# The computation of multiple roots of a polynomial using structure preserving matrix methods

by

Madina Hasan

A thesis submitted to the

Computer Science

in conformity with the requirements for

the degree of PhD

Sheffield University

England

July 2011

# Abstract

Solving polynomial equations is a fundamental problem in several engineering and science fields. This problem has been handled by several researchers and excellent algorithms have been proposed for solving this problem. The computation of the roots of ill-conditioned polynomials is, however, still drawing the attention of several researchers. In particular, a small round off error due to floating point arithmetic is sufficient to break up a multiple root of a polynomial into a cluster of simple closely spaced roots. The problem becomes more complicated if the neighbouring roots are closely spaced. This thesis develops a root finder to compute multiple roots of an inexact polynomial whose coefficients are corrupted by noise. The theoretical development of the developed root solver involves the use of structured matrix methods, optimising parameters using linear programming, and solving least squares equality and nonlinear least squares problems.

The developed root solver differs from the classical methods, because it first computes the multiplicities of the roots, after which the roots are computed. The experimental results show that the developed root solver gives very good results without the need for prior knowledge about the noise level imposed on the coefficients of the polynomial.

# Acknowledgment

First, thanks to God for endowing me with health and knowledge to complete this work.

I acknowledge, with deep gratitude and appreciation, the endless help, inspiration, time and guidance given to me by my supervisor, Dr. Joab Winkler.

A special word of gratitude to my husband, Hasan, and to my children Zahraa and Ali. Without them, this work couldn't have been achieved.

I am heartily thankful to my parents, sisters, brothers and friends for their continuous encouragement.

# Thesis contributions

1. Winkler, J. R. and Hasan M., A non-linear structure preserving matrix method for the low rank approximation of the Sylvester resultant matrix, *Journal of Computational and Applied Mathematics* 234 (2010) 3226-3242.

2. Winkler, J. R., and Hasan, M., An improved non-linear method for the computation of a structured low rank of the Sylvester resultant matrix, submitted to *Journal of Computational and Applied Mathematics.*

3. Winkler, J. R., Lao, X., and Hasan, M., The computation of a structured low rank of the Sylvester resultant matrix of two inexact polynomials by approximate polynomial factorisation, submitted to *Numerische Mathematik.*

4. Winkler, J. R., Hasan, M., and Lao, X., Two methods for the calculation of the degree of an approximate greatest common divisor of two inexact polynomials, submitted to *SIAM J. Matrix Analysis.*

5. Winkler, J. R., Lao, X., and Hasan, M., The computation of multiple roots of a polynomial, submitted to *Linear Algebra and Its Applications.*

# Abbreviations

| | | |
|---|---|---|
| APF | ... | approximate polynomial factorisation |
| AGCD | ... | approximate greatest common divisor |
| GCD | ... | greatest common divisor |
| deg $f(y)$ | ... | degree of the polynomial $f(y)$ |
| GM | ... | geometric mean |
| LS | ... | least squares |
| LSE | ... | least squares with equality |
| SNTLN | ... | structured non-linear total least norm |
| STLN | ... | structured total least norm |
| $S(f,g)$ | ... | Sylvester resultant matrix for the power basis polynomials $f(y)$ and $g(y)$ |
| $S_k(f,g)$ | ... | Sylvester subresultant matrix of order $k$ for the power basis polynomials $f(y)$ and $g(y)$ |
| $\|\cdot\|$ | ... | $\|\cdot\|_2$ |

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

Polynomials enjoy widespread use in several engineering and science fields, including control, coding theory, game theory, signal processing, computer graphics and many other applications. In triangle geometry, for example, polynomials are used to represent the relation between lengths and angles. Polynomials are also used in computer aided geometric design and geometric modeling for curve and surface representations. In more complicated applications such as robotics, polynomials are used to relate forces, trajectories and moments in order to control the robotic movements. In these applications, it is usually of interest to find the values at which a polynomial or a system of polynomials vanishes to indicate the occurrence of certain events such as the intersection of curves and surfaces. Such values are referred to as the zeros of the polynomials and the task of computing these zeros is called the root finding task. Many problems are reduced to the problem of root finding, such as the problem of shape interrogation in computer aided geometric design [55], spectral factorisation for the design of finite impulse response filters [3, 67], and phase unwrapping [70] in signal processing.

Computing the roots of a polynomial is a classical problem, and although a lot of excellent root finding algorithms are available, the computation of the roots of ill-conditioned polynomials is still drawing the attention of several researchers. Among these ill-conditioned polynomials is the polynomial whose zero set contains one or more multiple roots (those roots of multiplicity $k > 1$). Several methods have been introduced to solve this class of polynomials. However, most of the root finding algorithms experience difficulties in computing the roots of degree more than four [44]. This is mainly due to the numerical instability of the roots with high multiplicities [19, 76]. Moreover, the polynomial, in practice, is known in a perturbed form, $f(y) = \hat{f}(y) + e$, where $e$ is the noise attached to the exact polynomial $\hat{f}(y)$. This noise may occur due to roundoff or measurement errors, which in turn, deteriorates the robustness of not only the algorithms for computing multiple roots, but also of those algorithms designed for computing simple roots.

Well known numerical root finding methods include Newton's method [25, 58, 66], Müller's method [25, 58, 66], Bairstow's method [25], Graeffe's root squaring method [25, 66], Laguerre's method [25, 58], and the companion matrix eigenvalue method [9, 58]. These methods are adequate for normal well-conditioned polynomials that are of moderate degree with simple well-separated roots. As the degree of the polynomial increases, or the multiplicity of one or more of its roots increases, or the separation between its roots decreases, the quality of the results obtained from classical methods deteriorates. The reason lies in the fact that the multiple roots are extremely ill-conditioned i.e. they are very sensitive to small perturbations. As a result, in a floating point environment, roundoff errors will be sufficient to change the roots' distribution such that clusters of simple roots are formed around the multiple root.

It is therefore natural to expect that the case would be worse if the roots were closely spaced (nearly multiple roots). These roots pose the most difficult problems for the numerical algorithms [58].

This discussion leads to the aim of this thesis:

> The aim of this thesis is to develop a polynomial root finder that computes multiple roots of a univariate polynomial whose coefficients are corrupted by noise.

The theoretical development of this root finder involves the computation of a structured low rank approximation of the Sylvester resultant matrix, optimising parameters using linear programming, and solving least squares equality and non-linear least squares problems.

The rest of this chapter provides summaries of some commonly used root finding methods in Section 1.1, and presents some of the computational challenges that are associated with the computation of multiple roots in Section 1.2, using illustrative examples. These examples show that the computation of multiple roots is a non-trivial task and hence provide the motivation for the work presented in this thesis.

## 1.1 Standard root finding methods

This section gives a brief review of some of the classical methods for computing the roots of a polynomial. These methods include Newton's method [25, 58, 66], Müller's method [25, 58, 66], Bairstow's method [25], Graeffe's root squaring method [25, 66], Laguerre's method [25, 58], and the companion matrix eigenvalue method [9, 58].

### Newton's method

Newton's method, also referred to as the Newton-Raphson method, is a well known iterative method. This method computes the roots by approximating the function, $f(y)$, linearly, using the tangent of the function at an arbitrary point. Thus, it requires the evaluation of both the function and its derivative at that point. Given a good initial root estimate, $y_0$, that is not very far from the desired root, Newton's method can give iteratively better estimates $y_1, y_2, \cdots$, such that

$$y_{n+1} = y_n - \frac{f(y_n)}{f^{(1)}(y_n)}, \qquad n = 0, 1, \cdots .$$

These iterations should stop either when the successive estimates are very close to each other or the function value is very small. To prevent this method from converging to the same root in successive iterations, each computed root is deflated from the polynomial, and the deflated polynomial is then used in the next run of the iterative scheme. The convergence of this method is very fast if the initial estimates are sufficiently close to the exact root.

### Müller's method

Müller's method computes the zeros of the function $f(y)$ using quadratic approximation. It requires three initial roots estimates, $y_{k-2}$, $y_{k-1}$ and $y_k$, to compute the next approximation,

$$y_{k+1} = y_k - (y_k - y_{k-1}) \left[ \frac{2C}{\max\left(B \pm \sqrt{B^2 - 4AC}\right)} \right], \tag{1.1}$$

where

$$\begin{aligned} A &= qf(y_k) - q(1+q)f(y_{k-1}) + q^2 f(y_{k-2}) \\ B &= (2q+1)f(y_k) - (1+q)^2 f(y_{k-1}) + q^2 f(y_{k-2}) \\ C &= (1+q)f(y_k) \\ q &= \frac{y_k - y_{k-1}}{y_{k-1} - y_{k-2}}. \end{aligned}$$

To prevent the next estimated root from going too far from the current estimate, the sign in the denominator of (1.1) is chosen such that its absolute value is as large as possible. Starting with initial real estimates, Müller's method may converge to a complex estimate. It convergence rate is almost the same as Newton's method [66].

### Bairstow's method

Bairstow's method only works with polynomials whose coefficients are real. It is well known that the complex roots of such polynomials occur in complex conjugate pairs. To avoid complex arithmetic, this method extracts the quadratic factors that may generate these complex conjugate roots. Consider the polynomial

$$f(y) = a_n y^n + a_{n-1} y^{n-1} + \cdots + a_1 y + a_0, \tag{1.2}$$

where $a_i$, $i = 0, \cdots, n$, are the real coefficients of $f(y)$, and let a quadratic factor be $y^2 + py + q$. The polynomial $f(y)$ can then be written as

$$f(y) = (y^2 + py + q)(b_{n-2} z^{n-2} + b_{n-3} z^{n-3} + \cdots + b_0) + ry + s, \tag{1.3}$$

where $b_n = b_{n-1} = 0$ and $ry + s$ is the remainder of dividing $f(y)$ over the quadratic function. If the quadratic function is an exact divisor, then the remainder is equal to zero i.e. $r = s = 0$, and the roots of the quadratic function are also roots of $f(y)$. This however, requires good initial guesses for the values of $p$ and $q$, after which Newton's method is used to change the values of $p$ and $q$, such that the roots of the quadratic function are roots of $f(y)$, which certainly makes the values of $r$ and $s$ equal to zero. Equating the coefficients of (1.2) and (1.3) gives

$$\begin{aligned} b_k &= a_{k+2} - pb_{k+1} - qb_{k+2}, \quad k = n-2, \cdots, 0, \\ r &= a_1 - pb_0 - qb_1, \quad \text{and} \quad s = a_0 - qb_0. \end{aligned}$$

The solution of $a_1 - pb_0 - qb_1 = 0$ and $a_0 - qb_0 = 0$ yields the values of $p$ and $q$ for which the roots of the quadratic function are roots of $f(y)$ as well. The polynomial $f(y)$ is then deflated and the process is repeated for the deflated polynomials to reduce the effort of computing the roots in each step.

Given good initial estimates, this method converges quadratically, but it converges linearly if the multiplicity of the quadratic factor is greater than one.

### Graeffe's root squaring method

This method transforms the original polynomial to another polynomial of the same degree with new coefficients from which the roots of the original polynomial can be computed directly. This transformation requires successive squaring of the original roots by which the new roots spread widely apart if the original ones are real and distinct with absolute values greater than one. To illustrate this process, consider Example 1.1.

**Example 1.1.** Let

$$f(y) = (y + \alpha_1)(y + \alpha_2)(y - \alpha_3),$$

where $\alpha_i$, $i = 1, 2, 3$, are the absolute values of the distinct real roots. Then

$$\begin{aligned} f(-y) &= (-y + \alpha_1)(-y + \alpha_2)(-y - \alpha_3), \\ &= (-1)^3(y - \alpha_1)(y - \alpha_2)(y + \alpha_3). \end{aligned}$$

Using the binomial identity $(y^2 - \alpha^2) = (y - \alpha)(y + \alpha)$,

$$f(y)f(-y) = (-1)^3(y^2 - \alpha_1^2)(y^2 - \alpha_2^2)(y^2 + \alpha_3^2),$$

and if $z = y^2$, then

$$Q(z) = (-1)^3(z - \alpha_1^2)(z - \alpha_2^2)(z + \alpha_3^2).$$

The process is then repeated until the new roots are well-separated. Suppose that for a polynomial of degree $n$, the process described above is repeated $k$ times, then the roots can be estimated from the following formula

$$\alpha_i = \left| \frac{a_i}{a_{i-1}} \right|^{\frac{1}{2^k}}, \quad i = 1, 2, \cdots, n,$$

where the $a_i$'s are the coefficients of the $k^{th}$ polynomial. □

Example 1.1 considers the polynomial in factored form, but the results are the same for non-factored form. Clearly, this method has a problem if two or more of

its roots are of the same magnitude. Several amendments have been proposed to overcome this problem, but they make the method numerically expensive [29].

**Laguerre's method**

Laguerre's method is motivated by the relations between the roots of the polynomial, and its first and second derivatives. Consider the polynomial

$$f(y) = (y-\alpha_1)(y-\alpha_2)\cdots(y-\alpha_n),$$

whose natural logrithm is

$$\ln|f(y)| = \ln|(y-\alpha_1)| + \ln|(y-\alpha_2)| + \cdots + \ln|(y-\alpha_n)|. \tag{1.4}$$

The first and second derivatives of (1.4) respectively, yield

$$A = \frac{1}{(y-\alpha_1)} + \frac{1}{(y-\alpha_2)} + \cdots + \frac{1}{(y-\alpha_n)} = \frac{f^{(1)}}{f},$$

and

$$-B = -\frac{1}{(y-\alpha_1)^2} - \frac{1}{(y-\alpha_2)^2} - \cdots - \frac{1}{(y-\alpha_n)^2} = \frac{f^{(2)}}{f} - \left(\frac{f^{(1)}}{f}\right)^2.$$

Laguerre's method then assumes that the required root $\alpha_i$ is located at a distance $a$ from the current estimate, that is $a = y_0 - \alpha_i$, and all other roots are distinct and clustered at distance $b$. In terms of $a$ and $b$, the derivatives $A$ and $B$ can be expressed

respectively as

$$A = \frac{1}{a} + \frac{n-1}{b}, \quad \text{and} \quad B = \frac{1}{a^2} + \frac{n-1}{b^2}.$$

Hence,

$$a = \frac{n}{A \pm \sqrt{(n-1)(nB - A^2)}}.$$

The sign in the denominator should be taken such that the magnitude of the denominator is as large as possible. Starting with an estimate, $y_0$, the value of $a$ is computed and used in computing the next estimate $y_0 - a$. In each iteration it is required to calculate the values of the polynomial and its first and second derivatives at the current estimate, which is a disadvantage. An important property of this method is that, for any initial choice (i.e. not necessarily close enough from the true root), it always converges to a root if the roots of the polynomial are real. Laguerre's method converges cubically for the simple roots but linearly for the multiple roots. Moreover, starting with real initial estimates, Laguerrs's method may converge to a complex root.

### Companion matrix eigenvalue method

Using the companion matrix, the root finding problem reduces to the eigenvalue problem. Consider the monic polynomial

$$f(y) = y^n + a_{n-1}y^{n-1} + \cdots + a_1 y + a_0,$$

and let $C$ be an $n \times n$ square matrix, whose subdiagonal is filled with ones and whose last column contains the negative of the coefficients of the monic form of $f(y)$,

$$\mathbf{C} = \begin{pmatrix} 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & \cdots & 0 & -a_3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & -a_{n-1} \end{pmatrix}.$$

This matrix is referred to as the *companion matrix.* Its characteristic polynomial is given by [5], page 11,

$$\det(yI - \mathbf{C}) \equiv y^n + a_{n-1}y^{n-1} + \cdots + a_1 y + a_0 = f(y). \tag{1.5}$$

It follows from the identity (1.5) that the roots of $f(y)$ are exactly equal to the eigenvalues of $\mathbf{C}$. In other words, instead of finding the roots of $f(y)$, it is sufficient to compute the eigenvalues of the corresponding companion matrix. Recently, fast efficient algorithms have been introduced for the eigenvalue computation using the QR algorithm [4, 7, 9]. This is how the MATLAB function `roots()` computes the roots of $f(y)$ [30]. The stability and the accuracy of this method have been reported by Edelman and Murakami [22]. However, Cleve Moler [45] has pointed out that this method is computationally more expensive than the methods that are specifically designed for root finding algorithms.

The methods discussed above may yield satisfactory results if the polynomial is of moderate degree and its roots are well-separated, but an exception of this is the

Wilkinson polynomial [76]

$$f(y) = \prod_{i=1}^{20}(y - i),$$

whose roots are equally spaced.

On the other hand, the performance of these methods deteriorates as the degree of the polynomial or the multiplicities of its roots increase. Moreover, these methods fail if the coefficients of the polynomials are known imperfectly or the computations of the roots are performed in a floating point environment. Therefore, better methods should be developed to over come the ill-conditioned nature of polynomials that have multiple roots. This task has been handled by the work presented in this thesis, and a root solver has been developed. Some of the results of computing multiple roots using the developed root solver are presented in Examples 1.2 and 1.3.

**Example 1.2.** The first and second columns of Table 1.1 define the roots and associated multiplicities of the exact polynomial $p_1(y)$. The coefficients of $p_1(y)$ were perturbed by noise with componentwise noise-to-signal ratio of $\varepsilon_c = 10^{-8}$. The roots and associated multiplicities of the perturbed polynomial were calculated using the developed root solver, and the results of this computation are shown in the third, fourth and fifth columns of the table. □

**Example 1.3.** The same procedure used for Example 1.2 was applied to the polynomial $p_2(y)$ whose roots and associated multiplicities are given in the first and second columns of Table 1.2. The results of computing the roots and multiplicities of $p_2(y)$ after perturbing its coefficients are shown in the third, forth and fifth columns in the table. □

Table 1.1: The roots and multiplicities of $p_1(y)$.

| exact root | exact mult. | computed root | computed mult. | relative error |
|---|---|---|---|---|
| 2.62220000e+000 | 10 | 2.62220000e+000 | 10 | 2.12462358e-011 |
| -3.80360000e+000 | 10 | -3.80360002e+000 | 10 | 4.50940637e-009 |
| -1.26210000e+000 | 9 | -1.26210000e+000 | 9 | 3.11056307e-009 |
| -6.74950000e+000 | 6 | -6.74949998e+000 | 6 | 2.54458742e-009 |

Table 1.2: The roots and multiplicities of $p_2(y)$.

| exact root | exact mult. | computed root | computed mult. | relative error |
|---|---|---|---|---|
| -2.65620000e+000 | 5 | -2.65619930e+000 | 5 | 2.64976497e-007 |
| -5.21420000e+000 | 3 | -5.21421303e+000 | 3 | 2.49930364e-006 |
| 6.52500000e-001 | 3 | 6.52500010e-001 | 3 | 1.48046396e-008 |
| 1.07770000e+000 | 3 | 1.07769985e+000 | 3 | 1.40982369e-007 |
| 1.57850000e+000 | 3 | 1.57850048e+000 | 3 | 3.06745235e-007 |
| 3.60130000e+000 | 3 | 3.60129667e+000 | 3 | 9.25308150e-007 |
| 7.33770000e+000 | 3 | 7.33770947e+000 | 3 | 1.29126972e-006 |
| -7.74770000e+000 | 2 | -7.74766883e+000 | 2 | 4.02330826e-006 |
| -1.86450000e+000 | 2 | -1.86450022e+000 | 2 | 1.19410112e-007 |

Examples 1.2 and 1.3 show that the developed method preserves the multiplicities of the roots in the presence of noise. It is noted that the relative error of each computed root in Example 1.2 is less than $\epsilon_c$, even though two of the roots have multiplicity 10. The relative errors of the computed roots in Example 1.3 are greater than $\epsilon_c$, but there are some closely separated roots. These results should be compared with Zeng [85] who has developed a root solver that is explicitly designed for the computation of multiple roots of a polynomial. It achieves excellent results if the data is exact. However, in contrast to the method developed in this thesis, it is shown in Chapter 9, where more examples are given, that it gives incorrect result if inexact

data is considered and the noise level is not specified, or the incorrect noise level is specified.

The subsequent chapters illustrate the theoretical and computational implementation of the developed methods, and show how the results in Examples 1.2 and 1.3 were achieved. In order to motivate the difficulty of the problem of computing multiple roots in the presence of noise, the next section provides some examples that illustrate some of the challenges that arise with the computation of multiple roots of a polynomial.

## 1.2 Computational challenges of root finding algorithms

There are several classes of ill-conditioned polynomials, such as the polynomials whose roots are multiple, closely spaced or a combination of these classes. Furthermore, the computation of the roots of such polynomial classes becomes more challenging if their coefficients are imperfectly known. This section gives two examples to show some of the difficulties that are associated with the computation of roots of ill-conditioned polynomials. Example 1.4 illustrates the effect of roundoff errors associated with the computation of the multiple roots. Example 1.5 shows the effect of both roundoff and measurement errors on the computation of multiple roots of a polynomial.

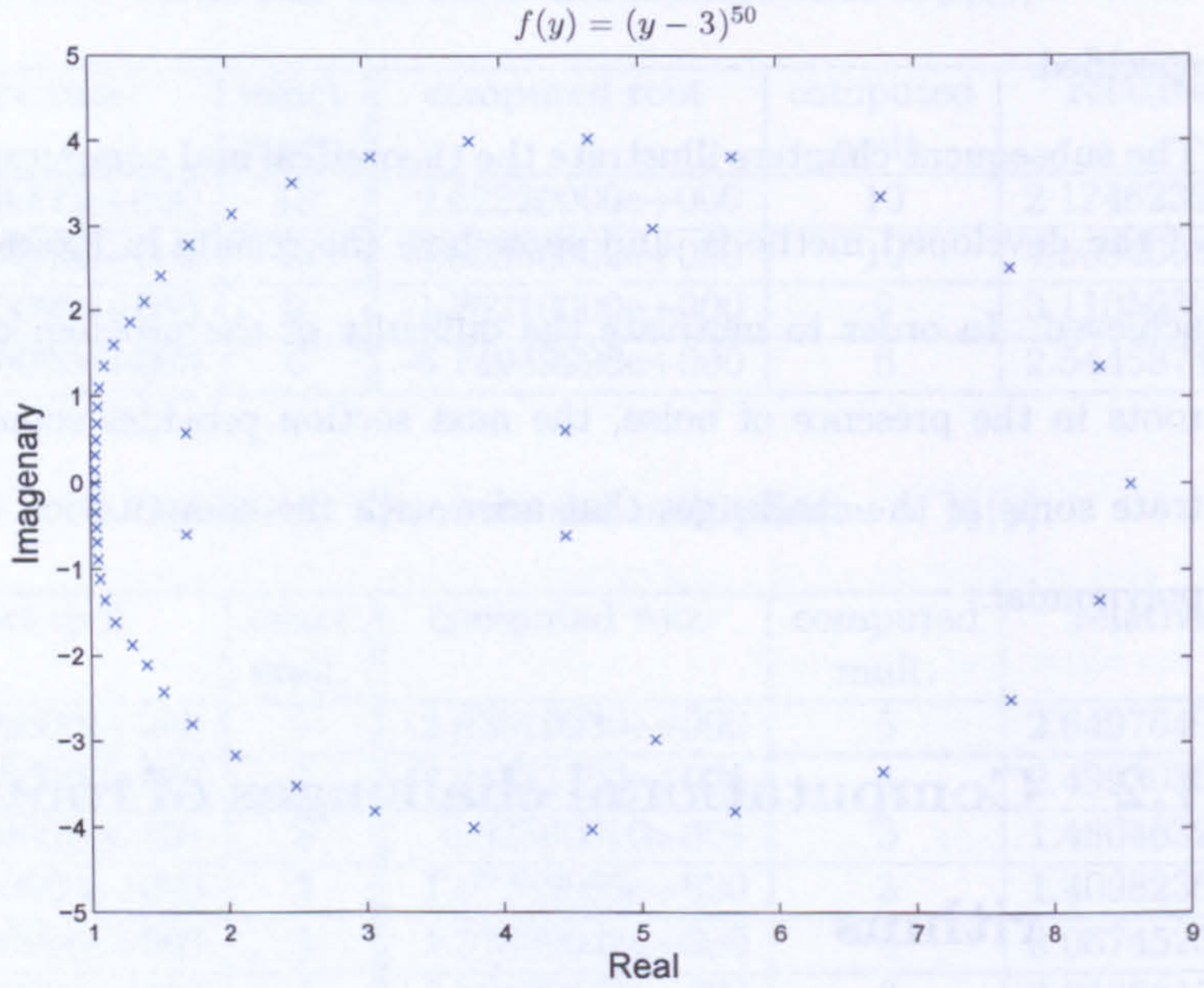



Figure 1.1: The plot of the computed roots of $f(y) = (y-3)^{50}$.

