuyaand Sakurai och Niu Xiao-Ming. "A Method for Finding the Zeros of Polynomials Using a Companion Matrix". I: *Japan Journal of Industrial and Applied Mathematics* 20.1 (juni 2003), s. 239–256.

[VE11] Anders Vretblad och Kerstin Eksting. *Algebra och geometri*. Andra. Uppsala, Sverige: Gleerup Utbildning AB, 2011. ISBN: 978-91-4064757-3.

[Wil62] H. S. Wilf. *Mathematics for the Physical Sciences*. First. New York, NY, USA: Dover Publications, Inc., 1962. ISBN: 0486635366.

A Appendix

A.1 Matlab kod

A.1.1 Kod för utförande av Sturmkedjormetoden

```
function u = sturm(p)
tol = 1e-6;
pNext = polyder(p); %hittar f2 = f'
 Z = [p; 0, pNext]; %lägger in f1 och f2 som vektorer i början av en matris
 %loopen körs så länge det sist tillagda polynomet i matrisen har grad > 0
 villkor = 1;   %sant
 u = [];
 while villkor
      if 0 >= nnz(pNext(1: end-1))
          break
      end
     [q, r] = deconv(p, pNext(find(pNext ~= 0, 1, 'first') : end));
      if  norm(r) < tol
           t = Z(end , :);
           t = t(find(t ~= 0, 1, 'first') : end);
           [p1, ~] = deconv(Z(1, :), t);   %Lägg till den första sturmkedjan
           u = p1;
           Langd = length(Z(:, 1));
           for j = 2:Langd
               [q, ~] = deconv(Z(j, :), t);
               pNext= [zeros(1, length(p1) - length(q)), q];
               u = [u; pNext];
           end
           villkor = 0;
      else
          p = pNext; %flyttar ett steg ned i kedjan
          %nästa polynom i kedjan är -resten vid tidigare polynomdivision
          pNext= [zeros(1, length(p) - length(r)), -r];
          Z = [Z; pNext]; %lägg till polynomet i matrisen
      end
 end
 if isempty(u)     %inte tom
     u = Z;
 end
end
```

```
function var = variation(a, b, u)
var = 0; sa = zeros(size(u, 1),1); sb = zeros(size(u, 1),1);
for i = 1:size(u, 1) %kör för alla polynom i sturmkedjan
     sa(i) = polyval(u(i, :), a); %finner värdet till alla polynom i pkt a
     sb(i) = polyval(u(i, :), b);
end
sa = sign(sa);
sb = sign(sb);
for i = 1:size(sa, 1)-1 %kör igenom och jämför alla värden i sturmkedjan
     if sa(i)*sa(i+1) < 0      %om f_k och f_k+1 har olika tecken i pkt a
          var = var+1; %variationen ökar då var = V(a)-V(b)
     elseif sa(i)*sa(i+1) == 0
          var = var+1;
          i = i+1;
     end
end

for i = 1:size(sa, 1)-1 %kör igenom och jämför alla värden i sturmkedjan
     if sb(i)*sb(i+1) < 0      %om f_k och f_k+1 har olika tecken i pkt a
          var = var-1; %variationen ökar då var = V(a)-V(b)
     elseif sb(i)*sb(i+1) == 0
          var = var-1;
          i = i+1;
     end
end
```

```
function [MaxRadie, SingleRoot] = MultRoot(p)
%Tar bort multipliciteten från p och beräknar maxvärdet för rötter
```

```

pPrim = polyder(p);
Gcd = GCD(p, pPrim);

SingleRoot = deconv(p, Gcd);

MaxRadie = 1 + max(abs(p))/abs(p(1));
end
```

```
function GcdKoeff = GCD(p1, p2)
%Beräknar koefficienterna hos gcd för två polynom av formen [c_1, ..., c_n]
%Ger tillbaka gcd-polynomets koefficienter på formen [c_1, c_2, ... c_n]

syms x; %säg att x ska vara variabeln
Polynom1 = poly2sym(p1, x); %Generera en funktion c_n*x^0 + c(n-1)*x^1 + ... (med x
    )
Polynom2 = poly2sym(p2, x);

GCDKoeff = coeffs(gcd(Polynom1, Polynom2), 'All'); %double: Konvertera från syms
    till num.

Lengd = length(GCDKoeff);
for i = 1:Lengd
    GcdKoeff(i) = double(GCDKoeff(i));
end

end
```

```
function Root = PolyLos(p)

Tolx =1e-6;

[R, p4] = MultRoot(p); %Alla rötter mellan -R,R och p1 är utan multiplicitet.
MaxVarde = (length(p)-1)*log10(R);

if MaxVarde > 250
   R = 10^((250-1) / (length(p)-1)); %Se till att vi inte får NaN
   MaxVarde = (length(p)-1)*log10(R);
end

if MaxVarde < 250
    [P, I] = approx(p4, -R, R, Tolx);
    if isempty(I) && isempty(P)   % Inga nollställen
        Root = [0, 0];
        disp('tom')
        return
    end

    AntalP = 0; AntalI = 0;

    if 0 < numel(P)
        AntalP = length(P(1, :));
    elseif 0 < numel(I)
        AntalI = length(I(:, 1));
    end

    Root = zeros(AntalI+AntalP, 2);
    epsilon = 1e-6;
    totAntal = AntalI + AntalP;