**Example 1.4.** Consider the polynomial

$$y^5 - 15y^4 + 90y^3 - 270y^2 + 405y - 243 = (y-3)^5,$$

which clearly has only one root of multiplicity 5 at $y = 3$. However, using the MATLAB function `roots()`[1], the following roots are returned,

$$3.0033, \quad 3.0010 \pm 0.0031i, \quad 2.9974 \pm 0.0019i.$$

This shows that the rounding errors which are about $10^{-16}$, are sufficient to cause a relative error of about $10^{-3}$ in the computed roots.

Figure 1.1 shows the roots of the above polynomial after increasing the multiplicity

[1] `roots()` uses the QR algorithm to compute the eigenvalues of the companion matrix.

of its root to 50. The multiple roots are now split into 50 distinct roots. These results suggest that as the multiplicities of the roots increase, the effect of roundoff errors becomes more significant, which causes a deterioration of the root finding algorithms. □

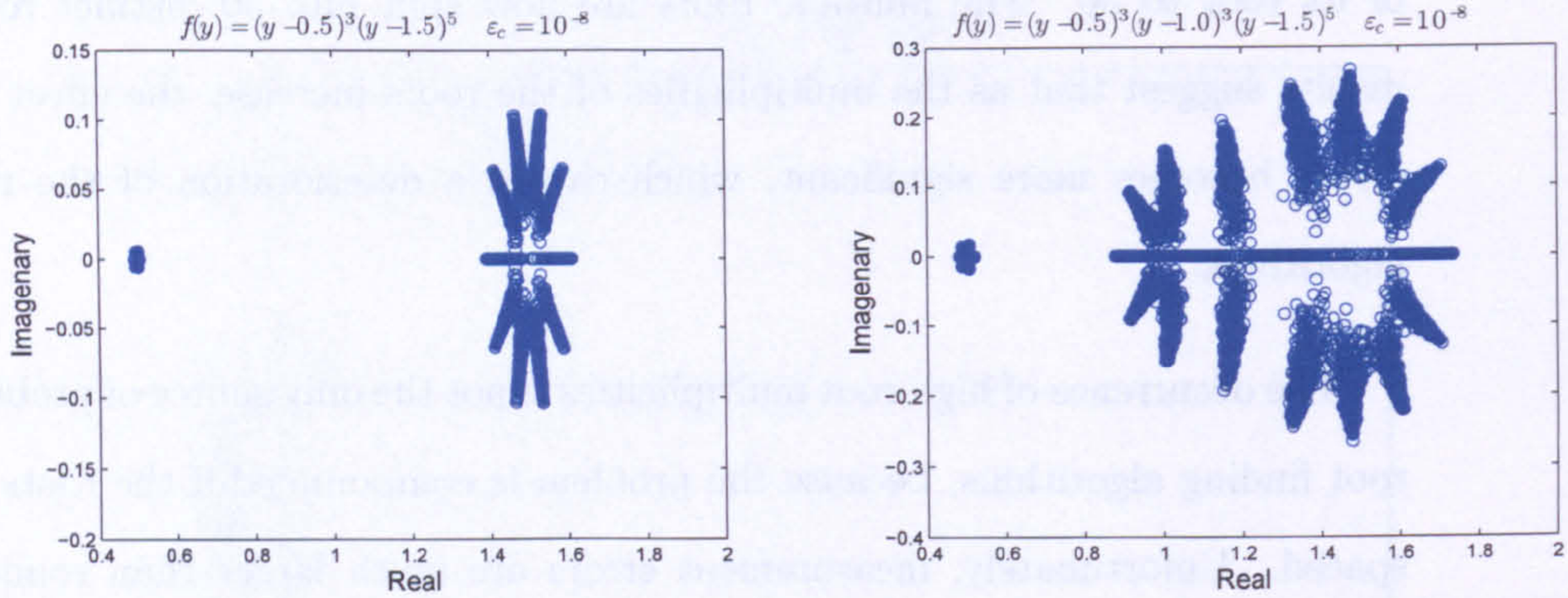
The occurrence of high root multiplicities is not the only source of problems for the root finding algorithms, because the problem is compounded if the roots are closely spaced. Unfortunately, measurement errors are much larger than roundoff errors, which makes the task of computing the roots of inexact polynomials more challenging. The following example illustrates the problem of computing the roots of an inexact polynomial, where roundoff errors and measurement errors are unavoidable.

**Example 1.5.** The MATLAB function `roots()` was used to compute the roots of the polynomial $f(y)$,

$$f(y) = (y - 0.5)^3(y - 1.5)^5,$$

which were evaluated 1000 times, after perturbing their coefficients by noise with signal-to-noise level of $\epsilon_c^{-1} = 10^8$ in a componentwise sense.

Figure 1.2(a) shows the plot of the computed roots. Clearly, it can be seen that the polynomial $f(y)$ contains two distinct roots, and approximations of these two roots can be calculated by evaluating the arithmetic means of the corresponding clusters. The clusters of roots have been studied by several researchers [33, 46, 64, 65] using computational methods such as symbolic-numeric methods [65], and algebraic methods [64], for computing the roots as well as their corresponding multiplicities. Although, clustering seems to be simple, it is restricted to well-separated clusters,




(a) (b)

Figure 1.2: The plot of the computed roots of (a) $f(y) = (y-0.5)^3(y-1.5)^5$, and (b) $f(y) = (y-0.5)^3(y-1)^3(y-1.5)^5$, whose coefficients have been perturbed by noise with signal-to-noise level of $\varepsilon_c^{-1} = 10^8$ in a componentwise sense.

where each cluster originates from one multiple root. If for example, a zero at $y = 1$ of degree 3 occurs between the two roots $y = 0.5$ and $y = 1.5$, in the polynomial given above, the clustering approach will fail to compute the correct values of the roots, as shown in Figure 1.2(b). □

Example 1.4 and 1.5 show that roundoff errors due to floating point arithmetic, and inexact data, can cause significant deterioration in the computed roots. Unfortunately, the presence of these two sources of errors are not avoidable. In particular, roundoff errors are always present in numerical computations and the uncertainties in the data due to measurement errors, for example, can not be avoided. It is therefore important to study the behavior of the roots of a polynomial in the presence of noise to obtain a better understanding of the problem, and to develop a numerical method that deals carefully with a corrupted polynomial, and this forms the main task of this thesis.

This thesis develops a method which is designed for the computation of multiple roots of a polynomial, whose coefficients are corrupted by noise. This method differs from the commonly used root solvers. It first computes the multiplicities of the roots of the given polynomial whose exact form is assumed to have multiple roots, then uses these multiplicities as constraints for the computation of the roots. By contrast, the classical methods described in Section 1.1 compute the roots directly without any restrictions that relate the roots to their multiplicities, and therefore, multiple roots are broken up into several simple roots.

The developed method follows the method by Uspensky [74], which involves:

1. The computation of the greatest common divisor (GCD) of several pairs of polynomials.

2. The computation of the division of several pairs of polynomials.

3. The solution of several polynomial equations, all of whose roots are simple and distinct.

The subsequent chapters illustrate the theoretical and numerical computations of the developed method. A brief description of the rest of this thesis is now given, whereas a detailed thesis layout is given after describing the developed method in Chapter 2 (see Section 2.4).

Chapter 2 studies the ill-conditioned nature of a multiple root and describes the method in [74], whose computational implementation is considered in this thesis, for the computation of multiple roots of a polynomial. It is shown that the implementation of this method in a floating point environment is a very challenging task because it involves ill-posed operations. Chapter 2 also describes the required modifications

to the method, in order for it to be implemented in a floating point environment with inexact data. A geometrical interpretation of the developed method is also included in Chapter 2.

The crucial part of the method is the determination of the multiplicities of the roots of a polynomial. They are computed by successive GCD computations. It is shown in Chapter 3 that the Sylvester subresultant matrix can be used for the computation of the GCD of two exact polynomials. This work is, however, designed for inexact data, and therefore it is necessary to modify the theory in Chapter 3. In particular, the coefficients of the given inexact polynomial must be preprocessed before being involved in the GCD computations, and it is shown in Chapter 4 that three preprocessing operations are required.

An overview of the previous work in GCD computations is given in Chapter 5, after which the modifications to the theory in Chapter 3, using non-linear structure preserving matrix methods, are considered in Chapters 6 and 7.

As noted above, the algorithm used for computing multiple roots of a polynomial requires the computation of the division of several pairs of polynomials. A robust method for this computation is described in Chapter 8.

The last stage in the method requires solving several polynomial equations, all of whose roots are simple. The computation of these roots and their refinement are considered in Chapter 9. This section also contains examples to demonstrate the application of the developed method to the computation of the roots of the theoretically exact form of an inexact polynomial. The conclusion and future work are then given in Chapter 10.

## 1.3 Summary

This chapter has illustrated the importance of developing a root solver that computes multiple roots of the exact form of an inexact polynomial. Some commonly used numerical methods for the computation of the roots of a polynomial have been reviewed, and their advantages and disadvantages have been stated. In order to motivate the difficulty of the problem, some of the challenges that are associated with the computation of multiple roots have been presented. It is concluded that a careful study for the behavior of a multiple root in the presence of noise is needed, in order to develop a better understanding and solution of the problem. This task is considered in the next chapter.

# Chapter 2

# Ill-conditioned polynomials

It was shown in Chapter 1 that the computation of a multiple root of a polynomial is an ill-conditioned problem because small errors, including roundoff errors, are sufficient to cause incorrect results with large errors. This chapter consists of two parts. The first part studies the sensitivity of a multiple root to perturbations in the coefficients of the polynomial. The forward error, backward error and condition number of a root are defined to quantify the result of computing a multiple root. It is shown that, with respect to random perturbations, the sensitivity of a multiple root increases as its multiplicity increases. On the other hand, a multiple root is insensitive to the structured perturbations that keep its multiplicity. It is concluded that a robust root finder requires that the multiplicities of the roots of the given polynomial be first determined, after which the values of these roots are computed. These computed roots are then refined, under the condition that their computed multiplicities are retained. The second part of this chapter describes the developed root finder that satisfies these requirements, and provides a geometrical interpretation of its implementation in a floating point environment. A detailed outline of the contents

of this thesis is given at the end of this chapter.

## 2.1 Forward and back word errors, and condition number

A polynomial is considered to be ill-conditioned if a small perturbation in its coefficients results in a big change in the solution. In order to quantify this solution, the numerical analysis concepts of *forward error*, *backward error* and *condition number* should be considered.

Let $\tilde{x}$ be the approximate value of $x = f(y)$. The question that then arises is: *How good is this approximation?* The simplest error measures are the absolute and relative errors,

$$\text{Absolute error} = |x - \tilde{x}|\,,$$
$$\text{Relative error} = \frac{|x - \tilde{x}|}{|x|}.$$

Though the computation of these error measures is straight forward, it is not always possible, since the exact value may not be known. The backward error on the other hand, does not suffer from this problem, as it examines the input data $y + \delta y$, for which the problem was actually evaluated. Thus, the obvious difference between the forward and the backward errors, which is illustrated in Figure 1.1, is that the forward error measures the distance, in the output space, between the exact and computed solutions, whereas the backward error measures the distance, in the input space, between the data $y$ for which the solution is sought and the data $y + \delta y$ for which the

solution has been actually computed.

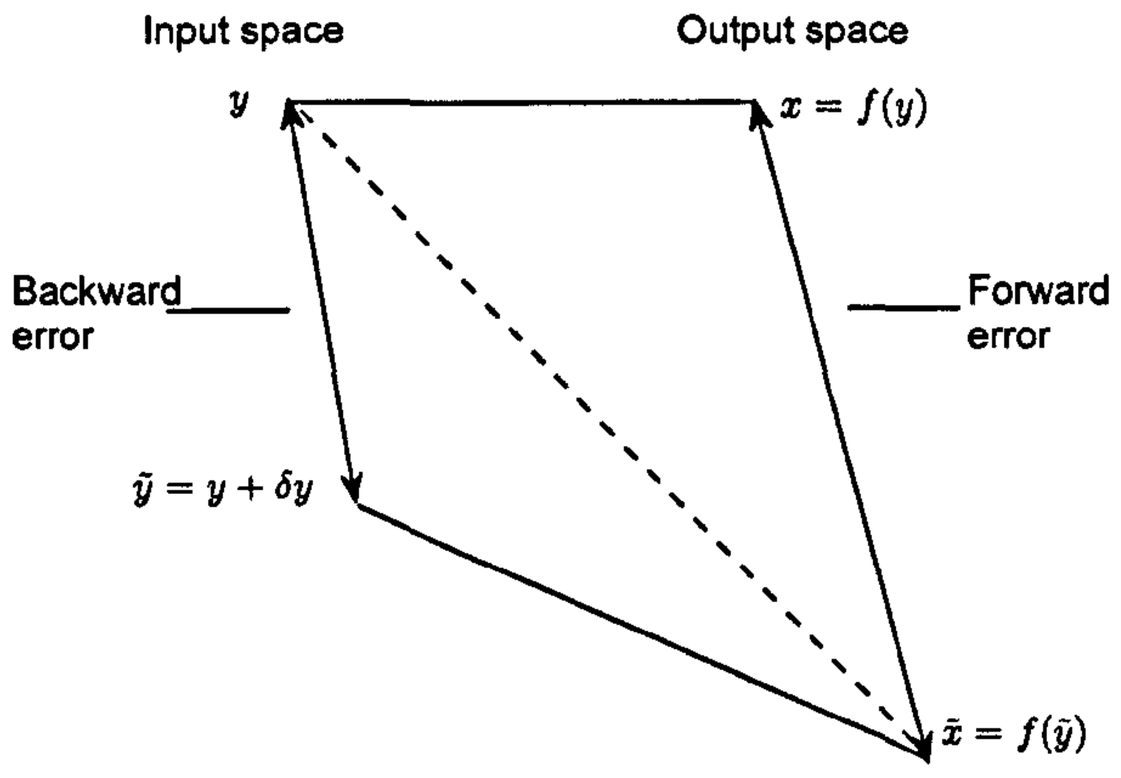



Figure 2.1: The backward and forward errors computed for $x = f(y)$, such that $\tilde{x}$ is the approximate value of $x$. The solid lines represent the exact computation and the dashed line represents the approximated computation. This figure was reproduced from [30].

However, both the forward and the backward errors are interrelated to express the sensitivity of the problem, which is referred to as the *condition number*. In particular, if for a certain numerical problem, a small backward error results in a large forward error, the problem is considered to be ill-conditioned.

In terms of a system of linear equations, $Ay = b$, where $A$ is a non-singular square matrix and $b$ is a non-zero vector, the solution of this system is $y = A^{-1}b$. For a small perturbation in $b$, the system $A\tilde{y} = b + \delta b$, has the solution $\tilde{y} = A^{-1}(b + \delta b)$, where

$\tilde{y} = y + \delta y$. This implies that $\delta y = A^{-1}\delta b$, thus

$$\|\delta y\| \leq \|A^{-1}\|\|\delta b\|. \tag{2.1}$$

Similarly,

$$\|b\| \leq \|A\|\|y\|. \tag{2.2}$$

Multiplying (2.1) and (2.2), then yields

$$\|\delta y\|\|b\| \leq \|A^{-1}\|\|A\|\|y\|\|\delta b\|,$$

or equivalently

$$\frac{\|\delta y\|}{\|y\|} \leq \kappa(A)\frac{\|\delta b\|}{\|b\|},$$

where $\kappa(A) = \|A^{-1}\|\|A\|$, which is referred to as the condition number of the matrix $A$. This result can be interpreted as follows: *A relatively small backward error may yield a large forward error, where the ratio between the forward error and backward error is bounded by the condition number.* Thus the relationship between the forward and the backward errors and the condition number is governed by the following formula to lowest order

$$\text{forward error} \lesssim \text{condition number} \times \text{backward error}, \tag{2.3}$$

which has been proved in [47, 73, 75].

The condition number of a multiple root has been studied by several researchers [20, 34, 77], and it is shown that the condition number of a multiple root approaches infinity, with respect to random perturbations, as the signal-to-noise ratio increases. For example, consider the polynomial $f(y) = y^r$, which has $r > 1$ roots at $y = 0$. Perturbing $f(y)$ by $\epsilon > 0$, yields the perturbed polynomial $\tilde{f}(y) = y^r - \epsilon$, which has $r$ complex roots of magnitude $\epsilon^{1/r}$. The backward and forward errors in this case are:

$$\begin{aligned} \text{Backward error} &= \epsilon, \\ \text{Forward error} &= \epsilon^{1/r}. \end{aligned}$$

Clearly it can be seen that for a small value of $\epsilon$, the forward error is very high, and for these values of the forward error and backward error, the condition number that satisfies the formula in (2.3) is

$$\text{Condition number} \geq \frac{\epsilon^{1/r}}{\epsilon},$$

which approaches infinity as $\epsilon$ reduces to zero.

Based on the error model assigned to the coefficients of a polynomial, there exist two types of backward error and condition number. Both of them can be measured in the componentwise and normwise senses. In particular, let

$$f(y) = \sum_{i=0}^{m} a_i \phi_i(y), \tag{2.4}$$

where $\phi_i(y)$, $i = 0, \cdots, m$, is a set of linearly independent basis functions, and $a_i$ is the coefficient of $\phi_i(y)$. Using the componentwise error model, it is assumed that

each coefficient $a_i$ is perturbed to $a_i + \delta a_i$ such that

$$a_i + \delta a_i \leq a_i(1 + r\epsilon_c), \quad i = 0, \cdots, m,$$

where $r$ is a uniformly distributed random variable, whose value fall in the range $[-1, +1]$, and $\epsilon_c^{-1}$ is the componentwise signal-to-noise ratio. It therefore follows that the componentwise error model is defined by

$$|\delta a_i| \leq \epsilon_c |a_i|, \quad i = 0, \cdots, m.$$

The normwise error model is defined by

$$\|\delta a\| \leq \epsilon_n \|a\|,$$

where $\epsilon_n^{-1}$is the normwise signal-to-noise ratio. The definition of the componentwise and normwise error models are given in Definitions 2.1 and 2.2, respectively, and the corresponding condition numbers are also stated. These expressions are taken from [77].

**Definition 2.1.** *The componentwise backward error of the root approximation $\tilde{y}_0$, of the root $y_0$ of $f(y)$ is defined as*

$$\eta_c(\tilde{y}_0) = \min \left\{ \epsilon_c : \Sigma_{i=0}^{m} \tilde{a}_i \phi_i(\tilde{y}_0) = 0 \quad and \quad |\delta a_i| \leq \epsilon_c |a_i|; \tilde{a} = a + \delta a \right\}.$$

An expression for the componentwise condition number of a multiple root $y_0$ of multiplicity $r$, has been derived in [77], for random perturbations. This derivation considers a multiple root $y_0$ of the polynomial $f(y)$ defined in (2.4).

Let the multiplicity of $y_0$ be $r$. It is proved in [77] that the componentwise condition number of $y_0$ is

$$\kappa(y_0) = \frac{1}{\varepsilon_c^{1-\frac{1}{r}}} \frac{1}{|y_0|} \left( \frac{r!}{|f^{(r)}(y_0)|} \sum_{i=0}^{m} |a_i \phi_i(y_0)| \right)^{\frac{1}{r}},$$

where $\varepsilon_c^{-1}$ is the componentwise signal-to-noise ratio.

**Definition 2.2.** *The normwise backward error of the root approximation $\tilde{y}_0$, of the root $y_0$ of $f(y)$ is defined as*

$$\eta_n(\tilde{y}_0) = \min \left\{ \varepsilon_n : \Sigma_{i=0}^{m} \bar{a}_i \phi_i(\tilde{y}_0) = 0 \quad and \quad \|\delta a\| \leq \varepsilon_n \|a\|; \tilde{a} = a + \delta a \right\}.$$

An expression for the condition number associated with the normwise backward error model of a multiple root $y_0$ of multiplicity $r$, has also been derived in [77], for random perturbations. This derivation considers a multiple root $y_0$ of the polynomial $f(y)$ defined in (2.4). Let the multiplicity of $y_0$ be $r$. It is proved in [77] that the normwise condition number of $y_0$ is

$$\kappa_n(y_0) = \frac{1}{\varepsilon_n^{1-\frac{1}{r}}} \frac{1}{|y_0|} \left( \frac{r!}{|f^{(r)}(y_0)|} \|a\| \|\phi(y_0)\| \right)^{\frac{1}{r}},$$

where $\varepsilon_n^{-1}$ is the normwise signal-to-noise ratio. For high multiplicities $r \gg 1$, $\kappa_c(y_0)$ and $\kappa_n(y_0)$ can be approximated by the following formulae,

$$\kappa_c(y_0) \approx \frac{1}{\varepsilon_c |y_0|}, \quad \text{and} \quad \kappa_n(y_0) \approx \frac{1}{\varepsilon_n |y_0|}, \tag{2.5}$$

respectively, and it can be seen that for random perturbation the componentwise

and normwise condition numbers are proportional to the signal-to-noise ratio. More specifically, the higher the signal-to-noise ratio, the more unstable the problem. These results should be compared with the result obtained when a structured perturbation that preserves the multiplicity of the root is considered. This comparison is made in Section 2.2, where it is shown that a multiple root is stable with respect to a perturbation that preserves the multiplicities of the multiple roots.

## 2.2 Geometric interpretation of an ill-conditioned polynomial

It has been shown in Section 1.2 that the computation of multiple roots of a polynomial is an ill-conditioned problem and small perturbations due to roundoff errors may break up the multiple roots into clusters of simple roots. However, Kahan [35] has pointed out that a polynomial is well-conditioned if the perturbations preserve the multiplicities of its roots. In particular, consider the polynomial

$$f(y) = (y-5)^5(y-2)^7(y+9)^{10}.$$

The polynomial $f(y)$ lies on a pejorative manifold which is defined by its multiplicity structure $\mathbf{m} = \{5, 7, 10\}$. Kahan has defined the pejorative manifold for a polynomial with a given multiplicity structure and stated that the roots of a polynomial are sensitive to random perturbations that move the polynomial off this manifold (i.e. the perturbed polynomial does not lie on the manifold on which its unperturbed form lies), but they are insensitive to the structured perturbations that keep the polynomial

on the manifold of its unperturbed form.

The work in this thesis uses Kahan's observations on pejorative manifolds to test the feasibility of the structure preserved matrix methods for robust computations of the multiple roots of high degree, inexact univariate polynomials expressed in the power basis.

The fundamental task in the root solver that is developed in this thesis is the identification of the pejorative manifold on which the theoretically exact polynomial lies. This corresponds to the determination of the multiplicities of the roots of the polynomial. An iterative procedure can then be used to locate the roots on the manifold, such that each iteration stays on the manifold and thus the multiplicities of the roots are preserved in this iterative scheme.

In order to understand Kahan's observations on the pejorative manifold and how they can be applied to the developed root solver, this section explains the theory of pejorative manifolds, and studies the sensitivity of a multiple root to structured perturbations compared to its sensitivity under random perturbations.

### 2.2.1 The pejorative manifold

A polynomial with one or more multiple roots forms a pejorative manifold that is a subset of the space of all polynomials [34]. The multiplicities of the roots of the polynomial are preserved if a perturbation keeps the polynomial on the manifold, but they are destroyed if the polynomial leaves the manifold. This pejorative manifold can be defined via Vieta's system [81].