    for j = 1:totAntal
        Fortsatt = 1;
        n = 0;
        p1 = p;
        if AntalI > 0
            a = I(j, 1);
            b = I(j, 2);
            AntalI = AntalI - 1;
        elseif AntalP > 0
            a = P(j - AntalI) - epsilon;  %litet intervall runt exakt punkt
```

```
            b = P(j - AntalI) + epsilon;
        end
        while Fortsatt == 1
            [R, p2] = MultRoot(p1);
            if 1 < numel(p2)
                [T,J] = approx(p2, a , b, Tolx);
                if isempty(J) && isempty(T)
                    Root(j, 2) = n;
                   Root(j, 1) = (a+b)/2;
                   Fortsatt = 0;
                elseif isempty(J) && numel(p2) < 2
                    Root(j, 2) = n;
                    Root(j, 1) = (a+b)/2;
                    Fortsatt = 0;
                elseif ~isempty(T)    %om den inte är tom
                    Root(j, 2) = n;
                    Root(j, 1) = T;
                else
                    Root(j, 2) = n;
                    Root(j, 1) = (a+b)/2;
                end
            else
                Fortsatt = 0;
            end

            p1 = polyder(p1);
            n = n+1;
        end

    end
else
    disp('too large number')
end
end
```

```
function [P, I]= approx(p, a, b, e)

% P kolonnmatris med exakta nollställen, I  matris då r varje rad är ett
% intervall av storlek e och  innehåller endast ett  nollställe.

[E, A]=isol(p, a, b);

P = E;
I = [];
while 0 < numel(A) %Kör till den gått igenom alla intervall (rader) i A

    if polyval(p, A(1, 1)) == 0 % Kollar om första ändpunkten är nollställe
        p1 = [ 1 -A(1, 1)];      % Sätter p2 = '(x-nollställe)'
        p = deconv(p, p1);       % Polynomdivision p=p/p1
    end

    if polyval(p, A(1, 2)) == 0 % Kollar om andra ändpunkten är nollställe
        p2 = [ 1 -A(1, 2)];
        p = deconv(p, p2);
    end

    c = A(1, 1);
    d = A(1, 2);
    m = (c+d)/2;

    while d-c > e % Hittar nollstället eller minskar intervallet som innehåller det

        if polyval(p, m) == 0
            P(end+1, :)= m;
            return % Om m är nollställe avslutas loopen
        end

        % Kollar om nollstället är mellan c och m eller mellan m och d
        if sign(polyval(p, c)) ~= sign(polyval(p, m)) && sign(polyval(p, m)) ~= 0
            d = m;
            m = (c+d)/2;
```

```
37          else
38              c = m;
39              m = (c+d)/2;
40          end
41      end
42
43      I(end+1, :) = [c d]; % Placera [c d] som en radvektor i I
44      A(1, :) = [];         % Tar bort översta intervallet (raden) i A
45
46  end
47  end
```

```
1  function [E, A] = isol(p, a, b)
2
3  E = []; A = [];
4  u = sturm(p);
5  r = variation(a, b, u);
6
7  if r == 0
8      return           % Inga nollställen, returerar E = [] och A = [].
9  elseif r == 1
10     A = [a b];    % Ett nollställe, returerar E = [] och A = [(a, b)]
11 else
12     W = [a b]; % Matris med intervall som innehåller fler än ett nollställe.
13
14     while 0 < numel(W) % Kör tills alla nollställen är hittade eller
15                        % varje nollställe är isolerat i ett intervall.
16         a = W(1, 1);
17         b = W(1, 2);
18         m = (a+b)/2;
19         W(1, :) = [];    % Tar bort översta
20         if polyval(p, m) == 0  % kollar om m är ett nollställe
21             E(end+1, :) = m;    % placerar m i E
22         end
23
24         r = variation(a, m, u);  % kollar hur många nollställen det är mellan a och
       m
25         if r == 1
26             A(end+1, :) = [a m]; % om ett nolställe, placera (a,m) i A
27         elseif r > 1
28             W(end+1, :) = [a m]; % om fler än ett nollställe placera [a m r] i W
     för vidare arbetning
29         end
30
31         r = variation(m,b,u);  % kollar hur många nollställen det är mellan m och b
32         if r == 1
33             A(end+1, :) = [m b]; % om ett nollställe, pacera (m,b) i A
34         elseif r > 1
35             W(end+1, :) = [m b]; % om fler än ett nollställe placera [m b r] i W
     för vidare arbetning
36         end
37     end
38
39 end
40 end
```