Consider the monic polynomial $f(y)$ whose $l$ distinct roots are $y_j$, $j = 1, \cdots, l$, with the multiplicity structure $\mathbf{m} = \{m_j\}$, $j = 1, \cdots, l$, such that if the degree of $f(y)$ is

equal to $n$, then $m_1 + m_2 + \cdots + m_l = n$. The polynomial $f(y)$ can be written as

$$\begin{aligned} f(y) &= (y - y_1)^{m_1}(y - y_2)^{m_2} \cdots (y - y_l)^{m_l}, \\ &= y^n + p_1(y_1, y_2, \cdots, y_l)y^{n-1} + p_2(y_1, y_2, \cdots, y_l)y^{n-2} \cdots + p_n(y_1, y_2, \cdots, y_l) \\ &= y^n + a_1 y^{n-1} + a_2 y^{n-2} + \cdots + a_n, \end{aligned} \tag{2.6}$$

where the functions $p_i$, $i = 1, \cdots, n$, define the relation between the roots and the coefficients of $f(y)$. Let the $n$ roots of $f(y)$ be denoted by $(y_1^*, y_2^*, \cdots, y_n^*)$, which are not necessarily distinct. Using Vieta's formulae, $p_i$ can be generalised as follows,

$$\mathbf{P_m(y)} = \begin{cases} p_1 &= \sum_{1 \leqslant j_1 \leqslant n} y_{j_1}^* = -a_1 \\ p_2 &= \sum_{1 \leqslant j_1 < j_2 \leqslant n} y_{j_1}^* y_{j_2}^* = a_2 \\ &\vdots \\ p_k &= \sum_{1 \leqslant j_1 < j_2 < \cdots < j_k \leqslant n} y_{j_1}^* y_{j_2}^* \cdots y_{j_k}^* = (-1)^k a_k \\ &\vdots \\ p_n &= \prod_{1 \leqslant j_1 \leqslant n} y_{j_1}^* = (-1)^n a_n. \end{cases} \tag{2.7}$$

The system in (2.7) is called the coefficient operator and it defines the pejorative manifold of $f(y)$ whose multiplicity structure is $\mathbf{m} = \{m_1, \cdots, m_l\}$. Clearly, it can be seen that the equations in this system constrain the coefficients of $f(y)$, $a_i, i = 1, \cdots, n$, in order for $f(y)$ to have the multiplicity structure $\mathbf{m}$. The pejorative manifold is defined in [85] as follows:

**Definition 2.3.** *For a given multiplicity $\boldsymbol{m} = \{m_1, \cdots, m_l\}$, the collection of vectors $\Pi_m \equiv \{\boldsymbol{P}(\boldsymbol{z}) | \boldsymbol{z} \in \boldsymbol{C}^{\boldsymbol{l}}\} \subset \boldsymbol{C}^{\boldsymbol{n}}$ is called the **pejorative manifold** of multiplicity structure $\boldsymbol{m}$.*

**Example 2.1.** Consider the polynomial $f(y)$, which has one simple root and one triple root,

$$\begin{aligned} f(y) &= (y-y_1)(y-y_2)^3 \\ &= y^4-(y_1+3y_2)y^3+3(y_1y_2+y_2^2)y^2-(3y_1y_2^2+y_2^3)y+y_1y_2^3. \end{aligned}$$

This polynomial lies on the pejorative manifold $\mathcal{M}$ which is defined by the multiplicity structure $\mathbf{m}=\{1,3\}$. In particular, $\mathcal{M}$ lies in $\mathbb{R}^4$ on which all real monic polynomials of degree four with one simple root and one triple root lie. The exact location of $f(y)$ on $\mathcal{M}$ is defined by the values of its roots. It follows that $\mathcal{M}$ is a surface defined by

$$\begin{pmatrix} -(y_1+3y_2) & 3(y_1y_2+y_2^2) & -(3y_1y_2^2+y_2^3) & y_1y_2^3 \end{pmatrix},$$

where $\mathbf{y}=[y_1 \quad y_2]^T \in \mathbb{R}^2$.

If $y_1=y_2$, $f(y)$ has a quadruple root and $f(y)$ can be written as follows

$$f(y)=y^4-4y_1y^3+6y_1^2y^2-4y_1^3y+y_1^4.$$

The polynomial $f(y)$ in this case lies on the pejorative manifold $\mathcal{M}$, which lies in $\mathbb{R}^4$, and is defined by the multiplicity structure $\mathbf{m}=\{4\}$. In particular, $\mathcal{M}$ is the manifold on which all real monic polynomials of degree four with one quadruple root lie. The exact location of $f(y)$ on $\mathcal{M}$ is defined by the values of its roots. It follows that $\mathcal{M}$ is a curve defined by

$$\begin{pmatrix} -4y_1 & 6y_1^2 & -4y_1^3 & y_1^4 \end{pmatrix},$$

where $\mathbf{y} = y_1 \in \mathbb{R}$. □

### 2.2.2 The sensitivity of a multiple root to a structured perturbation

Once the multiplicities of the roots of a polynomial have been determined, the pejorative manifold on which this polynomial lies is defined uniquely. The task now is to study the behavior of this polynomial on its manifold with respect to both random and structured perturbations. It has been noted in Section 2.1 that several researchers have shown that a multiple root is very sensitive to a random perturbation in the coefficients of the polynomial. The sensitivity of a multiple root to a structured perturbation that preserves its multiplicity is now considered.

Considering a structured perturbation, an expression for a componentwise condition number of a root has been derived in [77], and it is shown that a multiple root is insensitive to the perturbation that keeps its multiplicity.

**Theorem 2.1.** *[77] The condition number of the real root $y_0$ of multiplicity $r$ of the polynomial $f(y) = (y - y_0)^r$, such that the perturbed polynomial also has a root of multiplicity $r$ is*

$$\rho(y_0) := \frac{\Delta y_0}{\Delta f} = \frac{1}{r|y_0|} \frac{\|(y-y_0)^r\|}{\|(y-y_0)^{r-1}\|} = \frac{1}{r|y_0|} \left( \frac{\sum_{i=0}^{r} \binom{r}{i}^2 (y_0)^{2i}}{\sum_{i=0}^{r-1} \binom{r}{i}^2 (y_0)^{2i}} \right)^{\frac{1}{2}}, \tag{2.8}$$

*where*

$$\Delta f = \frac{\|\delta f\|}{\|f\|} \quad and \quad \Delta y_0 = \frac{|\delta y_0|}{|y_0|}$$

*Proof*: Let $f(y, y_0) := f(y)$. It follows that

$$\begin{aligned} f(y, y_0) &= (y - y_0)^r \\ &= \sum_{i=0}^{r} \binom{r}{i} y^{r-i}(-y_0)^i \\ &= y^r + \sum_{i=1}^{r} \binom{r}{i} (-1)^i (y_0)^i y^{r-i}. \end{aligned}$$

A neighboring polynomial that also has a root of multiplicity $r$ is

$$f(y, y_0 + \delta y_0) = (y - (y_0 + \delta y_0))^r,$$

and hence

$$\begin{aligned} f(y, y_0 + \delta y_0) - f(y, y_0) &= \sum_{i=1}^{r} \binom{r}{i} (-1)^i \left( (y_0 + \delta y_0)^i - y_0^i \right) y^{r-i} \\ &= \delta y_0 \sum_{i=1}^{r} \binom{r}{i} (-1)^i i y_0^{i-1} y^{r-i} + \mathrm{O}(\delta y_0^2). \end{aligned}$$

Since

$$\begin{aligned} (y - y_0)^{r-1} &= \sum_{i=0}^{r-1} \binom{r-1}{i} y^{r-1-i}(-y_0)^i \\ &= -\frac{1}{r} \sum_{i=1}^{r} \binom{r}{i} (-1)^i i (y_0)^{i-1} y^{r-i}, \end{aligned}$$

it follows that,

$$\delta f := f(y, y_0 + \delta y_0) - f(y, y_0) = -r \delta y_0 (y - y_0)^{r-1},$$

to first order. Therefore, the condition number of $y_0$, under a structured perturbation that preserves its multiplicity is

$$\frac{\Delta y_0}{\Delta f} = \frac{1}{r|y_0|} \frac{\|(y-y_0)^r\|}{\|(y-y_0)^{r-1}\|}, \tag{2.9}$$

and since

$$(y-y_0)^r = \sum_{i=0}^{r} \binom{r}{i} y^{r-i}(-y_0)^i,$$

the result (2.8) follows.

**Example 2.2.** Using (2.8), the condition number $\rho(5)$ of the root $y_0 = 5$ of the polynomial $f(y) = (y-5)^r$ was calculated for the multiplicities $r = 2, 10, 20$, and the results are shown in Table 2.1.

Table 2.1: The condition numbers of the root $y_0 = 5$.

| $r$ | The condition number $\rho(5)$ |
|---|---|
| 2 | 0.5284 |
| 10 | 0.1168 |
| 20 | 0.0592 |

Table 2.1 shows that $\rho(5)$ decreases as the multiplicity increases. This result is related to the fact that the curve $f(y) = (y-\alpha)^r$ becomes flatter at $y = \alpha$ as $r$ goes to infinity, and therefore it is less sensitive to the changes in the root. This result should be compared with the situation when a random perturbation is considered. For $r = 20$, for example, the componentwise condition number of the root $y_0 = 5$ is $\kappa(y_0) \approx \frac{1}{5\epsilon_c}$, which is proportional to the signal-to-noise ratio, where $\kappa(y_0)$ is defined in (2.5). □

Theorem 2.1 shows that the condition number of a multiple root is independent of the noise level if it preserves its multiplicity, and it decreases as the multiplicity increases. Another expression for the condition number of a multiple root, when a structured perturbation is considered, has been derived in [35]. This derivation considers the effect of the distance between neighboring roots on the condition number of a multiple root. It shows that the main factors that affect the sensitivity of a root are its relative distance from the other roots of the polynomial and its multiplicity. The proofs in [35] and [77] agree that the condition number of a multiple root is independent of the noise level, as long as it does not change the multiplicity of this root.

The discussion above supports Kahan's observations and suggests that the crucial stage in the computation of multiple roots of a polynomial is the determination of the multiplicities of its roots (i.e. defining the pejorative manifold on which the polynomial lies). The values of the computed roots can then be improved by a refinement process that preserves the multiplicities of the roots.

The method developed in this thesis for computing multiple roots of a polynomial handles this task efficiently, in addition to the computation of the initial root estimates. These root estimates are then refined under the constraints of the computed multiplicity structure, that is, the pejorative manifold has been identified and all computations are performed on this manifold, thereby guaranteeing numerical stability.

An overview of the method developed for the computation of multiple roots of a polynomial is considered next.

## 2.3 Polynomial root solver overview

This section is devoted to describe the method whose computational implementation is considered in this thesis. The theoretical development of this method for exact data is given in Section 2.3.1 along with an algorithm that describes its implementation. Section 2.3.2 then discusses its computational implementation when inexact data is considered. A geometric interpretation of the developed method is given in Section 2.3.3.

### 2.3.1 Theoretical development

It is well known that simple roots are better conditioned than multiple roots with respect to unstructured perturbations. It is therefore instructive to use the divide and conquer strategy to compute multiple roots of a polynomial, by which the polynomial that has multiple roots is broken up into several polynomials, each of which only has simple roots. Moreover, it has been stated in Section 2.2, that the computation of multiple roots of a polynomial $f(y)$ is more reliable if it is performed under the constraint that the multiplicity structure of $f(y)$ is known. It is therefore instructive to first compute the multiplicity structure of $f(y)$, after which the values of its roots can be computed.

A method that satisfies the above requirements has been developed in this work. This method follows the method described by Uspensky [74], pages $65 - 68$, whose computational implementation is considered in this work. It differs from the classical methods described in Section 1.1 because it first computes the multiplicities of the roots, then it computes the values of these roots. A description of this method is now given.

Let the polynomial $f(y)$ have the following factorised form

$$f(y) = w_1(y)w_2^2(y)w_3^3(y)\cdots w_l^l(y),$$

where $w_i$ is the product of all the factors of degree $i$, $i = 1, 2, \cdots, l$ and $l$ is the highest root multiplicity. If no factor of degree $k$ occurs, then the value of $w_k(y)$ is set equal to one. It follows that

$$\begin{aligned}
q_1(y) &= \text{GCD}\,(f(y), f^{(1)}(y)) = w_2(y)w_3^2(y)\cdots w_l^{l-1}(y)\\
q_2(y) &= \text{GCD}\,(q_1(y), q_1^{(1)}(y)) = w_3(y)w_4^2(y)\cdots w_l^{l-2}(y)\\
q_3(y) &= \text{GCD}\,(q_2(y), q_2^{(1)}(y)) = w_4(y)w_5^2(y)\cdots w_l^{l-3}(y)\\
&\vdots\\
q_l(y) &= \text{GCD}\,(q_{l-1}(y), q_{l-1}^{(1)}(y)) = \text{constant},
\end{aligned}$$

and

$$\begin{aligned}
h_1(y) &= \frac{f(y)}{q_1} &&= w_1(y)w_2(y)w_3(y)\cdots w_l(y)\\
h_2(y) &= \frac{q_1(y}{q_2(y)} &&= w_2(y)w_3(y)\cdots w_l(y)\\
h_3(y) &= \frac{q_2(y)}{q_3(y)} &&= w_3(y)w_4(y)\cdots w_l(y)\\
&&&\vdots\\
h_l(y) &= \frac{q_{l-1}(y)}{q_l(y)} &&= w_l(y).
\end{aligned}$$

The factors $w_i(y)$, $i = 1, 2, \cdots, l$, can then be determined from the following equations

$$w_1(y) = \frac{h_1(y)}{h_2(y)},\ w_2(y) = \frac{h_2(y)}{h_3(y)},\ \cdots,\ w_{l-1}(y) = \frac{h_{l-1}(y)}{h_l(y)},\ w_l = h_l(y),$$

whose roots are simple. Recall that $w_i(y)$ includes all the roots of multiplicity $i$, and therefore the root multiplicities are maintained. Algorithm 2.3.1 illustrates the use of this method in computing the roots of a polynomial that has at least one multiple root.

---

**Algorithm 2.3.1: Root solver**

**Input** A polynomials $f(y)$.

**Output** The roots of $f(y)$.

**Begin**

1. Set $j = 0$ and $q_j = f(y)$.
2. **while** $\deg(q_j) > 0$ **do**
   (a) Increment $j$.
   (b) Compute $q_j = \mathrm{GCD}(q_j, q_j^{(1)})$.
   (c) Compute $h_j = \frac{q_{j-1}}{q_j}$.
   (d) **if** $j > 1$ **then**
      i. Compute $w_{j-1} = \frac{h_{j-1}}{h_j}$.
      ii. Compute the roots of $w_{j-1}$.

   **end**

   **else** Compute the roots of $w_j$.

**end**

3. Set $w_j = h_j$ and find the roots of $w_j$.

---

Clearly it can be seen that Algorithm 2.3.1 ends with the factors $w_j(y)$, $j = 1, \cdots, l$. These factors might be constants or polynomials whose roots $y_j$ are simple, and if $\alpha$ is a root of $w_j(y)$ then $\alpha^j$ is a root of $f(y)$. Thus Algorithm 2.3.1 reveals the multiplicity structure of the given polynomial in addition to computing its roots.

Geometrically, the theoretically exact polynomial lies on a pejorative manifold, since it is assumed that it has at least one multiple root. If deg GCD $> 0$ in Step 2 of Algorithm 2.3.1, the pejorative manifold on which the exact polynomial lies is not defined uniquely, and more GCD computations are required to be performed. When deg GCD $= 0$, the pejorative manifold on which the theoretically exact polynomial lies, is defined uniquely. The roots of $w_j, j = 1, \cdots, l$ computed in Step 3 of Algorithm 2.3.1, define the unique point on this pejorative manifold, that represents the exact polynomial.

**Example 2.3.** Consider the polynomial

$$f(y) = (y-1)^3(y+3)^2(y-2) = q_0(y),$$

and its derivative

$$f^{(1)}(y) = (y-1)^2(y+3)(6y^2 - y - 17) = q_0^{(1)}(y).$$

Algorithm 2.3.1 yields

$$\begin{aligned}
q_1(y) &= \mathrm{GCD}\,(q_0(y), q_0^{(1)}(y)) = (y-1)^2(y+3), & q_1^{(1)} &= (y-1)(3y+5)\\
q_2(y) &= \mathrm{GCD}\,(q_1(y), q_1^{(1)}(y)) = (y-1), & q_2^{(1)} &= 1\\
q_3(y) &= \mathrm{GCD}\,(q_2(y), q_2^{(1)}(y)) = 1, & q_3^{(1)} &= 0,
\end{aligned}$$

$$\begin{aligned}
h_1(y) &= \tfrac{q_0(y)}{q_1(y)} = (y-1)(y+3)(y-2)\\
h_2(y) &= \tfrac{q_1(y)}{q_2(y)} = (y-1)(y+3)\\
h_3(y) &= \tfrac{q_2(y)}{q_3(y)} = (y-1),
\end{aligned}$$

$$\begin{aligned}
w_1(y) &= \tfrac{h_1(y)}{h_2(y)} = (y-2)\\
w_2(y) &= \tfrac{h_2(y)}{h_3(y)} = (y+3)\\
w_3(y) &= h_3(y) = (y-1).
\end{aligned}$$

It follows that $f(y)$ has one root at $y = 2$, a double root at $y = -3$ and a triple root at $y = 1$. Moreover, the polynomial $f(y)$ lies on the pejorative manifold $\mathcal{M} \in \mathbb{R}^5$ which is defined by the multiplicity structure $\mathbf{m} = \{1, 2, 3\}$. □

Although the flow of the operations in Algorithm 2.3.1 seems to be easy, its implementation in a floating point environment and/or with inexact data raises some numerical challenges. In particular, the implementation of Algorithm 2.3.1 involves:

1. The computation of the GCD of several pairs of polynomials.

2. The computation of the division of two polynomials.

3. The solution of several polynomial equations, each of which only has simple roots.

The first two operations are ill-posed if the computations are performed in a floating point environment. Therefore, it is required to perform these operations with high care. More details on the computational implementation of Algorithm 2.3.1 are given next.

### 2.3.2 Computational implementation

The first two stages of Algorithm 2.3.1, the computation of the GCD of two polynomials, and the division of two polynomials, are well-defined problems if the data is exact and the computations are performed symbolically. By contrast, the ill-posed nature of these computations makes them non-trivial when inexact data is considered. More precisely, the GCD of two polynomials is not a continuous function of the changes in the coefficients of these polynomials and a small error, including round-off error, is able to turn the two given non co-prime polynomials into two co-prime polynomials. A numerical solution for such problem requires that the two given inexact polynomials $f(y)$ and $g(y)$ be perturbed slightly such that their perturbed forms $\tilde{f}(y) = f(y) + \delta f(y)$ and $\tilde{g}(y) = g(y) + \delta g(y)$ have a non-constant GCD. The resulting GCD is referred to as an *approximate greatest common divisor* (AGCD), because it is an approximate GCD with respect to $f(y)$ and $g(y)$. However, this AGCD is an exact GCD of the corrected polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$.

Similarly, the computation of the division of two polynomials $p(y)$ and $q(y)$ is an ill-posed problem because even if the polynomial division $p(y)/q(y)$ is a polynomial, the polynomial division $\frac{p(y)+\delta p(y)}{q(y)+\delta q(y)}$ is, with high probability, a rational function, for arbitrary small errors $\delta f(y)$ and $\delta g(y)$. Since it is required that $\frac{p(y)+\delta p(y)}{q(y)+\delta q(y)}$ is a polynomial and not a rational function, a procedure similar to that described above is

adapted. In particular, perturbations are added to $p(y) + \delta p(y)$ and $q(y) + \delta q(y)$ such that the polynomial division of the perturbed forms of $p(y) + \delta p(y)$ and $q(y) + \delta q(y)$ yields a polynomial. The work presented in this thesis uses structure preserving matrix methods [60, 61] to produce the proper perturbations that satisfy the above requirements.

### 2.3.3 The geometric interpretation of Algorithm 2.3.1 (inexact case)

It is assumed that the theoretically exact form of the given inexact polynomial has at least one multiple root, and therefore it lies on a pejorative manifold, which is defined by its multiplicity structure. Furthermore, it has been noted in Section 2.3.1 that this multiplicity structure is determined from the successive GCD computations in Algorithm 2.3.1. The case is different when the inexact polynomial is considered. In particular, it is assumed that its roots are simple due to its ill-conditioned nature. Therefore, the inexact polynomial is an isolated point in space and it does not lie on a pejorative manifold. Therefore, the GCD computations in Algorithm 2.3.1 must be replaced by AGCD computations. Considering the modification procedure discussed in Section 2.3.2 for these computations leads to the following geometric interpretation: Each AGCD computation represents an orthogonal projection on to a pejorative manifold, since the nearest AGCD is required as stated in Definition 5.2. If deg AGCD $> 0$, the pejorative manifold on which the exact polynomial lies is not defined uniquely, and more AGCD computations are required to be performed. When deg AGCD $= 0$, the pejorative manifold on which the exact polynomial lies, is defined uniquely. This pejorative manifold is defined by the multiplicity structure

whose contents have been computed by the successive AGCD computations. The roots of $w_j, j = 1, \cdots, l$ in Algorithm 2.3.1 are regarded as initial estimates whose values are refined using the method of non-linear least squares, such that the polynomial of the refined roots in each iteration remains on the pejorative manifold, that is, the multiplicity structure of the polynomial is retained.

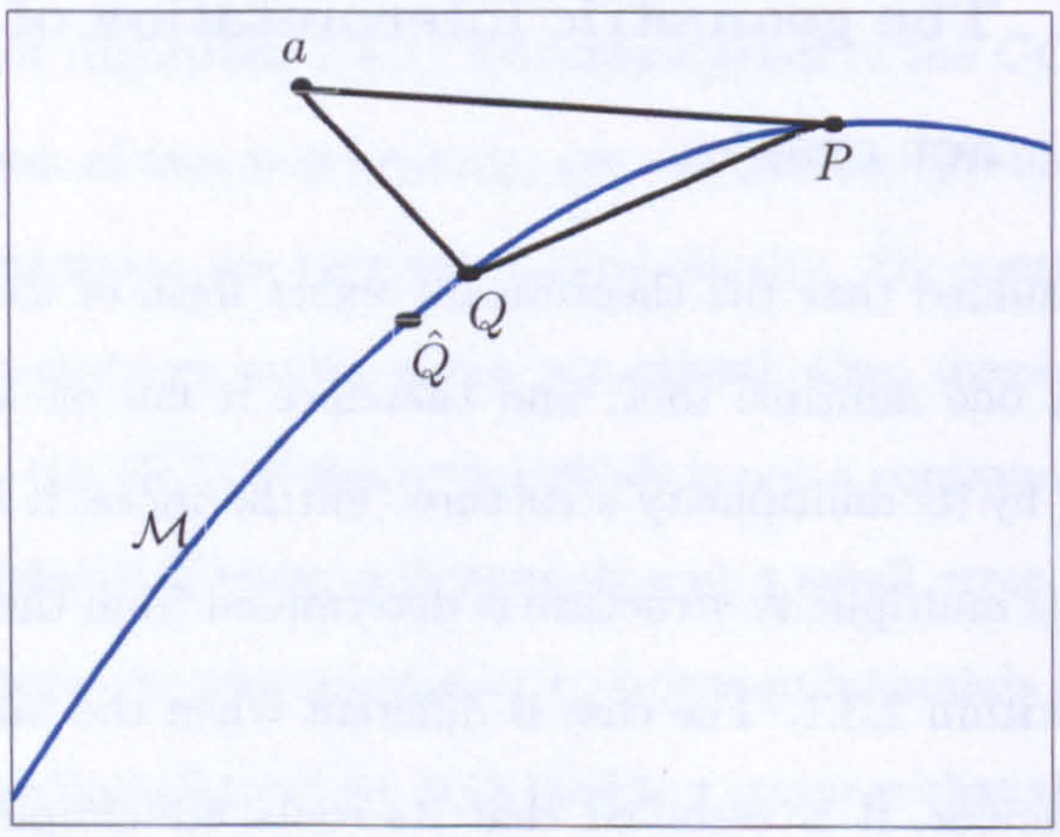



Figure 2.2: Graphical illustration of the refinement of the roots on the pejorative manifold $\mathcal{M}$.