A.1.2 Kod för kompanjonmatrismetoden

```
1  function A  = KomMatMet(p)
2
3  k=p(1)
4
5  for j = 1:length(p)
6
7       p(j) = p(j)/k
8
9  end
10
11 p(1) = [];            % tarbort 1:a elemetet
12 p = transpose(fliplr(p));
```

```
Langd = length(p);
z = zeros(Langd, Langd-1);
A = [z -p];

for i = 1:Langd-1
    A(i+1,i) = 1;
end

A = eig(A);

end
```

A.1.3 Kod för argumentprincipen

```
function r = ArgP(f, O, R)
warning ('off', 'all');
lastwarn('')
I1 = integral(@(x)f(x+1i*R+O), R, -R); %Översida, kurvan = t+i*R
% Om integralern är odefinerad ger matlab en varning och r sätts till NaN
if numel(lastwarn)>0
    r = NaN;
else
    I2= integral(@(x)f(-R+x*1i+O)*1i, R, -R); % Vänster sida, kurvan = -R+t*i
    if numel(lastwarn)>0
        r = NaN;
    else
        I3 = integral(@(x)f(x-1i*R+O), -R, R); % Undersida, kurvan = t-R*i
        if numel(lastwarn)>0
            r = NaN;
        else
            I4 = integral(@(x)f(R+1i*x+O)*1i, -R, R); % Höger sida, kurvan = R+t*i
            if numel(lastwarn)>0
                r = NaN;
            else
                r = double((I1+I2+I3+I4)/(2*pi*1i));
                return
            end
        end
    end
end
end
```

```
function A = ArgPSolve(p)
[R, pm] = MultRoot(p); % Tar bort nollställen med multiplicitet
syms x
f = matlabFunction(poly2sym(polyder(pm), x)/poly2sym(pm, x)); %f = p'/p
A = ImgIsol(f, 0, R, [], 0); % Hittar lösningar i kvadrat med sida R centrerad i O
pd = p;
% Kör då antalet hittade nollställen är färre än polynomets grad
while sum(A(:, 2)) < length(p)-1
    pd = polyder(pd);
    g = matlabFunction(poly2sym(polyder(pd), x)/poly2sym(pd, x));
    for i = 1:length(A)
        % Om nollställe för p är nollställe för p' har det multiplicitet
        if round(ArgP(g, A(i, 1), 10e-6)) ~= 0
            A(i, 2) = A(i, 2)+1; % Öka nollställets multiplicitet med 1
        end
    end
end
end
```

```
function A = ImgIsol(f, O, R, A, r)
AF = A; rf = r; tol = 10e-6; n = 0; O1 = O;
r = ArgP(f, O, R); % Antalet nollställen i kvadrat med sida R centrerad i O
while isnan(r) == 1 % Nollställe på randen, flytta kvadrat och försök igen
    O = O+max(tol*0.1, tol*(R^0.5))*(1+1i); % Nytt O för att flytta kvadraten
    n = n+1;
    if n < 5 && R > tol
        r = ArgP(f, O, R);
    % För många iterationer utan att hitta r, sätt r till max möjligt
```

```
10      elseif R > 10e-3
11          r = rf;
12          O = O1;
13      else % Sidan så liten att r antas vara 1
14          r = 1;
15          O = O1;
16      end
17  end
18  if r < tol % Inga nollställen
19          return
20  elseif  round(real(r)) == 1 &&  R < tol %Ett nollställe, lägger till O i A
21          A = [A; O, 1];
22  else
23      A = ImgIsol(f, O+(1+1i)*R/2, R/2, A, r); % Kollar rekursivt en kvadrant till
24      % Om man hittat alla nollställen i gällande kvadraten skippas resten
25      if (numel(A)-numel(AF))/2 < round(r)
26          A = ImgIsol(f, O+(1-1i)*R/2, R/2, A, r);
27      end
28      if (numel(A)-numel(AF))/2 < round(r)
29          A = ImgIsol(f, O+(-1-1i)*R/2, R/2, A, r);
30      end
31
32      if (numel(A)-numel(AF))/2 < round(r)
33          A = ImgIsol(f, O+(-1+1i)*R/2, R/2, A, r);
34      end
35  end
36  end
```