Consider the inexact polynomial $f(y)$ whose theoretically exact form $\hat{f}(y)$ has $l$ distinct roots of multiplicities $m_i$, $i = 1, 2, \cdots, l$, and whose $l$ distinct root initial estimates are $\mathbf{y}_0 = [y_{0,1}, y_{0,2}, \cdots, y_{0,l}]$. Figure 2.2 illustrates the refinement process of the computed roots $\mathbf{y}_0$ graphically. In particular, the multiplicity structure $\mathbf{m} = \{m_1, m_2, \cdots, m_l\}$ defines the pejorative manifold $\mathcal{M}$ on which $\hat{f}(y)$ lies. The points $\hat{Q}, Q$ and $P$ lie on this pejorative manifold, where:

(1) $\hat{Q}$ denotes the point which is defined by the coefficients $\hat{\mathbf{a}}$ of $\hat{f}(y)$.

(2) $Q$ denotes the point that arises from the orthogonal projection of a point $\mathbf{a}$, defined by the coefficients of $f(y)$, onto $\mathcal{M}$, and

(3) $P$ denotes a point that arises from the series of the orthogonal projections performed by the successive AGCD computations, and its exact location on $\mathcal{M}$ is defined by the initial root estimates $\mathbf{y_0}$.

The point $P$ may be distant from the point $Q$, whose location is assumed to be very close to the exact point $\hat{Q}$, and the desire is to move the point $P$ to a point very close to $Q$. However, this movement should be done such that the new location of $P$ is still on $\mathcal{M}$, and this can only be achieved if the multiplicity structure $\mathbf{m}$ is maintained. The movement of $P$ to $Q$ is achieved by the method of non-linear least squares.

## 2.4 Thesis outline

This thesis considers the computational implementation of the method described by Uspensky [74], pages 65-68, for the computation of multiple roots of the theoretically exact form of an inexact polynomial. The description of this method for the computation of the roots of an exact polynomial has been discussed in Section 2.3.1 and it has been shown that it consists of successive GCD computations, successive polynomial divisions and solving several polynomial equations.

The computation of the GCD of two exact polynomials requires that the degree of this GCD be first determined after which its coefficients are computed. It is shown in Chapter 3 that the Sylvester resultant matrix can be used for the computation of the GCD of two exact polynomials. However, it is assumed in this work that the given polynomial has multiple roots and their coefficients are not known perfectly, and thus

some modifications to the theory in Chapter 3 are required.

These modifications are considered in this work, where robust methods are used for the computation of the degree of an AGCD of two in exact polynomials, and structure preserving matrix methods are used for the computations of an AGCD of two inexact polynomials and the polynomial division of two inexact polynomials. Finally, the method of non-linear least squares is used for the refinement of the roots.

It is shown in Chapter 4 that the coefficients of the given inexact polynomial must be processed before being involved in the computation of the AGCD, and three preprocessing operations are discussed in this chapter. These preprocessing operations allow efficient computations of the AGCD.

An overview of AGCD computations is given in Chapter 5. The problem of computing the degree of an AGCD of two inexact polynomials is addressed in Chapter 6 by considering three methods for this computation. Chapter 7 presents two methods for the computation of the coefficients of the AGCD using non-linear structured matrix methods.

A robust method for the computation of the successive polynomial divisions is discussed in Chapter 8. The two sets of polynomial divisions in Algorithm 2.3.1 yield several polynomials, all of whose roots are simple. The computation of these roots and their refinement are considered in Chapter 9. This chapter also contains some examples that demonstrate the application of Algorithm 2.3.1 for the computation of multiple roots of inexact polynomials, using the developed methods given in this thesis.

The feasibility of using structure preserving matrix methods in computing the multiple roots is then discussed in Chapter 10. Future work that may extend the work

presented in this thesis is also discussed in this chapter.

## 2.5 Summary

This chapter has introduced the concept of ill-conditioned polynomials and provided a study of the sensitivity of a multiple root of a polynomial. The theory of pejorative manifolds enables the behavior of a multiple root in the presence of noise to be understood. It has been shown that a multiple root is ill-conditioned with respect to random perturbations that change its multiplicity. On the other hand, it is well-conditioned with respect to the structured perturbations that preserve its multiplicity. A root solver that utilises these observations has been presented along with its geometrical interpretation. The stages required by the developed root solver have been described.

# Chapter 3

# Sylvester resultant matrix

The polynomial root finder described in Algorithm 2.3.1 forms the high level description of the root finder proposed in this thesis. The implementation of this algorithm requires successive GCDs to be computed. This chapter describes the application of the Sylvester resultant matrix and its subresultant matrices for the computation of the GCD of two univariate polynomials expressed in the power basis.

The existence of a non-constant GCD of two polynomials can be verified by testing the singularity of their Sylvester resultant matrix. In particular, two polynomials have a non-constant GCD if and only if their Sylvester resultant matrix is singular. Moreover, if this resultant matrix is singular, then it is rank deficient and the deficiency in its rank equals the degree of the GCD, and the coefficients of the GCD lie in last non-zero row of this resultant matrix, after reducing it into an upper triangular form [5]. Thus, the Sylvester resultant matrix is closely related to the GCD computation.

Several resultant matrices can be used in the GCD computations, including the Sylvester, Bézout and companion matrices, as they have the same GCD information,

that is, the rank deficiency is equal to the degree of the GCD and the coefficients of the GCD can be found from the resultant matrix. The Sylvester resultant matrix has been chosen in this work due to its linear structure which simplifies the implementation of the structured computation methods that are required for the polynomial root solver developed proposed in this thesis.

This chapter first defines the Sylvester matrix and reviews some of its properties for the GCD computation in Section 3.1. Then, Section 3.2 introduces the Sylvester subresultant matrices and their importance for the computation of the degree of the GCD.

## 3.1 Sylvester resultant matrix

To define the Sylvester resultant matrix, let us first decide when a pair of polynomials, $\hat{f} = \hat{f}(y)$ and $\hat{g} = \hat{g}(y)$, has a non-constant common divisor. Let

$$\hat{f}(y) = \sum_{i=0}^{m} \hat{a}_i y^i \qquad \text{and} \qquad \hat{g}(y) = \sum_{i=0}^{n} \hat{b}_i y^i, \qquad \hat{a}_m, \hat{b}_n \neq 0. \tag{3.1}$$

If $\hat{f}(y)$ and $\hat{g}(y)$ have a non-constant common divisor, then there must exist a value of $y$ for which $\hat{f}(y) = 0$ and $\hat{g}(y) = 0$, simultaneously. Using these equations, construct a system of $N = m+n$ homogeneous equations in $N$ unknowns. The coefficient matrix of this system is called the *Sylvester resultant matrix*. In particular, multiplying

$\hat{f}(y) = 0$ by $y^{n-1}, y^{n-2}, \cdots, y, 1$, respectively, yields the $n$ equations

$$\begin{array}{ccccccccc}
\hat{a}_m y^{m+n-1} & + & \hat{a}_{m-1} y^{m+n-2} & + & \cdots & + & \hat{a}_0 y^{n-1} & & = 0 \\
 & & \hat{a}_m y^{m+n-2} & + & \cdots & + & \hat{a}_1 y^{n-1} & + \hat{a}_0 y^{n-2} & = 0 \\
 & & \vdots & & & & & & \vdots \\
 & & \hat{a}_m y^m & + & \hat{a}_{m-1} y^{m-1} & + & \cdots & + \hat{a}_0 & = 0
\end{array} \tag{3.2}$$

Similarly, multiplying $\hat{g}(y) = 0$ by $y^{m-1}, y^{m-2}, \cdots, y, 1$, respectively, yields the $m$ equations

$$\begin{array}{ccccccccc}
\hat{b}_n y^{m+n-1} & + & \hat{b}_{n-1} y^{m+n-2} & + & \cdots & + & \hat{b}_0 y^{m-1} & & = 0 \\
 & & \hat{b}_n y^{m+n-2} & + & \cdots & + & \hat{b}_1 y^{m-1} & + \hat{b}_0 y^{m-2} & = 0 \\
 & & \vdots & & & & & & \vdots \\
 & & \hat{b}_n y^n & + & \hat{b}_{n-1} y^{n-1} & + & \cdots & + \hat{b}_0 & = 0
\end{array} \tag{3.3}$$

The transpose of the $(m+n)$ equations in (3.2) and (3.3) can be written as a system of linear homogeneous equations

$$\left[\begin{array}{ccccc|ccccc} \hat{a}_m & & & & & \hat{b}_n & & & & \\ \hat{a}_{m-1} & \hat{a}_m & & & & \hat{b}_{n-1} & \hat{b}_n & & & \\ \vdots & \hat{a}_{m-1} & \ddots & & & \vdots & \hat{b}_{n-1} & \ddots & & \\ \hat{a}_1 & \vdots & \ddots & \hat{a}_m & & \hat{b}_1 & \vdots & \ddots & \hat{b}_n & \\ \hat{a}_0 & \hat{a}_1 & \ddots & \hat{a}_{m-1} & & \hat{b}_0 & \hat{b}_1 & \ddots & \hat{b}_{n-1} & \\ & \hat{a}_0 & \ddots & \vdots & & & \hat{b}_0 & \ddots & \vdots & \\ & & \ddots & \hat{a}_1 & & & & \ddots & \hat{b}_1 & \\ & & & \hat{a}_0 & & & & & \hat{b}_0 & \end{array}\right] \left[\begin{array}{c} y^{m+n-1} \\ y^{m+n-2} \\ \vdots \\ y \\ 1 \end{array}\right] = 0. \qquad (3.4)$$

Thus the Sylvester resultant matrix (which will henceforth be called the Sylvester matrix, for simplicity) of $f(y)$ and $\hat{g}(y)$ defined above is the $(m+n) \times (m+n)$

coefficient matrix,

$$S(\hat{f},\hat{g}) = \left[\begin{array}{cccc|cccc} \hat{a}_m & & & & \hat{b}_n & & & \\ \hat{a}_{m-1} & \hat{a}_m & & & \hat{b}_{n-1} & \hat{b}_n & & \\ \vdots & \hat{a}_{m-1} & \ddots & & \vdots & \hat{b}_{n-1} & \ddots & \\ \hat{a}_1 & \vdots & \ddots & \hat{a}_m & \hat{b}_1 & \vdots & \ddots & \hat{b}_n \\ \hat{a}_0 & \hat{a}_1 & \ddots & \hat{a}_{m-1} & \hat{b}_0 & \hat{b}_1 & \ddots & \hat{b}_{n-1} \\ & \hat{a}_0 & \ddots & \vdots & & \hat{b}_0 & \ddots & \vdots \\ & & \ddots & \hat{a}_1 & & & \ddots & \hat{b}_1 \\ & & & \hat{a}_0 & & & & \hat{b}_0 \end{array}\right], \tag{3.4}$$

$n$ columns $m$ columns

where the first $n$ columns contain the coefficients $\hat{a}_i$ of $\hat{f}(y)$ and the last $m$ columns contain the coefficients $\hat{b}_i$ of $\hat{g}(y)$. This is how the Sylvester matrix is defined in [5]. The Sylvester matrix can also be viewed as two Cauchy matrices. These Cauchy matrices are formed by the first $n$ columns and the last $m$ columns, respectively, of $S(\hat{f},\hat{g})$. Thus the Sylvester matrix can be represented as follows

$$S(\hat{f},\hat{g}) = \left[\begin{array}{cc} C(\hat{f}) & D(\hat{g}) \end{array}\right], \tag{3.5}$$

where $C(\hat{f}) \in \mathbb{R}^{(m+n)\times n}$, $D(\hat{g}) \in \mathbb{R}^{(m+n)\times m}$. The representation of the Sylvester matrix in terms of two Cauchy matrices will be used in the following chapters where it is shown that the vector of coefficients of the product of two polynomials can be

written as a matrix-vector product.

If $\hat{f}(y)$ and $\hat{g}(y)$ have a non-constant GCD then the homogeneous system in (3.4) must have a non-trivial solution. In general, a system $S\mathbf{y} = 0$ has a non-trivial solution if and only if $S$ is singular. Therefore, a necessary condition for $\hat{f}(y)$ and $\hat{g}(y)$ to have a non-constant GCD is that their Sylvester matrix be singular.

Theorem 3.1 establishes the relation between the Sylvester matrix and the GCD computation.

**Theorem 3.1.** *Let the polynomials $\hat{f}(y)$ and $\hat{g}(y)$ in (3.1) have a non-constant GCD. If the degree of the GCD is $\hat{d} > 0$, then the following properties of their Sylvester matrix, $S(\hat{f}, \hat{g})$, hold true:*

1. *$S(\hat{f}, \hat{g})$ is rank deficient and therefore,*

$$\det(S(\hat{f}, \hat{g})) = 0.$$

2. *The degree of the* GCD *of $\hat{f}(y)$ and $\hat{g}(y)$ equals the rank loss of $S(\hat{f}, \hat{g})$,*

$$\deg\left(\mathrm{GCD}\left(\hat{f}, \hat{g}\right)\right) = m + n - \mathrm{rank}\left(S(\hat{f}, \hat{g})\right).$$

3. *The coefficients of the* GCD *of $\hat{f}(y)$ and $\hat{g}(y)$ lie in the last non-vanishing row of $S(\hat{f}, \hat{g})^T$, after reducing it into an upper triangular form.*

These results are established in [5], pages 35-39, and [18]. The relation between the GCD of two polynomials and their Sylvester matrix can be clarified by the following example.

**Example 3.1.** Let the polynomials $\hat{f}(y)$ and $\hat{g}(y)$, respectively be

$$\hat{f}(y) = y^3 + 8y^2 + 5y - 50 = (y+5)^2(y-2),$$

and

$$\hat{g}(y) = y^5 + 7y^4 - 2y^3 - 46y^2 + 65y - 25 = (y-1)^3(y+5)^2,$$

whose GCD is

$$\hat{d}(y) = y^3 + 10y^2 + 25y = (y+5)^2.$$

Since $\deg(\hat{f}) = 3$ and $\deg(\hat{g}) = 5$, their Sylvester matrix which is $8 \times 8$ is

$$S(\hat{f},\hat{g}) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 8 & 1 & 0 & 0 & 0 & 7 & 1 & 0 \\ 5 & 8 & 1 & 0 & 0 & -2 & 7 & 1 \\ -50 & 5 & 8 & 1 & 0 & -46 & -2 & 7 \\ 0 & -50 & 5 & 8 & 1 & 65 & -46 & -2 \\ 0 & 0 & -50 & 5 & 8 & -25 & 65 & -46 \\ 0 & 0 & 0 & -50 & 5 & 0 & -25 & 65 \\ 0 & 0 & 0 & 0 & -50 & 0 & 0 & -25 \end{bmatrix}.$$

It can be verified that $\det(S(\hat{f},\hat{g})) = 0$ and, using the Sylvester matrix properties discussed above, this correctly suggests that the polynomials $\hat{f}(y)$ and $\hat{g}(y)$ have a

non-constant GCD. Furthermore, reducing $S(\hat{f}, \hat{g})^T$ into an upper triangular form, yields the matrix

$$\begin{bmatrix} -0.5 & -4 & -2.5 & 25 & 0 & 0 & 0 & 0 \\ 0 & -0.5 & -4 & -2.5 & 25 & 0 & 0 & 0 \\ 0 & 0 & -0.5 & -4 & -2.5 & 25 & 0 & 0 \\ 0 & 0 & 0 & -0.5 & -4 & -2.5 & 25 & 0 \\ 0 & 0 & 0 & 0 & -1 & -10 & -25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 10 & 25 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

whose rank is 6. Applying the second property yields,

$$\deg(\hat{d}(y)) = (m+n) - \text{rank}(S(\hat{f}, \hat{g})) = 2,$$

which is equal to the $\deg(\hat{d}(y))$. Finally, the last non-vanishing row provides the coefficients $1, 10$, and $25$, which define the coefficients of $\hat{d}(y)$, as required. □

In this section it is shown how the Sylvester matrix allows the computation of the degree and the coefficients of the GCD of two polynomials. In addition to the Sylvester matrix properties mentioned here, its subresultant matrices provide a means to compute the degree of the GCD of two polynomials. The next section considers these subresultant matrices and explains their relation to the degree of the GCD.

## 3.2 Sylvester subresultant matrices

The $k^{th}$ Sylvester subresultant matrix of the polynomials $\hat{f}(y)$ and $\hat{g}(y)$, $S_k(\hat{f},\hat{g}) \in \mathbb{R}^{(m+n-k+1)\times(m+n-2k+2)}$ for $1 \leq k \leq \min(m,n)$, is formed by deleting some rows and columns of $S(\hat{f},\hat{g})$. It is recalled that $S(\hat{f},\hat{g})$ can be represented in terms of two Cauchy matrices as shown in (3.5), and the $k^{th}$ subresultant matrix is formed by deleting the last $k-1$ columns of $C(\hat{f})$, the last $k-1$ columns of $D(\hat{g})$, and the last $k-1$ rows of $S(\hat{f},\hat{g})$. For $k=1$, the Sylvester subresultant matrix reduces to the Sylvester matrix.

**Example 3.2.** Let

$$\begin{aligned} \hat{f}(y) &= 5y^5+3y^4-4y^3+y^2-y+6, \\ \hat{g}(y) &= 7y^3-3y^2-2y-9. \end{aligned}$$

Then

$$S_1 = S(\hat{f},\hat{g}) = \begin{bmatrix} 5 & 0 & 0 & 7 & 0 & 0 & 0 & 0 \\ 3 & 5 & 0 & -3 & 7 & 0 & 0 & 0 \\ -4 & 3 & 5 & -2 & -3 & 7 & 0 & 0 \\ 1 & -4 & 3 & 9 & -2 & -3 & 7 & 0 \\ -1 & 1 & -4 & 0 & 9 & -2 & -3 & 7 \\ 6 & -1 & 1 & 0 & 0 & 9 & -2 & -3 \\ 0 & 6 & -1 & 0 & 0 & 0 & 9 & -2 \\ 0 & 0 & 6 & 0 & 0 & 0 & 0 & 9 \end{bmatrix},$$

$$S_2(\hat{f},\hat{g}) = \begin{bmatrix} 5 & 0 & 7 & 0 & 0 & 0 \\ 3 & 5 & -3 & 7 & 0 & 0 \\ -4 & 3 & -2 & -3 & 7 & 0 \\ 1 & -4 & 9 & -2 & -3 & 7 \\ -1 & 1 & 0 & 9 & -2 & -3 \\ 6 & -1 & 0 & 0 & 9 & -2 \\ 0 & 6 & 0 & 0 & 0 & 9 \end{bmatrix}, \quad S_3(\hat{f},\hat{g}) = \begin{bmatrix} 5 & 7 & 0 & 0 \\ 3 & -3 & 7 & 0 \\ -4 & -2 & -3 & 7 \\ 1 & 9 & -2 & -3 \\ -1 & 0 & 9 & -2 \\ 6 & 0 & 0 & 9 \end{bmatrix}.$$

□

Consider the two polynomials, $\hat{f}(y)$ and $\hat{g}(y)$ defined in (3.1), and let these two polynomials be non-coprime polynomials with a GCD of degree $\hat{d}$. Factorising these two polynomials for $k = 1, \ldots, \hat{d}$, yields

$$\hat{f}(y) = \hat{u}_k(y)\hat{d}_k(y), \tag{3.6}$$

and

$$\hat{g}(y) = \hat{v}_k(y)\hat{d}_k(y), \tag{3.7}$$

where

$$\hat{u}_k(y) = \sum_{i=0}^{m-k} \hat{u}_{k,i} y^{m-k-i}, \qquad \text{and} \qquad \hat{v}_k(y) = \sum_{i=0}^{n-k} \hat{v}_{k,i} y^{n-k-i},$$

are two quotient polynomials of degree $m-k$ and $n-k$ respectively, and

$$\hat{d}_k(y) = \sum_{i=0}^{k} \hat{d}_{k,i} y^{k-i},$$

is a common divisor polynomial of degree $k$. Note that for $k = 1, \cdots, d-1$, the polynomials $\hat{u}_k(y)$ and $\hat{v}_k(y)$ are not co-prime, but for $k = \hat{d}$ they are co-prime polynomials. It follows that

$$\hat{v}_k(y)\hat{f}(y) = \hat{u}_k(y)\hat{g}(y) \quad \Longleftrightarrow \quad \hat{d}_k(y) = \frac{\hat{f}(y)}{\hat{u}_k(y)} = \frac{\hat{g}(y)}{\hat{v}_k(y)}. \tag{3.8}$$

The polynomial products in (3.8) can be written in a matrix-vector product as follows

$$\begin{bmatrix} C_k & D_k \end{bmatrix} \begin{bmatrix} \hat{\mathbf{v}}_k \\ -\hat{\mathbf{u}}_k \end{bmatrix} = S_k \begin{bmatrix} \hat{\mathbf{v}}_k \\ -\hat{\mathbf{u}}_k \end{bmatrix} = 0, \qquad k = 1, \ldots, \hat{d}, \tag{3.9}$$

where $C_k = C_k(\hat{f})$, $D_k = D_k(\hat{g})$, are Cauchy matrices of the polynomials $\hat{f}(y)$ and $\hat{g}(y)$ respectively, $S_k = S_k(\hat{f}, \hat{g}) \in \mathbb{R}^{(m+n-k+1)\times(m+n-2k+2)}$ is the $k$th subresultant matrix, and

$$\begin{aligned} \hat{\mathbf{u}}_k &= \begin{bmatrix} \hat{u}_{k,0} & \hat{u}_{k,1}, & \cdots & \hat{u}_{k,m-k} \end{bmatrix}^T \in \mathbb{R}^{m-k+1}, \\ \hat{\mathbf{v}}_k &= \begin{bmatrix} \hat{v}_{k,0} & \hat{v}_{k,1}, & \cdots & \hat{v}_{k,n-k} \end{bmatrix}^T \in \mathbb{R}^{n-k+1}. \end{aligned}$$

Since the degree of the GCD of $\hat{f}(y)$ and $\hat{g}(y)$ is $\hat{d}$, it follows that $\hat{d}_{k,0} \neq 0$, and thus $\hat{u}_{k,0}, \hat{v}_{k,0} \neq 0$. Also, it is clear that $\hat{f}(y)$ and $\hat{g}(y)$ possess common divisors of degrees

$1, \cdots, \hat{d}$, but they do not possess a common divisor of degrees $\hat{d}+1$ or more, and thus

$$\begin{aligned} \operatorname{rank} S_k(\hat{f}, \hat{g}) &< m+n-2k+2, \qquad k = 1, \ldots, \hat{d} \\ \operatorname{rank} S_k(\hat{f}, \hat{g}) &= m+n-2k+2, \qquad k = \hat{d}+1, \ldots, \min(m,n). \end{aligned} \tag{3.10}$$

This implies that the degree $\hat{d}$ of the GCD of $\hat{f}(y)$ and $\hat{g}(y)$ equals the largest value of $k$ for which $S_k(\hat{f}, \hat{g})$ is rank deficient. This clearly shows how the computation of the GCD degree reduces to a rank determination problem. Furthermore, the result above implies that, for the homogeneous equation in (3.9), if

$$S_k(\hat{f}, \hat{g}) = \begin{bmatrix} c_k & A_k \end{bmatrix},$$

where $c_k$ is the first column of $S_k(\hat{f}, \hat{g})$ and $A_k$ is the matrix formed from the remaining columns of $S_k(\hat{f}, \hat{g})$, the following linear algebraic equation possesses solutions only for $k = 1, \cdots, \hat{d}$,

$$A_k x_k = c_k, \tag{3.11}$$

where

$$x_k = \begin{bmatrix} \hat{v}_{k,1} & \cdots & \hat{v}_{k,n-k} & -\hat{u}_{k,0} & \cdots & -\hat{u}_{k,m-k} \end{bmatrix}^T \in \mathbb{R}^{m+n-2k+1}. \tag{3.12}$$

These results are also established in [17, 28]. Chapter 6 shows how these results can be used to introduce new methods for the computation of the degree of the GCD, which is one of the main building blocks of the proposed root solver.

**Example 3.3.** Let the polynomials $\hat{f}(y)$ and $\hat{g}(y)$, respectively be

$$\hat{f}(y) = (y+1)^4(y-2)^2 = y^6 - 6y^4 - 4y^3 + 9y^2 + 12y + 4,$$

and

$$\hat{g}(y) = (y-2)^4 = y^4 - 8y^3 + 24y^2 - 32y + 16,$$

whose GCD is of degree 2, and their orders, respectively, are $m = 6$ and $n = 4$. Then the row echelon form of the subresultant matrices, for $k = 1, \cdots, \min(m, n)$ are

$$S_1^{\text{re}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -4 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -4 & 17 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 4 & -20 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & 4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & -4 & 15 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -6 & 20 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -4 & 10 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$S_2^{\text{re}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -4 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -4 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad S_3^{\text{re}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

and

$$S_4^{\text{re}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}.$$

The results of computing the rank of the subresultant matrices for $k = 1, \cdots, 4$ are shown in Table 3.1. These results imply that $S_k(\hat{f}, \hat{g})$ is rank deficient only for $k \leq 2$, and thus it follows from (3.10) that deg GCD $(\hat{f}, \hat{g}) = 2$. □

An important issue that should be addressed is the nature of the solution $x_k$ in (3.12), for $k = 1, \cdots, \min(m, n)$, from which the estimates $\hat{d}_k(y)$ are calculated. In

Table 3.1: The ranks and dimensions of $S_k, k = 1, \cdots, 4$ for Example 3.3.

| $S_k$ | Rank | $m+n-2k+2$ |
|---|---|---|
| $S_1$ | 8 | 10 |
| $S_2$ | 7 | 8 |
| $S_3$ | 6 | 6 |
| $S_4$ | 4 | 4 |

particular, estimates for the vectors $\hat{\mathbf{u}}_k$ and $\hat{\mathbf{v}}_k$ can be calculated from (3.11), and estimates for $\hat{\mathbf{d}}_k$ can be obtained from $\hat{\mathbf{u}}_k$ and $\hat{\mathbf{v}}_k$. In particular, (3.6) and (3.7) can be combined into one matrix-vector equation,

$$\begin{bmatrix} Q_{k,1} \\ Q_{k,2} \end{bmatrix} \hat{\mathbf{d}}_k = \begin{bmatrix} \hat{\mathbf{f}} \\ \hat{\mathbf{g}} \end{bmatrix} \quad k = 1, \cdots, \min(m,n),$$

where $Q_{k,1}$ and $Q_{k,2}$ are Cauchy matrices whose entries are the coefficients of $\hat{\mathbf{u}}_k$ and $\hat{\mathbf{v}}_k$ respectively, that are calculated from (3.12), and $\hat{\mathbf{f}}$ and $\hat{\mathbf{g}}$ are the vectors of the coefficients of $\hat{f}(y)$ and $\hat{g}(y)$, respectively. Thus, $\hat{\mathbf{d}}_k$ can be obtained from,

$$\hat{\mathbf{d}}_k = \begin{bmatrix} Q_{k,1} \\ Q_{k,2} \end{bmatrix}^{\dagger} \begin{bmatrix} \hat{\mathbf{f}} \\ \hat{\mathbf{g}} \end{bmatrix} \quad k = 1, \cdots, \min(m,n).$$

Equation (3.11) possesses solutions for $k = 1, \cdots, \hat{d}$, but it does not possess a solution for $k > \hat{d}$. More specifically, it follows from (3.10) that the solutions of (3.11) satisfy the following conditions:

1. For $k = 1, \cdots, \hat{d} - 1$, $\operatorname{rank} A_k < m + n - 2k + 1$, and thus for each of these values of $k$ there is an infinite number of solutions. Only a finite number yields the coefficients of a polynomial $\hat{d}_k(y)$. All the other solutions yield rational

functions, which are not of interest.

2. For $k = \hat{d}$, $\operatorname{rank} A_k = m+n-2k+1$, and thus there is one unique solution $\hat{d}_k(y)$. This solution must be the coefficients of a polynomial not a rational function, and thus $d_d(y)$ is also a polynomial.

3. For $k = \hat{d}+1, \ldots, \min(m,n)$, $\operatorname{rank} A_k = m+n-2k+1$, and thus there is no solution.

**Example 3.4.** Consider the exact polynomials

$$\begin{aligned} \hat{f}(y) &= (y-2)^2(y-4)(y-6), \\ \hat{g}(y) &= (y-2)(y-4)^2, \end{aligned}$$

whose GCD is of degree 2. The first Sylvester subresultant matrix $S_1(\hat{f},\hat{g})$ of these polynomials is

$$S_1(\hat{f},\hat{g}) = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ -14 & 1 & 0 & -10 & 1 & 0 & 0 \\ 68 & -14 & 1 & 32 & -10 & 1 & 0 \\ -136 & 68 & -14 & -32 & 32 & -10 & 1 \\ 96 & -136 & 68 & 0 & -32 & 32 & -10 \\ 0 & 96 & -136 & 0 & 0 & -32 & 32 \\ 0 & 0 & 96 & 0 & 0 & 0 & -32 \end{bmatrix},$$

whose null space has the following family of vectors

$$\begin{bmatrix} \hat{\mathbf{v}} \\ -\hat{\mathbf{u}} \end{bmatrix} = \begin{bmatrix} \hat{v}_{1,0} \\ \hat{v}_{1,1} \\ \hat{v}_{1,2} \\ -\hat{u}_{1,0} \\ -\hat{u}_{1,1} \\ -\hat{u}_2 \\ -\hat{u}_3 \end{bmatrix} = \begin{bmatrix} \hat{v}_{1,0} \\ \hat{v}_{1,1} \\ -4(4\hat{v}_{1,0} + \hat{v}_{1,0}) \\ -\hat{v}_{1,0} \\ -\frac{8}{3}(5\hat{v}_{1,0} + 2\hat{v}_{1,1}) \\ 4(5\hat{v}_{1,0} + 2\hat{v}_{1,1}) \\ 12(4\hat{v}_{1,0} + \hat{v}_{1,1}) \end{bmatrix}.$$

It follows from (3.8) that

$$\hat{d}_1(y) = \frac{\hat{f}(y)}{\hat{u}_1(y)} = \frac{\hat{g}(y)}{\hat{v}_1(y)} = \frac{(y-2)(y-4)}{\hat{v}_{1,0}y + \hat{v}_{1,1} - 4(4\hat{v}_{1,0} + \hat{v}_{1,1})}, \tag{3.13}$$

where $\hat{v}_{1,0}$ and $\hat{v}_{1,1}$ are arbitrary constants that are not simultaneously zero. Clearly, it can be seen that $\hat{d}_1(y)$ is in general a rational function that has an infinite number of forms. However, only those forms for which $\hat{d}_1(y)$ is a polynomial are of interest, and for this example, there are three forms of this type. In particular, (3.13) is proportional to the common divisors $(y-2)$ and $(y-4)$, for $\hat{v}_{1,1} = -4\hat{v}_{1,0}$ and $\hat{v}_{1,1} = -\frac{14}{3}\hat{v}_{1,0}$, respectively, and it is equal to the GCD of $\hat{f}(y)$ and $\hat{g}(y)$ for $\hat{v}_{1,0} = 0, \hat{v}_{1,1} \neq 0$. All other values of $\frac{\hat{v}_{1,1}}{\hat{v}_{1,0}}$ are not of interest as they yield rational forms of $\hat{d}_1(y)$.

Equation (3.11) restricts the solutions to be from a subspace of the null space of the Sylvester matrix $S_1(\hat{f}, \hat{g})$ of $\hat{f}(y)$ and $\hat{g}(y)$, by forcing $\hat{v}_{1,0}$ to be equal to $-1$. It can be verified that this restriction allows the common divisors $(y-2)$ and $(y-4)$ to be recovered from $\hat{v}_{1,1} = (-1) \times (-4) = 4$ and $\hat{v}_{1,1} = (-1) \times (-\frac{14}{3}) = \frac{14}{3}$, respectively, but it can not recover the GCD of $\hat{f}(y)$ and $\hat{g}(y)$, because the condition $v_{1,0} = -1$

contradicts the condition $v_{1,0} = 0$ required for the GCD to be recovered. However, the GCD can be easily obtained from $A_2 x_2 = c_2$, whose solution is unique in this example, as is now shown.

Considering $k = 2$, (3.11) becomes

$$\begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & -10 & 1 & 0 \\ -14 & 32 & -10 & 1 \\ 68 & -32 & 32 & -10 \\ -136 & 0 & -32 & 32 \\ 96 & 0 & 0 & -32 \end{bmatrix} \begin{bmatrix} \hat{v}_1 \\ -\hat{u}_0 \\ -\hat{u}_1 \\ -\hat{u}_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -14 \\ 68 \\ -136 \\ 96 \\ 0 \end{bmatrix},$$

which has the unique solution

$$\hat{v}_1 = 4, \quad \hat{u}_0 = -1, \quad \hat{u}_1 = 8, \quad \hat{u}_2 = -12,$$

and since $\hat{v}_0 = -1$, it follows that $\hat{u}(y)$ and $\hat{v}(y)$ are

$$\hat{u}(y) = -(y-2)(y-6) = -y^2 + 8y - 12 \qquad \text{and} \qquad \hat{v}(y) = -(y-4),$$

which are co-prime. Therefore,

$$\hat{d}_2(y) = -(y-2)(y-4),$$

is the common divisor and it is the unique solution. Finally, when $k = 3$, (3.11)

becomes

$$\begin{bmatrix} 1 & 0 \\ -10 & 1 \\ 32 & -10 \\ -32 & 32 \\ 0 & -32 \end{bmatrix} \begin{bmatrix} \hat{v}_1 \\ -\hat{u}_0 \end{bmatrix} = \begin{bmatrix} 1 \\ -14 \\ 68 \\ -136 \\ 96 \end{bmatrix},$$

which does not possess a solution. Thus the polynomials $\hat{f}(y)$ and $\hat{g}(y)$ do not have a common divisor of degree $k = 3$, and thus the degree of the GCD of $\hat{f}(y)$ and $\hat{g}(y)$ is equal to two. □

It is shown in Example 3.4, that for $k = 1, \cdots, \hat{d} - 1$, (3.8) is satisfied by an infinite number of solutions $\hat{\mathbf{u}}_k$ and $\hat{\mathbf{v}}_k$ of (3.11). However, since the common divisors are polynomials, interest is restricted to a finite number of solutions for which $\hat{d}_k(y)$ are polynomials, not rational functions. For $k = \hat{d}$, (3.8) is satisfied by the unique solution of (3.11), and finally, (3.11) does not possess solutions for $k > \hat{d}$ because there does not exist a common divisor of degree greater than $\hat{d}$.

It has been shown in this chapter that the Sylvester resultant matrix and its subresultant matrices allow the computations of both the degree of the GCD and its coefficients. Although this result is valid theoretically, it is more involved computationally if inexact polynomials are considered. More precisely, these computations fail in practice as data is usually corrupted by noise, through which the exact non co-prime polynomials are contaminated and become, with high probability, co-prime. This also implies that the corresponding Sylvester matrix is non-singular and all the properties in Theorem 3.1 will not be applicable.

A numerical solution for this problem is to slightly perturb the given polynomials by structured perturbations so that they have a non-trivial GCD. In terms of the Sylvester matrix of the given inexact polynomials, this full rank matrix is perturbed such that it becomes rank deficient. This is referred to as the structured low rank approximation of the Sylvester matrix, which yields an AGCD because the polynomials have been moved slightly to have a non-constant GCD.

The problem of computing an AGCD of two inexact polynomials is addressed in detail in Chapters 6 and 7, where the two main stages of the AGCD computation, the computation of its degree and its coefficients, are considered, respectively. A crucial step that has to be applied before considering the numerical computation of an AGCD of imperfectly known polynomials is to preprocess them. This preprocessing will be considered in the next chapter and it will be shown that their inclusion is vital to obtain good result.

## 3.3 Summary

This chapter has described the properties of the Sylvester matrix and its use in computing the GCD of two non co-prime polynomials. It was shown how the singularity of the Sylvester matrix provides a means not only for detecting the existence of the GCD, but also for computing its degree and coefficients. Important results for the GCD computation were established. For example, the degree of the GCD is equal to the rank loss of the Sylvester matrix and the coefficients of the GCD are defined by the last non-zero row of the transpose of the Sylvester matrix after reducing it to an upper triangular form. Moreover, it was shown that the order of the Sylvester subresultant matrices is also important when it is required to compute the degree of

the GCD.

# Chapter 4

# Preprocessing operations

This chapter considers the preprocessing operations that must be performed on the given polynomial before being involved in the computation of its roots. In particular, it was noted in the previous chapter that the implementation of Algorithm 2.3.1, whose computational implementation is considered in this thesis, requires successive GCD computations. It was also shown that the Sylvester matrix can be used for the theoretical computation of the GCD of two exact polynomials. However the case differs when two inexact polynomials are considered, as they need to be preprocessed before computations are performed on their Sylvester matrix. This is to reduce the possible occurrence of catastrophic problems such as those associated with computations performed on polynomials whose coefficients suffer from a wide variation in magnitude, which occurs frequently in GCD-based polynomial root finders [26].

This chapter considers three preprocessing operations, the first of which normalises the coefficients of the polynomials to have unit magnitude. The other two operations minimise the ratio of the maximum coefficient in magnitude to the minimum coefficient in magnitude, using two parameters $\alpha$ and $\theta$. In particular, let the polynomials

$\hat{f}(y)$ and $\hat{g}(y)$ that are defined in (3.1) denote, respectively, the exact forms of the inexact polynomials $f(y)$ and $g(y)$,

$$f(y) = \sum_{i=0}^{m} a_i y^i \qquad \text{and} \qquad g(y) = \sum_{i=0}^{n} b_i y^i, \qquad a_m, b_n \neq 0. \tag{4.1}$$

The parameter $\alpha$, that arises from the partitioned structure of the Sylvester matrix, and the parameter $\theta$ that is used to scale the independent variable $y$, are introduced in order to transform the normalised forms of the given polynomials $f(y)$ and $g(y)$ into another set of polynomials, whose coefficient variations are smaller. Computational experiments showed that failure to implement this transformation led to a significant degradation in the results. All computations on the developed root solver are therefore performed on this transformed set of polynomials.

## 4.1 Normalisation

It was shown in Chapter 3 that the structure of the $k$th Sylvester matrix $S_k(f, g)$ dedicates the first $n-k+1$ columns for the coefficients of $f(y)$, and the last $m-k+1$ columns for the coefficients of $g(y)$. This partitioned structure may be unbalanced, especially if the coefficients of $f(y)$ are significantly larger or smaller than the coefficients of $g(y)$. This problem can be overcome by normalising both polynomials. Normalising the polynomials by the 2-norm of their coefficients is frequently used in the literature such as [2, 17], but to provide better averaging, the geometric mean (GM) is preferred, especially if the coefficients vary widely in magnitude. Consider

for example, the polynomial $\hat{p}(y)$,

$$\hat{p}(y) = 10y^2 + 10^8 y + 10^4,$$

for which

$$\hat{\mathbf{p}} = [\ 10 \quad 10^8 \quad 10^4\ ].$$

It follows that the geometric mean GM( ), and the 1−, 2− and ∞−norms are

$$\mathrm{GM}(\hat{\mathbf{p}}) = 2.15 \times 10^4, \quad \|\hat{\mathbf{p}}\|_1 \approx \|\hat{\mathbf{p}}\|_2 = 10^8, \quad \text{and} \quad \|\hat{\mathbf{p}}\|_\infty = 10^8,$$

which shows that in contrast to the geometric mean, the 1−, 2− and ∞−norms neglect the small coefficients, that is, they are insensitive to the changes in the small coefficients. For example, if the coefficient of $y^2$ in the polynomial $\hat{p}(y)$ given above, is changed to $10^{-3}$,

$$\hat{q}(y) = 10^{-3}y^2 + 10^8 y + 10^4,$$

whose coefficient vector $\hat{\mathbf{q}}$,

$$\hat{\mathbf{q}} = [\ 10^{-3} \quad 10^8 \quad 10^4\ ],$$

then

$$\mathrm{GM}(\hat{\mathbf{q}}) = 10^3, \quad \|\hat{\mathbf{q}}\|_1 \approx \|\hat{\mathbf{q}}\|_2 = 10^8, \quad \text{and} \quad \|\hat{\mathbf{q}}\|_\infty = 10^8.$$

Clearly, it can be seen that a change of 99.99% in the coefficient of $y^2$ causes a change of 95.4% in the geometric mean of the coefficients, whereas the 1−, 2− and ∞−norms change by a negligible amount. Therefore the geometric mean has been used to redefine the polynomials in (4.1) as follows,

$$f(y) = \sum_{i=0}^{m} \bar{a}_i y^i, \qquad \bar{a}_i = \frac{a_i}{\left(\prod_{j=0}^{m} |a_j|\right)^{\frac{1}{m+1}}}, \tag{4.2}$$

and

$$g(y) = \sum_{i=0}^{n} \bar{b}_i y^i, \qquad \bar{b}_i = \frac{b_i}{\left(\prod_{j=0}^{n} |b_j|\right)^{\frac{1}{n+1}}}, \tag{4.3}$$

where only the non-zero coefficients are considered by the normalisations in (4.2) and (4.3). This normalisation by the geometric mean defines the first preprocessing operation.

## 4.2 Relative scaling of polynomials

If $\alpha$ is a non-zero scalar, then the GCD of $\hat{f}(y)$ and $\hat{g}(y)$, satisfies

$$\text{GCD}(\hat{f}, \hat{g}) \sim \text{GCD}(\hat{f}, \alpha\hat{g}),$$

where $\sim$ denotes equivalence. This equivalence fails numerically when inexact data is considered, because different values of $\alpha$ yield different AGCDs, even if they retain the same degree. The variable $\alpha$ can be used as a parameter to be computed according to a specific criterion such that good results are obtained. Considering the normalised

forms of $f(y)$ and $g(y)$ in (4.2) and (4.3), respectively, $\alpha$ can be interpreted as the weight of $g(y)$ relative to the unit weight of $f(y)$.

Consider the rank of the Sylvester matrix $S(f(y), \alpha g(y))$, where $f(y)$ and $g(y)$ are as defined in (4.2) and (4.3), respectively,

$$\text{rank}\, S(f, \alpha g) = \deg g \quad \text{if} \quad \alpha \quad \text{is very small,}$$

$$\text{rank}\, S(f, \alpha g) = \deg f \quad \text{if} \quad \alpha \quad \text{is very large.}$$

Thus a bad choice of $\alpha$ causes missleading results and an optimal value of $\alpha$ must be computed.

The inclusion of $\alpha$ was first introduced in [78], where it is shown that the rank of $S(f, \alpha g)$ is a function of $\alpha$ and not all the values of $\alpha$ are associated with a well defined rank of this Sylvester matrix. However, computing an optimal value of $\alpha$, to obtain a good approximation for an AGCD is not considered in this reference. The preprocessing operations considered in this thesis require the computation of an optimal value of $\alpha$, and the criterion and method for this computation will be considered after introducing the third preprocessing operation.

## 4.3 Scaling the independent variable

The parameter $\alpha$ introduced in Section 4.2 performs relative scaling of $g(y)$ with respect to the unit weight of $f(y)$. The preprocessing operation discussed in this section introduces the parameter $\theta$ that scales the independent variable $y$, using the

substitution

$$y = \theta w, \tag{4.4}$$

where $w$ is the new independent variable and $\theta$ is a parameter that is chosen to minimise the ratio of the maximum coefficient in magnitude to the minimum coefficient in magnitude. This substitution does not change the degree of the GCD of $\hat{f}(y)$ and $\hat{g}(y)$, that is,

$$\text{GCD}(\hat{f}(y), \hat{g}(y)) = \text{GCD}(\hat{f}(\theta w), \hat{g}(\theta w)).$$

Considering the effect of the scaling in (4.4), it is proved in [80], that the componentwise condition number of a real root $y_0$ of $f(y)$ is equal to the componentwise condition number of the real root $w_0 = y_0/\theta$ of $f(\theta w)$, and this also applies to the polynomial $g(y)$.

The method used to calculate optimal values $\alpha_0$ and $\theta_0$ of $\alpha$ and $\theta$, respectively, is considered in the next section. Scaling a polynomial by this factor has also been considered in [21] and [26] for one polynomial, and it is extended in this thesis to two polynomials.

## 4.4 Calculating optimal values of the scaling parameters

Polynomials whose coefficients suffer from wide variations in their magnitude create numerical problems in several applications. One of those applications where such

problems occur frequently is the GCD-based polynomial root finder [26]. Thus a suitable criterion for computing optimal values of $\alpha$ and $\theta$ is: *Minimise the variation in the magnitude of the coefficients of the polynomials whose GCD is to be computed.* Consider the normalised polynomials $f(y)$ and $g(y)$ that are defined in (4.2) and (4.3), respectively. Using the substitution in (4.4) they are redefined as,

$$f_\theta(w) = \sum_{i=0}^{m} (\bar{a}_i \theta^{m-i}) w^{m-i}, \tag{4.5}$$

$$g_\theta(w) = \sum_{j=0}^{n} (\bar{b}_j \theta^{n-j}) w^{n-j}. \tag{4.6}$$

The combination of the discussion in Section 4.2, and (4.5) and (4.6), shows that the Sylvester matrix $S(f_\theta, \alpha g_\theta)$ must be considered for the AGCD computations. The entries of this Sylvester matrix are $\{\bar{a}_i \theta^{m-i}\}_{i=0}^{m}$ and $\{\alpha \bar{b}_j \theta^{n-j}\}_{j=0}^{n}$ and thus the optimal values of $\theta$ and $\alpha$ are chosen such that the ratio of the maximum coefficient (in magnitude), to the minimum coefficient (in magnitude), of the polynomials $f_\theta(w)$ and $\alpha g_\theta(w)$ is minmised, that is

$$\alpha_0, \theta_0 = \arg\min_{\alpha,\theta} \left\{ \frac{\max\left\{\max_{i=0,\ldots,m} \left|\bar{a}_i \theta^{m-i}\right|, \max_{j=0,\ldots,n} \left|\alpha \bar{b}_j \theta^{n-j}\right|\right\}}{\min\left\{\min_{i=0,\ldots,m} \left|\bar{a}_i \theta^{m-i}\right|, \min_{j=0,\ldots,n} \left|\alpha \bar{b}_j \theta^{n-j}\right|\right\}} \right\}. \tag{4.7}$$

This minmisation can be rewritten as:

Minimise $\frac{t}{d}$

Subject to

$$
\begin{aligned}
t &\geq |\bar{a}_i|\,\theta^{m-i}, \qquad i = 0,\ldots,m \\
t &\geq \alpha\,|\bar{b}_j|\,\theta^{n-j}, \quad j = 0,\ldots,n \\
d &\leq |\bar{a}_i|\,\theta^{m-i}, \qquad i = 0,\ldots,m \\
d &\leq \alpha\,|\bar{b}_j|\,\theta^{n-j}, \quad j = 0,\ldots,n \\
d &> 0 \\
\theta &> 0 \\
\alpha &> 0,
\end{aligned}
$$

where $\bar{a}_i \neq 0, i = 0,\ldots,m$, and $\bar{b}_j \neq 0, j = 0,\ldots,n$.