A.1.4 Kod för test av funktioner

```
1  Antalggr = 7000;
2  Beee = zeros(Antalggr, 2);
3  villkor = 1;
4  Tol = 1E-3;
5  faila = 0;
6  for z = 1:Antalggr
7
8  j = round(rand*30)+1;   %+1 så att inte lika med noll
9  A = zeros(1, j);
10 for i = 1:j
11     A(i) = round(((rand*2 -1))*50)+1;
12 end
13
14 ii = 1;
15 pp = 2;
16 B = [];
17 T = 1;
18 while (ii <= j)
19     v = 1;
20     while (pp <= j)
21         if isempty(find(T == pp))
22             if A(ii) == A(pp)
23                 v = v+1;
24                 T = [T pp];
25             end
26         end
27         pp = pp+1;
28     end
29     B = [B; A(ii) v];
30     ii = ii+1;
31     pp = ii+1;
32 end
33
34 Er = length(B(:, 1));
35 ii = 1;
36 pp = 2;
37 while ii <= Er
38     while pp <= Er
39         if B(ii, 1) == B(pp,1)
40             B(pp, :) = [];
41             pp = ii;
```

```
42             Er = Er-1;
43         end
44         pp = pp+1;
45     end
46     ii = ii+1;
47     pp = ii+1;
48 end
49
50 p5 = poly(A);
51 Root = []; tid = [];
52 if length(B(1, :)) == length(A) %enkla nollställen
53     tic;
54     Root = PolyLos(p5);
55     tid = toc;
56 elseif numel(GCD(p5, polyder(p5))) > 1
57     tic;
58     Root = PolyLos(p5);
59     tid = toc;
60 else
61    faila = faila + 1;
62 end
63
64 if ~isempty(Root)
65     Root = sortrows(Root);
66     B = sortrows(B);
67     if length(B(:, 1)) == length(Root(:, 1))
68         BLangd = length(B(:, 1));
69         Skillnaden = B(:, 1) - Root(:, 1);
70         for pii = 1:BLangd
71             test = norm(Skillnaden(pii));
72             if test > Tol
73                 Villkor = 0;
74             end
75         end
76     else
77         villkor = 0;
78     end
79     if ~isempty(tid) && villkor
80         Beee(z, 1) = j;  %se till att rötter är rätt
81         Beee(z, 2) = tid;
82     end
83 end
84
85 if mod(z, 10) == 0
86 disp(z)
87 end
88
89 clearvars -except Beee z Antalggr Tol villkor faila
90 villkor = 1;
91 end
92
93 [x, y] = find(Beee == 0);
94 Beee(x, :) = [];
95
96 %%plotta upp mätvärden
97
98 hold on
99 minsta = min(Beee(:, 1));
100 storsta = max(Beee(:, 1));
101 set(gca, 'fontsize', 10)
102 p9 = polyfit(Beee(:, 1), Beee(:, 2), 1);   %linjär anpassning
103 plot1 = plot(Beee(:, 1), Beee(:, 2), 'r*')
104 xlabel('Gradtal för Polynomet','fontsize', 10)
105 ylabel('Tid att Beräkna Nollställen [s]','fontsize', 10)
106 plot2 = fplot(poly2sym(p9), [minsta, storsta], 'b')
107 legend('location', 'Northwest')
108 legend( [plot1 plot2], 'Beräknade tidpunkter', strcat( num2str(p9(1)), 'x + ',
    num2str(p9(2))))
```