Using the transformations

$$
T = \log t, \quad D = \log d, \quad \phi = \log\theta, \quad \mu = \log\alpha, \quad \bar{\alpha}_i = \log|\bar{a}_i|, \quad \bar{\beta}_j = \log|\bar{b}_j|,
$$

the minimisation problem can be restated as:

Minimise $T - D$

Subject to

$$
\begin{aligned}
T &- (m-i)\phi &&\geq \bar{\alpha}_i, && i = 0,\ldots,m \\
T &- (n-j)\phi - \mu &&\geq \bar{\beta}_j, && j = 0,\ldots,n \\
-D &+ (m-i)\phi &&\geq -\bar{\alpha}_i, && i = 0,\ldots,m \\
-D &+ (n-j)\phi + \mu &&\geq -\bar{\beta}_j, && j = 0,\ldots,n,
\end{aligned}
$$

which is a standard linear programming (LP) problem whose objective function is

$$T - D = \begin{bmatrix} 1 & -1 & 0 & 0 \end{bmatrix} \begin{bmatrix} T \\ D \\ \phi \\ \mu \end{bmatrix}.$$

There are $2(m+n+2)$ constraints in this LP problem, and as noted above, if a coefficient $a_i$ or $b_j$ is equal to zero, then the corresponding constraints are deleted. To recover the optimal values $\alpha_0$ and $\theta_0$ of $\alpha$ and $\theta$, respectively, recall that $\phi = \log\theta$ and $\mu = \log\alpha$. The solutions $\alpha_0$ and $\theta_0$ of (4.7) are the optimal values of $\alpha$ and $\theta$, respectively, and thus the polynomials (4.5) and (4.6) become

$$f_{\theta_0}(w) = \sum_{i=0}^{m} (\bar{a}_i \theta_0^{m-i}) w^{m-i}, \tag{4.8}$$

$$g_{\theta_0}(w) = \sum_{j=0}^{n} (\bar{b}_j \theta_0^{n-j}) w^{n-j}. \tag{4.9}$$

Since the Sylvester matrix and its subresultant matrices are used for the computation of an AGCD of two inexact polynomials, all the computations are performed on $S_k(f_{\theta_0}, \alpha_0\, g_{\theta_0})$, where $k$ denotes the subresultant matrix order, and $f_{\theta_0}(w)$ and $g_{\theta_0}(w)$ are defined in (4.8) and (4.9), respectively.

**Example 4.1.** Let two exact polynomials be

$$\begin{aligned}\hat{f}(y) &= (y-10^{-3})^4(y-10^{-1})^2(y-10^4)\\ \hat{g}(y) &= (y-10^{-3})^3(y-10^{-2})^3(y-10^4),\end{aligned}$$

whose coefficients have been scaled by $\alpha_0 = 1.1545$ and $\theta_0 = 0.0362$, after normalising each polynomial by the geometric mean of its coefficients, and it can be seen that deg GCD$(\hat{f},\hat{g}) = 4$. Figures 4.1 (a) and (b) show how scaling by these two parameters reduces the variation in the magnitude of the coefficients of $\hat{f}(y)$ and $\hat{g}(y)$, respectively. Figure 4.1(a) shows that the ratios

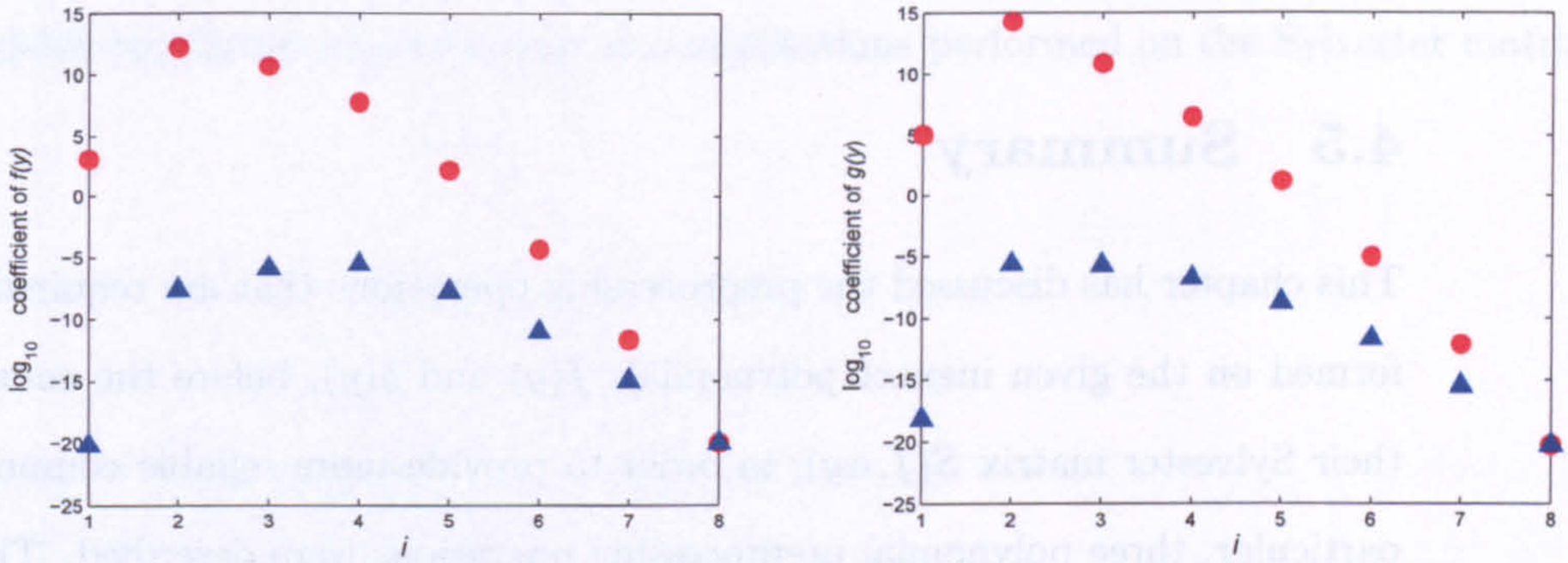
$$\log \frac{\max_{i=0,\ldots,7}|\hat{a}_i|}{\min_{i=0,\ldots,7}|\hat{a}_i|}, \quad \text{and} \quad \log \frac{\max_{i=0,\ldots,7}|\bar{a}_i\theta_0^{m-i}|}{\min_{i=0,\ldots,7}|\bar{a}_i\theta_0^{m-i}|},$$

of the coefficients of $\hat{f}(y)$ and $f_{\theta_0}(w)$, respectively, have reduced from 32.2362 to 14.6347, and for $\hat{g}(y)$, Figure 4.1(b) shows that the reduction in the ratios

$$\log \frac{\max_{j=0,\ldots,7}|\hat{b}_j|}{\min_{j=0,\ldots,7}|\hat{b}_j|}, \quad \text{and} \quad \log \frac{\max_{j=0,\ldots,7}|\bar{b}_j\theta_0^{m-j}|}{\min_{j=0,\ldots,7}|\bar{b}_j\theta_0^{m-j}|},$$

of $\hat{g}(y)$ and $g_{\theta_0}(w)$ is from 34.5388 to 14.6347. This shows that the preprocessing operations have reduced the variations in the magnitude of the coefficients of the polynomial by several orders of magnitude.

Figure 4.2 shows the normalised singular values of $S(\hat{f},\hat{g})$ and $S(f_{\theta_0}, \alpha_0\, g_{\theta_0})$, where clearly it can be seen that the rank of $S(\hat{f},\hat{g})$ is not defined, whereas the rank of $S(f_{\theta_0}, \alpha_0\, g_{\theta_0})$ is equal to 10 and thus deg GCD$(f_{\theta_0}, g_{\theta_0}) = 4$, which is correct. Thus, in this example, the preprocessing operations that have been presented in this chapter




(a) (b)

Figure 4.1: (a) The coefficient ranges of the normalised exact polynomial $\hat{f}(y)$, • and the scaled version of it, ▲, (b) The coefficient ranges of the normalised exact polynomial $\hat{g}(y)$, • and the scaled version of it, ▲.

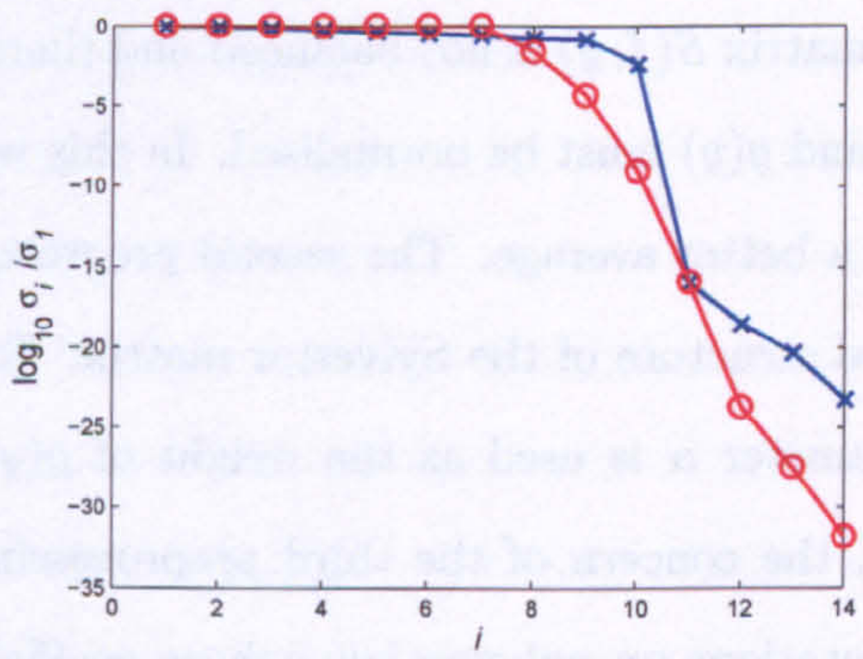



Figure 4.2: The normalised singular values of $S(\hat{f}, \hat{g})$ ∘ and $S(f_{\theta_0}, \alpha_0\, g_{\theta_0})$ ×, for Example 4.1.

do not only minimise the coefficient variations, but also help to obtain a well-defined rank deficient matrix. □

## 4.5 Summary

This chapter has discussed the preprocessing operations that are required to be performed on the given inexact polynomials $f(y)$ and $g(y)$, before the computation of their Sylvester matrix $S(f, \alpha g)$, in order to provide more reliable computations. In particular, three polynomial preprocessing operations were described. The first preprocessing operation normalises each polynomial by the geometric mean of its coefficients. The second uses the relative scaling of the given polynomials by the parameter $\alpha$. The third preprocessing operation scales the independent variable with the parameter $\theta$.

The first preprocessing operation is motivated by the fact that if the coefficients of the polynomial $f(y)$ are much smaller or larger than those of the polynomial $g(y)$, the Sylvester matrix $S(f, g)$ is not balanced and therefore the coefficients of the polynomials $f(y)$ and $g(y)$ must be normalised. In this work, the geometric mean is used as it provides a better average. The second preprocessing operation is motivated by the partitioned structure of the Sylvester matrix. To balance this partitioned structure, the parameter $\alpha$ is used as the weight of $g(y)$ relative to the unit weight of $f(y)$. Finally, the concern of the third preprocessing operation is to provide more reliable computations on polynomials, whose coefficients vary widely in magnitude, through scaling the independent variable. The criterion used for the computation of the parameters $\alpha$ and $\theta$ is based on minimising the ratio of the maximum coefficient, in magnitude, to the minimum coefficient, in magnitude, and a technique from linear

programming has been used to compute these two parameters, simultaneously.

It has also been shown, in this chapter, that the use of these preprocessing techniques yields significant improvements in computations performed on the Sylvester matrix.

# Chapter 5

# Overview of AGCD computation

It has been shown in Chapter 1 that the roots of a polynomial $f(y)$ can be computed using a GCD-based algorithm. In particular, if $d(y) = \mathrm{GCD}(f(y), f^{(1)}(y))$, then the values of the distinct roots of $f(y)/d(y)$ are identical to the distinct roots of $f(y)$. Furthermore, the roots of $f(y)/d(y)$ are simple, and they are therefore numerically better conditioned than the multiple roots of $f(y)$.

In addition to its application in the root finding problem, the computation of the GCD of two polynomials has widespread applications in different fields, such as control theory [5], signal processing, system identification [56, 59, 71], satellite communications and image processing [42, 62]. In communications, for example, the output data is represented as a convolution of the input data and the impulse response of the channel. Let $x_i(n)$ represent the $i$th input of a multichannel system whose impulse response is given by $h(n)$. The $i$th output $y_i(n)$ of this system can be represented by $y_i(n) = x_i(n) * h(n)$, and the $z$-transform $Y_i(z)$ of $y_i(n)$ is

$$Y_i(z) = X_i(z)H(z),$$

where $X_i(z)$ and $H(z)$ are the $z$-transforms of $x_i(n)$ and $h(n)$, respectively. The property $H(z)$ *is a common factor in all receiving blocks* is used for the blind channel identification problem [56, 72]. In [56], for example, two receiving blocks were used such that the output of these blocks are given by,

$$Y_1(z) = X_1(z)H(z), \quad \text{and} \quad Y_2(z) = X_2(z)H(z)$$

respectively. The $z$-domain channel impulse response $H(z)$ is then regarded as the GCD of $Y_1(z)$ and $Y_2(z)$, provided that the inputs $X_1(z)$ and $X_2(z)$ are co-prime. In a similar manner, in image processing the final image is represented by a convolution of the desired image and the blurring function (e.g. camera movement). The desired image in the $z$-domain can be considered as the GCD of two or more bivariate polynomials representing images of the same scene.

Practically, exact GCDs are not defined as data are corrupted by random noise, in addition to roundoff error associated with the computation in a floating point environment. Therefore, the computation of the exact GCD in the presence of noise is meaningless and an approximate solution, that is, an AGCD, should be considered. The problem of computing an AGCD for a pair of polynomials has been presented intensively in the literature. This chapter is devoted to provide an overview of the previous work proposed for the computation of an AGCD of two inexact univariate polynomials. The problem of computing an AGCD is first addressed, in Section 5.1. Section 5.2 discusses some of the literature on AGCD definitions. Some known approaches are summarised in 5.3, and these include Euclid's algorithm, the resultant approach, and the optimisation approach. The root solver developed in this thesis involves the development of robust methods for the computation of an AGCD

of two polynomials, and its contributions towards the research done in the AGCD computation is listed in Section 5.4.

## 5.1 Problem statement

The problem of computing the GCD of two polynomials is usually stated as follows: Given two polynomials $f(y)$ and $g(y)$, compute their greatest common divisor, $d(y)$, that is,

$$f(y) = u(y)d(y), \quad g(y) = v(y)d(y),$$

where, $u(y)$ and $v(y)$ are co-prime. The most widely known algorithm for calculating the GCD of a pair of polynomials is Euclid's algorithm [1] which is the oldest non-trivial algorithm still in use [40]. This algorithm is efficient if the given data and the arithmetic operations are error-free. This ideal situation is not achieved in practice because a small perturbation in the coefficients of the given polynomials may yield, with high probability, a constant GCD. Thus in the presence of noise, the GCD is not defined and only an AGCD can be considered. An important difference between these two types of GCDs is that the exact GCD is unique, up to an arbitrary non-zero scalar multiplier, whereas the AGCD is not unique. In particular, an AGCD can be defined in several ways, and the definition used must be appropriate for the problem to be solved. Moreover, for a certain AGCD definition, there may exist several polynomials that satisfy the requirements of this definition. Some of the AGCD definitions found in the literature are considered in the next section.

## 5.2 Definitions of the AGCD

Several definitions of an AGCD of a pair of inexact polynomials are proposed in the literature, including the $\epsilon$-GCD [8, 16, 24, 51, 63], the $\delta$-GCD [51, 53], the appGCD [63] and the quasi-GCD [68].

The $\epsilon$-GCD or so-called the nearest GCD is the most widely considered AGCD. The common underlying principle in computing this type of AGCD is to find a nearby pair of polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$ that are within a distance $\epsilon$ from the given polynomials $f(y)$ and $g(y)$, and have a non-trivial GCD. An $\epsilon$-GCD of two given inexact polynomials can be defined as follows:

**Definition 5.1.** *A polynomial $\tilde{d}(y)$ is said to be an* $\epsilon$-GCD *of the given inexact polynomials $f(y)$ and $g(y)$, whose degrees are $m$ and $n$, respectively, if there exist small perturbations $\delta f(y)$ and $\delta g(y)$, such that for $\tilde{f}(y) = f(y) + \delta f(y)$ and $\tilde{g}(y) = g(y) + \delta g(y)$, the following statements hold true:*

1. $\deg \tilde{f}(y) = m$ *and* $\deg \tilde{g}(y) = n$.

2. $\|f(y) - \tilde{f}(y)\| \leq \epsilon \|f(y)\|$ *and* $\|g(y) - \tilde{g}(y)\| \leq \epsilon \|g(y)\|$.

3. $\tilde{d}(y)$ *is an exact* GCD *of* $\tilde{f}(y)$ *and* $\tilde{g}(y)$.

It follows from Definition 5.1 that an $\epsilon$-GCD is not unique because there may exist several polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$ that satisfy the properties in the definition.

In [53], Pan argues that the $\epsilon$-GCD is unstable since its degree is sensitive to small perturbations of its coefficients. He introduces the $\delta$-GCD. This AGCD is defined by the roots rather than the coefficients of the polynomial. It implies that the roots of the polynomials have to be computed first, and Pan has suggested some available

root finder algorithms such as [50, 52, 54]. An obvious disadvantage of this approach is that it requires the roots to be calculated accurately.

The definitions of the $\epsilon$-GCD and $\delta$-GCD consider the inexact polynomials whose coefficients are known imperfectly or are perturbed with roundoff error. This is in contrast to the quasi-GCD definition proposed by Schönhage. In particular, given two univariate polynomials $f(y)$ and $g(y)$ with error bound $\epsilon > 0$, a polynomial $h(y)$ is called a quasi-GCD if $h(y)$ is an $\epsilon$-approximate divisor of $f$ and $g$, and any exact common divisor of $f$ and $g$ is an approximate divisor of $h(y)$. The computation of $h(y)$ requires the computation of the cofactors $u(y)$ and $v(u)$ such that $|uf+vg-h| < \epsilon|h|$. While the input data, in reality, can only be found for limited digits of accuracy, the quasi-GCD definition assumes that more digits can be obtained on demand. This assumption limits the use of the quasi-GCD polynomials to symbolical computations.

Zeng [84] states that an AGCD must possess the following properties:

1. Nearness: The AGCD is the exact solution of a nearby pair of polynomials.

2. Max-degree: The AGCD has the maximum degree among all polynomials that satisfy the nearness property.

3. Min-distance: The AGCD of a given pair of polynomials minimises the distance between the polynomials for which it is exact, and the given polynomials.

The definition of an AGCD for the work presented in this thesis is based on the assumption that the degree of the AGCD is known using the methods described in Chapter 6, and it can be formalised as follows:

**Definition 5.2.** *Given a pair of polynomials $f = f(y)$ and $g = g(y)$, whose AGCD $\tilde{d}(y)$ is of degree $d$, with* deg $f = m$ *and* deg $g = n$, *compute $\tilde{f} = \tilde{f}(y)$ and $\tilde{g} = \tilde{g}(y)$,*

*such that*

1. $\deg \tilde{f} = m$, *and* $\deg \tilde{g} = n$.

2. *The error*

$$\| f(y) - \tilde{f}(y) \|^2 + \| g(y) - \tilde{g}(y) \|^2,$$

*is minimised.*

3. $\tilde{d}(y)$ *is an exact* GCD *of* $\tilde{f}(y)$ *and* $\tilde{g}(y)$, *and* $\deg \tilde{d}(y) = d$.

## 5.3 The AGCD computations: Some known approaches

Once a definition for the AGCD has been chosen, the computation of an AGCD for a given pair of inexact polynomials can be performed via several approaches. This section summarises some known approaches for the computation of an AGCD, which includes Euclid's algorithm [33, 48], the resultant approach [16, 24], and the optimisation approach [14, 37, 38].

### 5.3.1 Euclid's algorithm

Euclid's algorithm for the computation of the GCD of two exact polynomials is considered by Brown [13], and Collins and George [15]. Its numerical case has been handled by several researchers, using variants of Euclid's algorithm. In order to illustrate the use of the modified versions of Euclid's algorithm in the computation

of an AGCD of two inexact polynomials, it is important to first review the classical Euclidian algorithm for the computation of the GCD of two exact polynomials.

### Classical Euclid's algorithm

The computation of the GCD of two polynomials using Euclid's algorithm can be described as follows: Given a pair of exact polynomials, $\hat{f}(y)$ and $\hat{g}(y)$, with deg $\hat{f} \geq$ deg $\hat{g}$, compute the GCD of $\hat{f}(y)$ and $\hat{g}(y)$ through repeated polynomial divisions $\hat{f}_i(y)/\hat{g}_i(y)$, such that

$$\hat{f}_i(y) = \hat{q}_i(y)\hat{g}_i(y) + \hat{r}_i(y),$$

where $\hat{q}_i(y)$ are the quotient polynomials, $\hat{r}_i(y)$ are the remainder polynomials, and

$$\begin{aligned} \hat{f}_i(y) &= \hat{f}(y), & \hat{g}_i(y) &= \hat{g}(y), & i &= 0 \\ \hat{f}_i(y) &= \hat{g}_{i-1}(y), & \hat{g}_i(y) &= \hat{r}_{i-1}(y) & i &> 0. \end{aligned}$$

The process is repeated until $r_i = 0$ in which case $\text{GCD}(\hat{f}, \hat{g}) = g_i$. Algorithm 5.3.1 describes the application of Euclid's algorithm for the computation of the GCD of two polynomials.

---

**Algorithm 4.1: Euclid's Algorithm**

**Input** The exact polynomials $\hat{f}(y)$ and $\hat{g}(y)$.

**Output** The GCD $\hat{d}(y)$ of $\hat{f}(y)$ and $\hat{g}(y)$.

**Begin**

1. Set $i = 0, \hat{f}_i(y) = \hat{f}(y), \hat{g}_i(y) = \hat{g}(y)$.

2. **While** $\hat{r}_i(y) \neq 0$.

   (a) Compute the polynomials $\hat{q}_i(y)$ and $\hat{r}_i(y)$, such that

   $$\hat{f}_i(y) = \hat{q}_i(y)\hat{g}_i(y) + \hat{r}_i(y).$$

   (b) $i \rightarrow i + 1$.

   (c) Set $\hat{f}_i(y) = \hat{g}_{i-1}(y)$, $\hat{g}_i(y) = \hat{r}_{i-1}(y)$.

   **End While**

3. Set $\hat{d}(y) = \hat{g}_i(y)$.

**End**

---

The set of polynomials $\hat{r}_1, \hat{r}_2, \cdots, \hat{r}_k \neq 0$ is called the polynomial remainder sequence, PRS. Note that Euclid's algorithm must terminate as the degree of $\hat{r}_i(y)$ is decreasing with $i$.

**Example 5.1.** Consider the polynomials

$$\begin{aligned} \hat{f}(y) &= y^3 + 3y^2 - 4 = (y+2)^2(y-1), \\ \hat{g}(y) &= y^2 + 2y - 3 = (y-1)(y+3). \end{aligned}$$

The polynomial division $\hat{f}(y)/\hat{g}(y)$ yields

$$y^3 + 3y^2 - 4 = (y+1)(y^2 + 2y - 3) + (y-1),$$

for which $\hat{r}_0(y) = y - 1 \neq 0$, and therefor the process should continue, that is $\hat{f}_1(y) = \hat{g}(y)$ and $\hat{g}_1(y) = \hat{r}_0(y)$. The second polynomial division yields,

$$y^2 + 2y - 3 = (y+3)(y-1).$$

Since $\hat{r}_1 = 0$, the divisions stop and $\mathrm{GCD}(\hat{f}, \hat{g}) = y - 1$, which is correct. □

As noted above Euclid's algorithm requires recursive polynomial long divisions, which is computationally unstable [85]. This instability of Euclid's algorithm was pointed out by several researchers, and stabilised versions of Euclid's algorithm have been proposed. The stabilised versions are mainly based on either a careful choice of the termination criterion [32, 33, 48] of the algorithm or look ahead strategies that jump over the ill-conditioned subproblems [6].

### 5.3.2 Resultant approach

It is recalled from Chapter 3 that the GCD of two exact polynomials can be computed using resultant matrices. More precisely, the GCD computation involves two stages:

1. The identification of the degree of the GCD
2. The determination of the coefficients of the GCD.

It was shown in Chapter 3 that the problem of computing the degree of the GCD is reduced to a rank determination problem. Once the degree of the GCD is known,

its coefficients can be found by applying a triangular decomposition of the resultant matrix, such as LU or QR [28], (see Example 3.1). These computations are usually applied to the Sylvester resultant matrix and the other resultant matrices.

Considering the inexact polynomials, several algorithms have been proposed to calculate approximate solutions for the two stages of the GCD computation.

The computation of the rank of a matrix is usually performed using the singular value decomposition, SVD [28], and the theoretical basis of this use of the SVD for the calculation of the degree of an AGCD is considered in detail in Chapter 6. Corless et al. [16], for example, compute the degree of the AGCD by using the SVD of the Sylvester matrix of the given polynomials. In particular, they look for the largest gap in the singular values of the Sylvester matrix. To compute the coefficients of the AGCD, they propose several strategies, including solving a minimisation problem using an optimisation technique. Emiris et al. [23], apply the SVD to the subresultant matrices of the Sylvester matrix of the given polynomials to compute upper bounds of the degree of the AGCD. Generally, the use of the SVD-based approach for the computation of an AGCD is considered to be stable. However, the SVD based methods are computationally intensive, especially as the degree of the polynomial increases. Moreover, Emiris et al. [24], show that using the singular values of the Sylvester matrix is not enough to solve the problem completely, and established a gap theorem on the singular values of the subresultant matrices that provides conditions under which the degree of the AGCD can be certified.

Methods based on the QR decomposition have also been used [17, 83], for the computation of an AGCD of two inexact polynomials. These methods exploit the fact that the QR decomposition of the resultant matrix of two polynomials reveals the

coefficients of the GCD in the last non-zero row of the upper triangular factor $R$. However, if the coefficients of the polynomial are perturbed randomly with noise, it is impossible to identify the last non-zero row. Zarowski et al. [83] show that the rank of $R$ is equal to the index of last non-zero row, and instead of using the SVD to compute the rank of $R$, they use the estimated smallest singular values of leading principal submatrices, using an estimator provided by Bischof [10]. The estimated smallest singular values are then further processed to estimate the last non-zero singular value using the algorithm presented by Zarowski [82]. The methods based on the QR decomposition are generally stable, but suffer from instability if the given polynomials have large common roots. Corless et. al. [17] have pointed out this instability and suggested a method to improve it. Zarowski et. al. [83] attempt to solve this problem by making the polynomial monic, but it is observed in [87] that this strategy does not guarantee stability. However, the results in [2] show that the QR-based method proposed by Corless et. al. in [17] fails to achieve the correct GCD degree if the leading coefficient is less than $10^{-5}$. Bini and Boito [2] suggest an alternative solution to overcome the instability of the QR decomposition. They use the QR decomposition with pivoting to determine the upper bound on the degree of the AGCD. Due to the pivoting, the coefficients of the AGCD are no longer available in $R$, and in order to compute the coefficients of the AGCD, Bini and Boito compute the co-prime factors of the given polynomials from the null space of the Sylvester matrix after which they apply polynomial division to obtain the coefficients of the AGCD. The computed AGCD is then refined iteratively.

### 5.3.3 Optimisation approach

In this approach, the problem of computing an AGCD is posed as an optimisation problem, with the aim of minimising the distance between the given polynomials and the computed perturbed pair. The problem formulation for this approach can be stated as follow: For a given pair of polynomials $f$ and $g$ with $\deg(f) = m$ and $\deg(g) = n$, compute $\tilde{f}$ and $\tilde{g}$ with $\deg(\tilde{f}) = m$ and $\deg(\tilde{g}) = n$, such that the error

$$\| f - \tilde{f} \|^2 + \| g - \tilde{g} \|^2,$$

is minimised [14, 37, 38]. Karmarkar and Lakshman [38] developed algorithms to compute the nearest AGCDs of maximum degree of a given inexact pair of polynomials by minimising the perturbations to be added to the given polynomials. They point out that the efficiency of [31] decreases as the multiplicities of the roots increase, and they claim that their optimisation approach can be extended to compute roots of higher multiplicities.

Chin and Corless [14] have formulated and solved a non-linear optimization problem that minimises the perturbation that can be added to a polynomial pair to have a non-trivial GCD. They assumed that the degree of the AGCD as well as its initial estimate are known, using the methods in [16].

Other AGCD computation approaches involve the Padé approximation [6, 51] and statistical approaches [71]. Stoica and Söderström [71] introduced a non-iterative maximum likelihood-based method through which the coefficients of the polynomials are assumed to have a Gaussian random distribution, which is in reality not true and

the polynomials may be much more complicated.

Currently, the most computationally efficient methods are the structured matrix based methods [39, 41, 78, 87]. Despite the extensive work conducted using this approach, there are still some open issues such as those related to the nature of the minimum perturbation to be added to the inexact polynomials in order to have a non-constant GCD, the technical complexities encountered with hard classes of polynomials, such as those with several multiple roots, roots with high multiplicities and close roots, and more importantly, the development of data driven methods that do not require prior knowledge about the noise level.

## 5.4 Contributions to the literature

Generally, the study of the methods considered in the literature shows that the vast majority of these methods require a threshold to determine the index of the largest non-zero singular value in order to determine the degree of the AGCD of the given polynomials from the rank of their resultant matrix. This requires prior knowledge about the noise level, which may not be known or only known approximately. Thus, methods that do not require any prior knowledge about the noise level need to be developed. In addition, the preprocessing operations that scale the given polynomials before the AGCD computations are performed, provide more reliable computations, but these operations are neglected from the majority of the studies.

The work in this research differs from the previous work in the following aspects:

1. It develops of a set of preprocessing operations that improve the quality of the computed AGCD. In particular, they involve

(a) The normalisation of the given polynomials by the geometric means of their coefficients, instead of the 2-norms of their coefficients.

(b) Technique from linear programming for the computation of the optimal values of two parameters that make the AGCD computations more reliable. These parameters are necessary if the coefficients of the polynomials vary widely in magnitude.

2. It introduces methods for the determination of the rank of the AGCD of two inexact polynomials that do not require the noise level to be known.

3. It introduces the use of the non-linear structure preserving matrix methods for the computation of an AGCD.

## 5.5 Summary

This chapter has provided an overview of the AGCD computation which forms one of the main stages in the developed root solver. The concept of the AGCD has been illustrated and its non-uniqueness has been discussed, based on the different definitions available in the literature. The main approaches for the AGCD computation have been presented and some of the deficiencies in these approaches have been discussed. Finally, the contributions of the work in this research towards the research done in the area of AGCD computations have been listed.

# Chapter 6

# The computation of the degree of an AGCD

The root solver considered in Section 2.3 is a GCD-based root solver and it can be seen from Algorithm 2.3.1 that the computation of the GCD of two polynomials forms the first step in this root solver. However, practically, problems exist when computations are performed on polynomials whose coefficients are contaminated with error. Thus the GCD of two polynomials is not defined and it is only possible to compute an AGCD. Usually, the computation of an AGCD of two inexact polynomials involves two stages. In particular, in the first stage the degree of the AGCD is determined, after which the coefficients of the AGCD are calculated. This chapter and the next chapter consider, respectively, the computation of the first and second stages in the AGCD computation.

The computation of an estimate for a degree of an AGCD of two inexact polynomials is a non-trivial task. It has been shown in Section 3.1 that this problem is reduced to a rank determination problem because the degree of the GCD of two polynomials

is equal to the rank loss of their Sylvester matrix. The most frequently used method for solving this rank determination problem is the SVD [16]. In particular, let $R$ be an $m \times n$ matrix and $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ be orthogonal matrices. The matrix $R$ can be factorised as [28]

$$R = U\Sigma V^T,$$

where

$$\Sigma = \text{diag}(\sigma_1, \sigma_2, \cdots, \sigma_p) \in \mathbb{R}^{m \times n}, \qquad p = \min(m, n),$$

and the $\sigma_i, i = 1, 2, \cdots, p$, are the singular values of $R$. Let the number of the non-zero values of $\sigma_i$ be $r$, which is then referred to as the rank of the matrix $R$. This implies that

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \cdots = \sigma_p = 0.$$

The question that now arises is: *Is this condition always satisfied by exact matrices whose entries are specified exactly, when computations are performed on them?* The answer to this question will be clear from the following two examples.

**Example 6.1.** Consider the following two exact polynomials

$$\begin{aligned} \hat{f}(y) &= (y-3)(y-1)^2(y-2)^3, \\ \hat{g}(y) &= (y-1)^2. \end{aligned}$$

Clearly it can be seen that deg $\mathrm{GCD}(\hat{f}, \hat{g}) = 2$. Thus, by Theorem 3.1, the theoretical rank of $S(\hat{f}, \hat{g})$ is $(6+2)-2 = 6$. This result agrees with the result obtained using the MATLAB function `svd()` to compute the singular values of $R$. The normalised values of these singular values are plotted in Figure 6.1, in a logarithmic scale. It is obvious that the rank of $S(\hat{f}, \hat{g})$ is $r = 6$, and therefore deg $\mathrm{GCD}(\hat{f}, \hat{g}) = 2$, which is correct. □

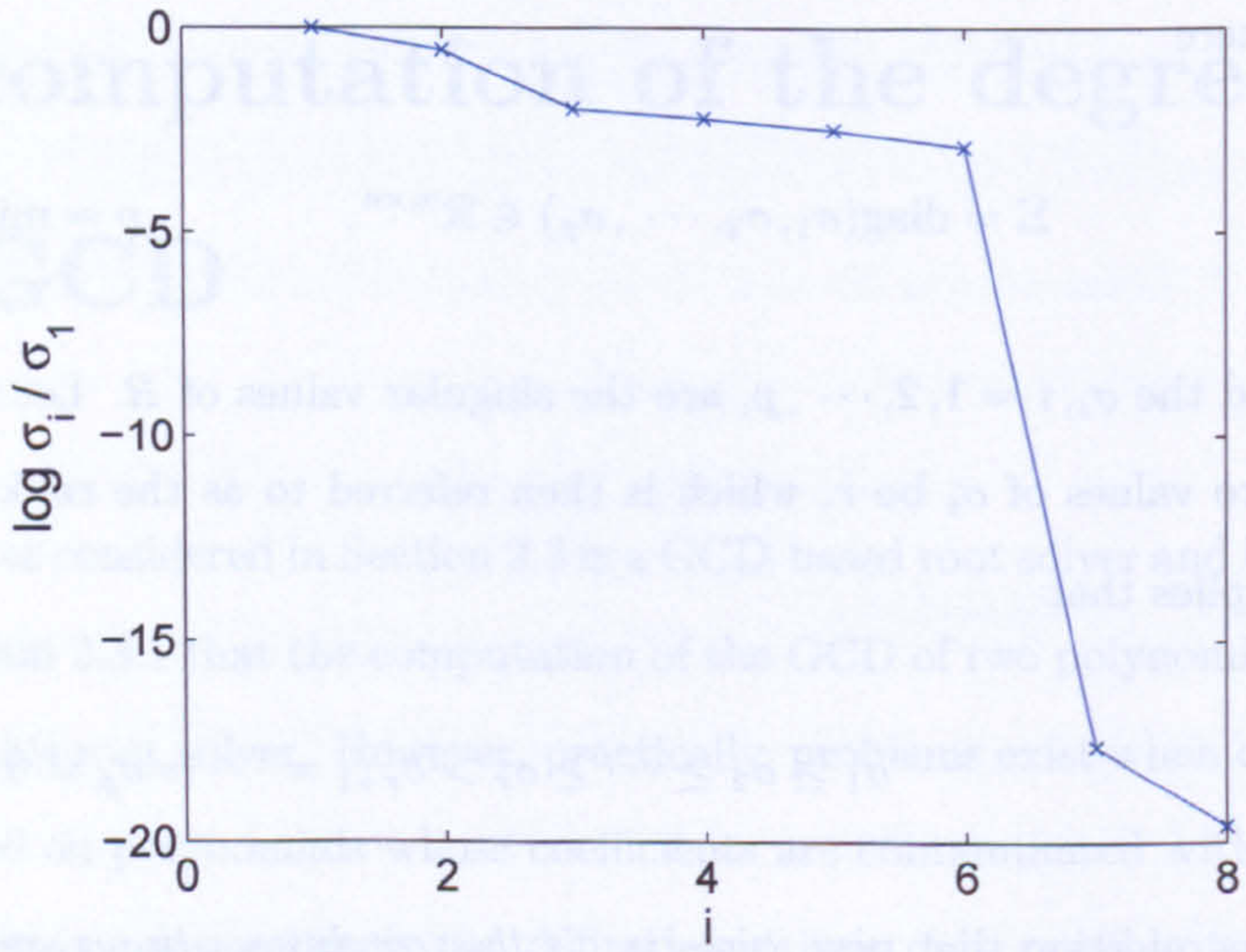



Figure 6.1: The normalised singular values of $S(\hat{f}, \hat{g})$, on a logarithmic scale, for Example 6.1.

**Example 6.2.** Consider the following two exact polynomials

$$\begin{aligned} \hat{f}(y) &= (y-3.5)(y-3.7)^2(y-3.9)^3, \\ \hat{g}(y) &= (y-3.7)^5. \end{aligned}$$

Clearly it can be seen that deg $\mathrm{GCD}(\hat{f}, \hat{g}) = 2$. Thus, by Theorem 3.1, the theoretical rank of $S(\hat{f}, \hat{g})$ is $(6+5)-2 = 9$. This result is not obtained by the function `svd()` in MATLAB. In particular, the normalised values of $R$ are shown in Figure 6.2, in a logarithmic scale, and it is obvious that the rank of $S(\hat{f}, \hat{g})$ is $r = 6$, and therefore the degree of the GCD of $\hat{f}(y)$ and $\hat{g}(y)$ is equal to $11 - 6 = 5$, which is incorrect. □

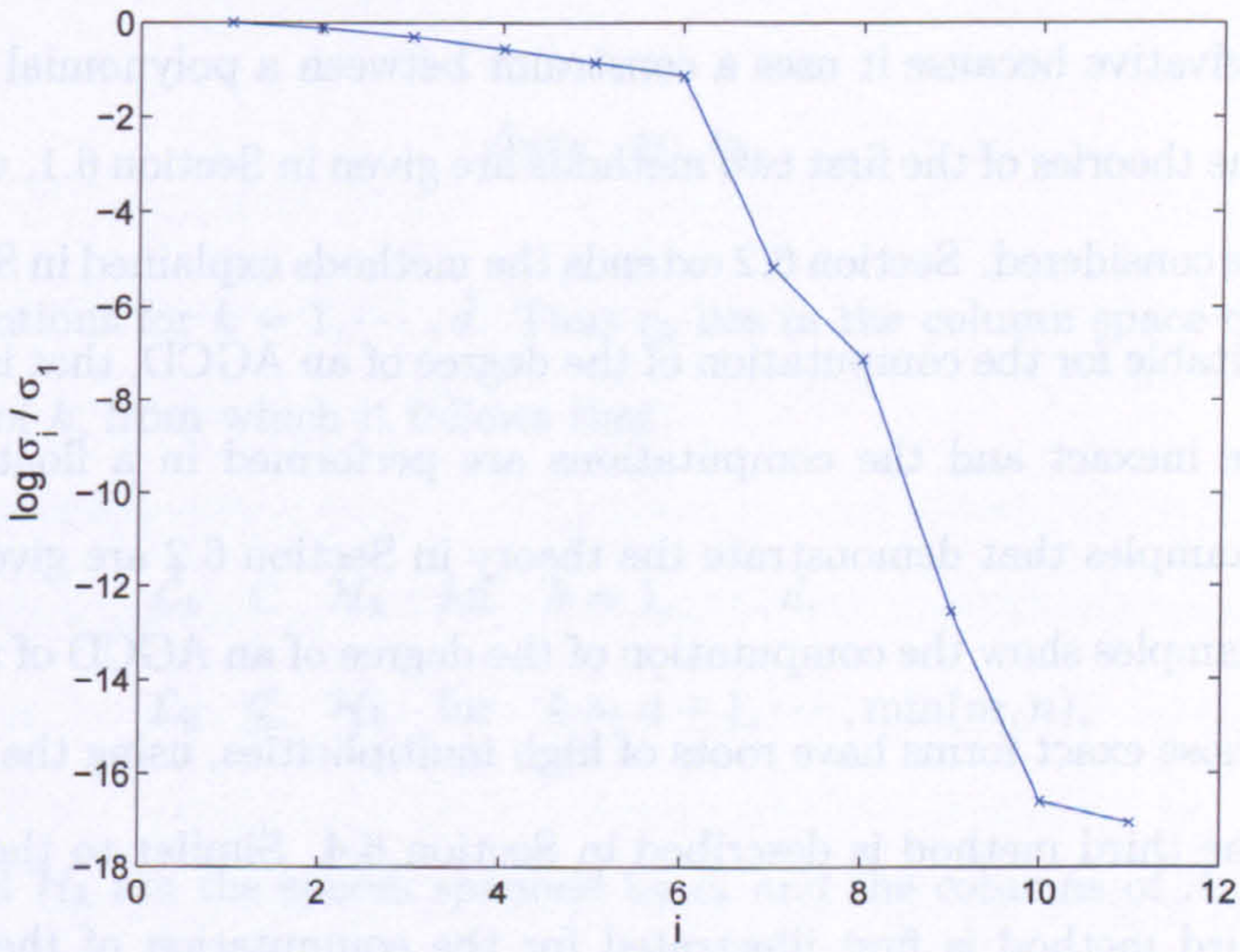



Figure 6.2: The normalised singular values of $S(\hat{f}, \hat{g})$, on a logarithmic scale, for Example 6.2.

Example 6.2 shows how easily the SVD-based method fails in the presence of small errors such as roundoff errors. In practice, the SVD-based methods usually require a threshold on the small singular values of $S(\hat{f}, \hat{g})$, and they fail to compute the correct rank loss of $S(\hat{f}, \hat{g})$ if inexact data is considered. These limitations provide the motivation for the computational methods for the determination of the rank of the Sylvester matrix that are proposed in this chapter. Three methods are developed

for computing the degree of a GCD/AGCD of two univariate polynomials. All of them use the Sylvester matrix, but they differ in the criteria used. These methods perform all the computations on the coefficients of the polynomials and do not require any prior knowledge about the noise level.

The first two methods are applicable to an arbitrary pair $(f(y), g(y))$ of polynomials. The third method, on the other hand, is only applicable to a polynomial and its derivative because it uses a constraint between a polynomial and its derivative.

The theories of the first two methods are given in Section 6.1, where exact polynomials are considered. Section 6.2 extends the methods explained in Section 6.1 to make them suitable for the computation of the degree of an AGCD, that is, when the polynomials are inexact and the computations are performed in a floating point environment. Examples that demonstrate the theory in Section 6.2 are given in Section 6.3. These examples show the computation of the degree of an AGCD of two inexact polynomials whose exact forms have roots of high multiplicities, using the methods in Section 6.2. The third method is described in Section 6.4. Similar to the first two methods, the third method is first illustrated for the computation of the degree of the GCD of an exact polynomial and its derivative, in Section 6.4.1. Modifications of the theory discussed in Section 6.4.1 are given in Section 6.4.2 to illustrate the computation of the degree of an AGCD of an inexact polynomial and its derivative. Examples that demonstrate the theory in Section 6.4.2 are given in Section 6.5.

## 6.1 Computing the degree of the GCD of two exact polynomials

Consider the two exact polynomials $\hat{f}(y)$ and $\hat{g}(y)$ defined in (3.1), and let these two polynomials be non-coprime whose GCD is of degree $\hat{d}$. It is recalled from Section 3.2 that the linear algebraic equation

$$A_k x_k = c_k, \tag{6.1}$$

possesses solutions for $k = 1, \cdots, \hat{d}$. Thus $c_k$ lies in the column space of $A_k$ only for these values of $k$, from which it follows that

$$\mathcal{L}_k \subset \mathcal{H}_k \quad \text{for} \quad k = 1, \cdots, \hat{d},$$
$$\mathcal{L}_k \not\subset \mathcal{H}_k \quad \text{for} \quad k = \hat{d}+1, \cdots, \min(m, n), \tag{6.2}$$

where $\mathcal{L}_k$ and $\mathcal{H}_k$ are the spaces spanned by $c_k$ and the columns of $A_k$ respectively. This forms the bases of two new approaches for the compution of the degree of the GCD of $\hat{f}(y)$ and $\hat{g}(y)$. These two methods determine whether $c_k$ lies in the column space of $A_k$ or not, using two different criteria, namely the angle between the subspaces $\mathcal{L}_k$ and $\mathcal{H}_k$, and the residual of (6.1). In each of these approaches the degree $\hat{d}$ is equal to the largest value of $k$ for which $c_k$ lies in the column space of $A_k$.

**Method 1: First principal angle.** Let $F$ and $G$ be subspaces in $\mathbb{R}^n$, and assume that

$$p = \dim F \geq \dim G = q \geq 1.$$

**Definition 6.1.** *[12] The principal angles $\theta_i \in [0, \pi/2]$, for $i = 1, \cdots, q$, between $F$ and $G$ are defined by*

$$\cos\theta_i = \max_{u \in F} \max_{v \in G} u^H v = u_i^H v_i, \ \|u\|_2 = 1, \|v\|_2 = 1,$$

*such that $u_j^H u = 0$, $v_j^H v = 0$, $j = 1, \cdots, k-1$.*

The vectors $(u_1, \cdots, u_p)$ and $(v_1, \cdots, v_q)$ are called principal vectors of $F$ and $G$. The determination of the degree $\hat{d}$ of the GCD of $\hat{f}(y)$ and $\hat{g}(y)$, in this method, is done through computing the angle $\phi_k$ between the subspaces $\mathcal{L}_k$ and $\mathcal{H}_k$ that are spanned by the column $c_k$ and the columns of $A_k$ respectively, for $k = 1, \cdots, \min(m, n)$. If this angle is equal to zero, then $\mathcal{L}_k$ lies in $\mathcal{H}_k$, which implies that the vector $c_k$ lies in the column space of the matrix $A_k$. Therefore, due to (6.2), $\phi_k$ satisfies,

$$\begin{aligned} \phi_k &= 0 \quad \text{for} \quad k = 1, \cdots, \hat{d}, \\ \phi_k &> 0 \quad \text{for} \quad k = \hat{d}+1, \cdots, \min(m, n). \end{aligned} \tag{6.3}$$

The degree, $\hat{d}$, can then be chosen to be the largest value of $k$ for which $\phi_k = 0$.

**Method 2: Residual.** In this method, the residual of equation (6.1) is used as an error indicator, from which the GCD degree can be deduced. Particulary, in light

of (6.2) it can be seen that the residual $r_k$,

$$r_k = \|c_k - A_k x_k\|,$$

satisfies

$$\begin{aligned} r_k &= 0 \quad \text{for} \quad k = 1, \cdots, \hat{d}, \\ r_k &> 0 \quad \text{for} \quad k = \hat{d}+1, \cdots, \min(m,n). \end{aligned} \tag{6.4}$$

Clearly, this shows that the change in $r_k$ occurs after $k = \hat{d}$, and therefore the GCD degree is equal to largest value of $k$ for which $r_k = 0$.

Though these two methods have been developed independently, they are related geometrically as shown in Figure 6.3. Consider the linear algebraic equation $Ax = b$,

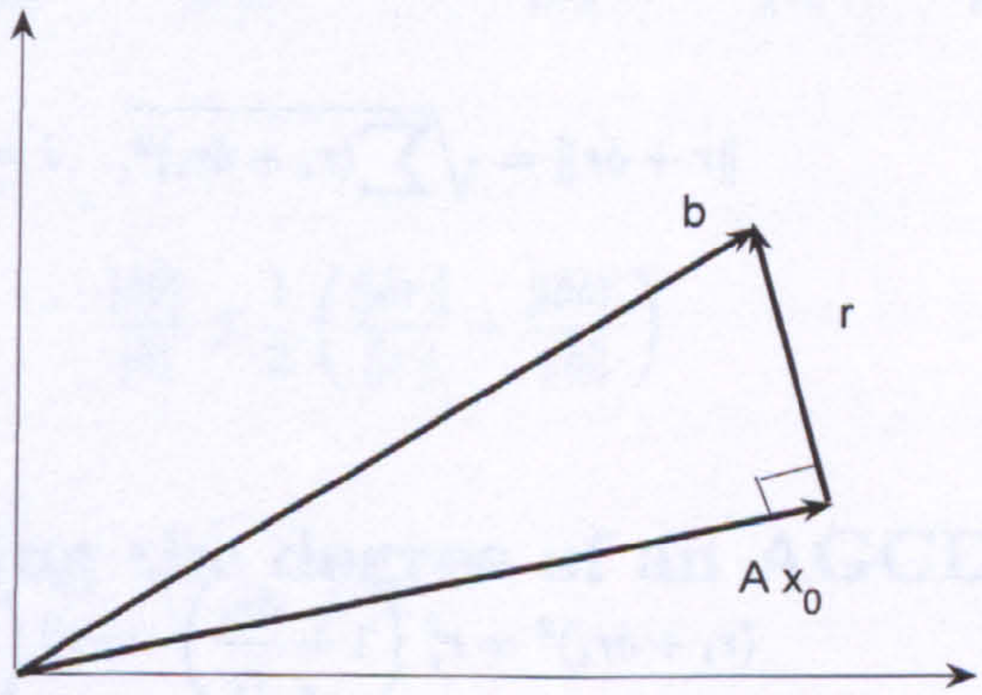



Figure 6.3: Geometry of the least squares problem.

$A \in \mathbb{R}^{r_A \times q_A}$. The least squares solution of this equation is

$$x_0 = A^\dagger b, \quad A^\dagger = (A^T A)^{-1} A^T,$$

and the residual associated with this approximate solution is,

$$r = b - Ax_0 = (I - AA^\dagger)b.$$

This residual is related to the angle $\theta$ between $Ax_0$ and $b$ by the relation $\sin\theta = \frac{\|r\|}{\|b\|}$. Let $\delta\theta$ and $\delta r$ represent the small changes in the angle and the residual, respectively, then

$$\sin(\theta + \delta\theta) = \frac{\|r + \delta r\|}{\|b + \delta b\|},$$

where

$$\|r + \delta r\| = \sqrt{\sum (r_i + \delta r_i)^2}, \quad i = 1, \cdots, r_A$$

and

$$(r_i + \delta r_i)^2 = r_i^2 \left(1 + \frac{\delta r_i}{r_i}\right)^2 \simeq r_i^2 \left(1 + 2\frac{\delta r_i}{r_i}\right).$$

Thus

$$\|r + \delta r\| \simeq \sqrt{\|r\|^2 + 2r^T \delta r} = \sqrt{\|r\|^2 \left(1 + \frac{2r^T \delta r}{\|r\|^2}\right)} \simeq \|r\| \left(1 + \frac{r^T \delta r}{\|r\|^2}\right),$$

to the first order. Similarly,

$$\|b+\delta b\|^{-1} \simeq \frac{1}{\|b\|}\left(1-\frac{b^T\delta b}{\|b\|^2}\right).$$

Therefore,

$$\frac{\sin(\theta+\delta\theta)}{\sin\theta} = 1-\frac{b^T\delta b}{\|b\|^2}+\frac{r^T\delta r}{\|r\|^2},$$

to the first order. Since $\theta \ll 1$,

$$\frac{\delta\theta}{\theta} = \frac{r^T\delta r}{\|r\|^2}-\frac{b^T\delta b}{\|b\|^2} = \frac{1}{2}\left[\frac{\delta(\|r\|^2)}{\|r\|^2}-\frac{\delta(\|b\|^2)}{\|b\|^2}\right]$$

where

$$\frac{\delta(\|r\|^2)}{\|r\|^2} = \frac{\delta(r^Tr)}{\|r\|^2} = \frac{2r^T\delta r}{\|r\|^2} \quad \text{and} \quad \frac{\delta(\|b\|^2)}{\|b\|^2} = \frac{\delta(b^Tr)}{\|b\|^2} = \frac{2b^T\delta b}{\|b\|^2}.$$

Thus,

$$\frac{|\delta\theta|}{|\theta|} \le \frac{1}{2}\left(\frac{\|\delta r\|}{\|r\|}+\frac{\|\delta b\|}{\|b\|}\right).$$

## 6.2 Computing the degree of an AGCD of two inexact polynomials

The previous section considers the computation of the degree of the GCD of two exact polynomials. The methods discussed in that section assume that the data is error free and the computations are done in a perfect computational environment. Practically,

the subresultant matrices of the Sylvester matrix of inexact polynomials have full rank and only an AGCD for these polynomials is defined and can be computed. This section extends the theory discussed in the previous section to make it suitable for the computation of the degree of an AGCD of two inexact polynomials.

It is recalled that the preprocessing operations discussed in Chapter 4 are needed when inexact polynomials are specified. These preprocessing operations transform the given polynomials $f(y)$ and $g(y)$ into $f_{\theta_0}(w)$ and $\alpha_0\, g_{\theta_0}(w)$, which are defined in (4.8) and (4.9) respectively, where $\alpha_0$ and $\theta_0$ are the optimal values of $\alpha$ and $\theta$ respectively, and their values are obtained by solving the LP problem in (4.7). Thus all the computations are performed on these polynomials.

### 6.2.1 Best column selection

It was shown in Section 3.2 that if the exact polynomials $\hat{f}(y)$ and $\hat{g}(y)$ are considered, and they have a common divisor of degree $k$, then the first column $c_k$ of $S_k(\hat{f}, \hat{g})$ can be moved to the right hand side because it necessarily lies in the column space of $S_k(\hat{f}, \hat{g})$. The situation is more complicated when the inexact polynomials $f_{\theta_0}(w)$ and $\alpha_0\, g_{\theta_0}(w)$ are considered because $S_k(f_{\theta_0}, \alpha_0\, g_{\theta_0})$ has full rank and none of columns of $S_k(f_{\theta_0}, \alpha_0\, g_{\theta_0})$ lie in the space spanned by the remaining columns of $S_k(f, \alpha_0\, g_{\theta_0})$, for all values of $k = 1, \cdots, \min(m, n)$. Equation (3.11) must therefore be modified to reflect the non-singular property of $S_k(f_{\theta_0}, \alpha_0\, g_{\theta_0})$.

Let $c_{k,i}$ denote the $i^{th}$ column of $S_k(f_{\theta_0}, \alpha_0\, g_{\theta_0})$, $i = 1, \cdots, m+n-2k+2$. If the $i^{th}$ column is moved to the right hand side, (3.11) is replaced by the approximation,

$$A_{k,i}x_{k,i} \approx c_{k,i}, \tag{6.5}$$

where $A_{k,i}$ is the matrix formed by the remaining columns of $S_k(f_{\theta_0}, \alpha_0 g_{\theta_0})$, that is,

$$A_{k,i} = \begin{bmatrix} c_{k,1} & c_{k,2} & \cdots & c_{k,i-1} & c_{k,i+1} & \cdots & c_{k,m+n-2k+2} \end{bmatrix}.$$

The best values of $k$ and $i$ are calculated such that the error in (6.5) is minimum. Note that, depending on the value of $i$, the $c_{k,i}$ entries may contain either the coefficients of $f_{\theta_0}(w)$ or the coefficients of $\alpha_0 g_{\theta_0}(w)$. This problem is ignored in the literature, where the default value $i = 1$ is always used, but it is addressed in detail in this thesis. Moreover, the careful selection of the indices $k$ and $i$ allows robust methods for the computation of the degree of an AGCD of two inexact polynomials to be developed. Kaltofen et. al. [36] use one example to argue that the first column of the Sylvester matrix of the exact polynomials $\hat{f}(y)$ and $\hat{g}(y)$ should be chosen to form the overdetermined system (6.5), that is $i = 1$. The following two examples demonstrate the weakness of this argument. The first example considers the same example given in [36] and the second example considers another pair of polynomials.

**Example 6.3.** Consider the two exact polynomials,

$$\begin{aligned} \hat{f}(y) &= y^2 + y = y(y+1), \\ \hat{g}(y) &= y^2 + 4y + 3 = (y+3)(y+1), \end{aligned}$$

whose Sylvester matrix is given by

$$S = S(\hat{f}, \hat{g}) = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 4 & 1 \\ 0 & 1 & 3 & 4 \\ 0 & 0 & 0 & 3 \end{bmatrix}.$$

To examine whether the first column is the best column to be taken to the right hand side or not, $S(\hat{f}, \hat{g})$ has been partitioned into $[c_i \quad A_i]$, where $c_i$ is the $i$th column of $S(\hat{f}, \hat{g})$ and $A_i$ is the matrix formed by the remaining columns of $S(\hat{f}, \hat{g})$. The systems of the equations $A_i x = c_i$ derived from $S(\hat{f}, \hat{g})$ were formed for $i = 1, 2, 3$ and 4, and the results of computing the error made in solving these systems are shown in Table 6.1.

Table 6.1: The solutions and the associated residuals of the systems of equations for Example 6.3.

| Column index | $x^T$ | Residual |
|---|---|---|
| 1 | [-3 1 0] | $2.9966 \times 10^{-15}$ |
| 2 | [-0.3333 0.3333 0] | $1.4199 \times 10^{-15}$ |
| 3 | [1 3 0] | $1.1102 \times 10^{-15}$ |
| 4 | [-1273 0.8182 0.7273] | 3.4641 |

These results show that the first, second and third columns of $S(\hat{f}, \hat{g})$ are all perfectly adequate and any one of them can be moved to the right hand side, but the fourth column gives the wrong answer and thus it should not be moved to the right hand side. □

Table 6.2: Comparing the residuals for Example 6.4.

| Column index | Residual |
|---|---|
| 1 | $8.0382 \times 10^2$ |
| 8 | 1.1894 |

**Example 6.4.** Consider the two exact polynomials,

$$\begin{aligned} \hat{f}(y) &= (y+4.6)^6(y-8.5)^{10}(y-1.8)^4 \\ \hat{g}(y) &= (y-8.5)^{10}(y-1.8)^7, \end{aligned}$$

whose GCD degree is 14. The residuals of the overdetermined systems of $S_{14}(\hat{f},\hat{g})$ have been computed for $i = 1, 2, \cdots, 11$, where $i$ denotes the column index of $S_{14}(\hat{f},\hat{g})$. The minimum residual, which was found to be associated with $i = 8$, was then compared with the residual computed at $i = 1$. Table 6.2 shows this comparison, where it is shown that the first column is not the best column to be taken to the right hand side to form the system (6.5). This result contradicts the theory in Section 3.2 where it is assumed that, considering exact data, the first column of the $k$th subresultant matrix can always be taken to the right hand side to form (3.11). This discrepancy between the theoretical and computational results is due to roundoff error. Also the wide variation in the magnitude of the coefficients of the polynomials of this example, which does not exist in Example 6.3, may also explain the differences between the theoretical and the computational results. □

Examples 6.3 and 6.4 show that even if exact polynomials are considered, the first column of their Sylvester matrix is not always the best column to form $A_k x_k = c_k$, especially if the coefficients of the polynomials have wide variation in magnitude and

the computations are performed in a floating point environment. It is therefore clear that, the situation is more complicated when the inexact polynomials $f(y)$ and $g(y)$, are considered.

Considering inexact polynomials, it follows from (6.5) that for the values of $k = 1, \cdots, \min(m, n)$, the values of $\phi_k$ and $r_k$ defined in (6.3) and (6.4), respectively, are non-zero. Thus, Methods 1 and 2 described in the previous section need to be modified to accommodate the inexact nature of the polynomials.

The rest of this section extends the computation of the degree of the GCD discussed in the previous section to make them suitable for inexact polynomials $f_{\theta_0}(w)$ and $\alpha_0\, g_{\theta_0}(w)$. It is shown that the computation of the optimal column index $i$ of $S_k(f_{\theta_0}, \alpha_0\, g_{\theta_0})$ follows directly from the computation of the degree $k = d$ of the AGCD of $f_{\theta_0}(w)$ and $\alpha_0\, g_{\theta_0}(w)$.

### 6.2.2 Method 1: First principal angle

The smallest angle between the space $\mathcal{L}_{k,i}$ spanned by $c_{k,i}$ and the space $\mathcal{H}_{k,i}$ spanned by the columns of $A_{k,i}$ is called the *first principal angle*, $\psi_{k,i}$ [75]. Thus,

$$\psi_{k,i} = \min \angle(\mathcal{L}_{k,i}, \mathcal{H}_{k,i}), \quad k = 1, \cdots, \min(m, n); i = 1, \cdots, m + n - 2k + 2, \quad (6.6)$$

where

$$\dim \mathcal{L}_{k,i} = 1 \qquad \text{and} \qquad \dim \mathcal{H}_{k,i} = m + n - 2k + 1.$$

For each value of $k$, the minimum value $\phi_k$ of $\psi_{k,i}$ is computed such that

$$\phi_k = \min_i\{|\ \psi_{k,i}\ |: i = 1, \cdots, m+n-2k+2\}, \tag{6.7}$$

and the column index $i = q_{\phi,k}$ at which the minima $\phi_k$ occurs is recorded to yield the optimal column vector $\mathbf{q}_\phi$,

$$\mathbf{q}_\phi = [q_{\phi,1}, q_{\phi,2}, \cdots, q_{\phi,\min(m,n)}], \tag{6.8}$$

where the subscript $\phi$ denotes that a criterion based on the first principal angle is used to compute these column indices.

Let $d_\phi$ denote the computed degree of an AGCD. Although the values of $\phi_k$ for $k = 1, \cdots, d_\phi$ can not be zero because of the presence of inexact data, the values of $\phi_k$ for these values of $k$ are small compared to those values of $\phi_k$ for $k > d_\phi$. Thus the degree $d_\phi$ of an AGCD equals the index $k$ for which the change in $\phi_k$ between two successive values of $k$ is maximum,

$$d_\phi = \{k : (\phi_{k+1} - \phi_k) \longrightarrow \max; k = 1, \cdots, \min(m,n)\}, \tag{6.9}$$

and the index $i = q_{\phi,d_\phi}$ of the optimal column $c_{k,i}$ in (6.5) is the $d_\phi$th element in (6.8).

**Example 6.5.** Let $\min(m,n) = 7$ and the vector $\phi = [\ \phi_1,\ \ \phi_2,\ \ \cdots,\ \ \phi_7\ ]$ of the angles be

$$\phi = [\ 10^{-8}\ \ 10^{-9}\ \ 10^{-8}\ \ 10^{-3}\ \ 10^{-3}\ \ 10^{-2}\ \ 10^{-3}\ ].$$

The values of the angles $\phi_1$, $\phi_2$ and $\phi_3$ are relatively small and therefore the associated

approximate solutions of (6.5) are acceptable. By contrast, the values of the angles $\phi_4, \cdots, \phi_7$ are large, which suggests that the associated approximate solutions of (6.5) are associated with large errors. This discussion therefore leads to the conclusion that there are common divisors of degrees 1, 2 and 3, but there is no common divisor of degree greater than 3. Thus the degree $d_\phi$ of the AGCD is equal to three. □

To be able to use the expression in (6.9) that defines the criterion for computing $d_\phi$, it is first required to evaluate the angle $\psi_{k,i}$ defined in (6.6), between the space $\mathcal{L}_{k,i}$ spanned by $c_{k,i}$ and the space $\mathcal{H}_{k,i}$ spanned by the columns of the matrix $A_{k,i}$. The following theory discusses the computation of $\psi_{k,i}$. According to [49], this computation goes back to Jordan 1875, and it has been considered in [28, 75].
In order to obtain an expression for $\psi_{k,i}$, it is required to calculate an orthonormal basis for $\mathcal{H}_{k,i}$, and this can be obtained by applying the QR decomposition to the matrix $A_{k,i}$,

$$A_{k,i} = N_{k,i} R_{k,i}, \qquad N_{k,i}^T N_{k,i} = I_{m+n-2k+1}, \tag{6.10}$$

where the columns of $N_{k,i} \in \mathbb{R}^{(m+n-k+1)\times(m+n-2k+1)}$ define an orthonormal basis for $\mathcal{H}_{k,i}$, and $R_{k,i} \in \mathbb{R}^{(m+n-2k+1)\times(m+n-2k+1)}$ is an upper triangular matrix. Thus each vector $v_{k,i} \in \mathcal{H}_{k,i}$ can be written as

$$v_{k,i} = N_{k,i} w_{k,i}, \qquad w_{k,i} \in \mathbb{R}^{m+n-2k+1}.$$

The first principal angle $\psi_{k,i}$ between $\mathcal{L}_{k,i}$ and $\mathcal{H}_{k,i}$ is equal to the smallest angle

between the unit vectors $u_{k,i} = \frac{c_{k,i}}{\|c_{k,i}\|} \in \mathcal{L}_{k,i}$ and $v_{k,i} \in \mathcal{H}_{k,i}$,

$$\cos\psi_{k,i} = \max_{\|v_{k,i}\|=1} u_{k,i}^T v_{k,i} = \max_{\|w_{k,i}\|=1} \left(u_{k,i}^T N_{k,i}\right) w_{k,i}. \tag{6.11}$$

If the SVD of $u_{k,i}^T N_{k,i}$ is

$$u_{k,i}^T N_{k,i} = \Sigma_{k,i} Q_{k,i}^T,$$

where $\Sigma_{k,i} = \begin{bmatrix}\sigma_{k,i,1} & 0 & \cdots & 0\end{bmatrix} \in \mathbb{R}^{m+n-2k+1}$ and $Q_{k,i} \in \mathbb{R}^{(m+n-2k+1)\times(m+n-2k+1)}$ is an orthogonal matrix, then (6.11) yields

$$\cos\psi_{k,i} = \max_{\|v\|=1} u_{k,i}^T v_{k,i} = \max_{\|w\|=1} \left(\Sigma Q_{k,i}^T\right) w_{k,i} = \left\|u_{k,i}^T N_{k,i}\right\| = \sigma_{k,i,1}.$$

This implies that the cosine of the first principal angle is equal to the 2-norm of $u_{k,i}^T N_{k,i}$, or equivalently, the largest singular value of $u_{k,i}^T N_{k,i}$,

$$\cos\psi_{k,i} = \left\|u_{k,i}^T N_{k,i}\right\| = \sigma_{k,i,1}.$$

This maximum is attained when $w_{k,i}$ is equal to the first column $q_{k,i,1}$ of $Q_{k,i}$,

$$v_{k,i} = N_{k,i} q_{k,i,1}.$$

Thus, the first principal angle between $\mathcal{L}_{k,i}$ and $\mathcal{H}_{k,i}$ is given by

$$\psi_{k,i} = \cos^{-1}\sigma_{k,i,1}. \tag{6.12}$$

However, this equation implies that to first order,

$$\delta\psi_{k,i,1} = -\frac{\delta\sigma_{k,i,1}}{\sin\psi_{k,i}}, \tag{6.13}$$

and clearly it can be seen that in the presence of inexact data, the formula in (6.12) does not yield correct results for small angles, $\psi_{k,i} \approx 0$, because $|\delta\psi_{k,i}| \gg |\delta\sigma_{k,i,1}|$ if $\psi_{k,i} \approx 0$. An alternative approach for computing $\psi_{k,i}$ is given in [75] by considering the orthonormal complements $\mathcal{L}_{k,i}^{\perp}$ and $\mathcal{H}_{k,i}^{\perp}$, where

$$\mathcal{L}_{k,i} \cup \mathcal{L}_{k,i}^{\perp} = \mathbb{R}^r \qquad \text{and} \qquad \mathcal{H}_{k,i} \cup \mathcal{H}_{k,i}^{\perp} = \mathbb{R}^r,$$
$$\dim \mathcal{L}_{k,i}^{\perp} = r-1 \qquad \text{and} \qquad \dim \mathcal{H}_{k,i}^{\perp} = r-q,$$

and $r = m+n-k+1$ and $q = m+n-2k+1$. The consideration of the orthogonal complements $\mathcal{L}_{k,i}^{\perp}$ and $\mathcal{H}_{k,i}^{\perp}$ leads to a stable expression for small $\psi_{k,i}$.

Let the columns of the matrices $U_{k,i,2} \in \mathbb{R}^{r\times(r-1)}$ and $N_{k,i,2} \in \mathbb{R}^{r\times(r-q)}$ define the orthonormal bases for $\mathcal{L}_{k,i}^{\perp}$ and $\mathcal{H}_{k,i}^{\perp}$ respectively, and redefine the vector $u_{k,i}$ and the matrix $N_{k,i}$ to be $u_{k,i,1}$ and $N_{k,i,1}$ respectively,

$$u_{k,i,1} := u \in \mathbb{R}^r \qquad \text{and} \qquad N_{k,i,1} := N \in \mathbb{R}^{r\times q}.$$

Thus $u_{k,i,1}$ defines a unit vector that spans $\mathcal{L}_{k,i}$, and the columns of $N_{k,i,1}$ define an orthonormal basis for $\mathcal{H}_{k,i}$. It follows that the columns of $U_{k,i}$ and $N_{k,i}$, which are redefined as

$$U_{k,i} = \begin{bmatrix} u_{k,i,1} & U_{k,i,2} \end{bmatrix} \in \mathbb{R}^{r\times r}, \qquad U_{k,i}^T U_{k,i} = U_{k,i} U_{k,i}^T = I_r, \tag{6.14}$$

and

$$N_{k,i} = \left[ \begin{array}{cc} N_{k,i,1} & N_{k,i,2} \end{array} \right] \in \mathbb{R}^{r \times r}, \qquad N_{k,i}^T N_{k,i} = N_{k,i} N_{k,i}^T = I_r, \tag{6.15}$$

respectively, define orthonormal bases for $\mathbb{R}^r$.

**Theorem 6.1.** *Let $\mathcal{L}_{k,i}$ and $\mathcal{H}_{k,i}$ be subspaces of $\mathbb{R}^r$, and let $\theta_i$ be the ith principal angle between them. The unit vector $u_{k,i,1} \in \mathbb{R}^r$ spans the line $\mathcal{L}_{k,i}$, and the columns of $N_{k,i,1} \in \mathbb{R}^{r \times q}$ define an orthonormal basis for $\mathcal{H}_{k,i}$. Also, let the columns of $U_{k,i,2} \in \mathbb{R}^{r \times (r-1)}$ and $N_{k,i,2} \in \mathbb{R}^{r \times (r-q)}$ define orthonormal bases for $\mathcal{L}_{k,i}^{\perp}$ and $\mathcal{H}_{k,i}^{\perp}$ respectively, where (6.14) and (6.15) are satisfied. Then the singular values of $U_{k,i,2}^T N_{k,i,1} \in \mathbb{R}^{(r-1) \times q}$ and $u_{k,i,1}^T N_{k,i,2} \in \mathbb{R}^{r-q}$ are*

$$\sin\theta_1 \leq \sin\theta_2 \leq \cdots \leq \sin\theta_q.$$

**Proof** Since $U_{k,i}$ is an orthogonal matrix and $N_{k,i,1}$ has orthonormal columns, the columns of $W_1 \in \mathbb{R}^{r \times q}$,

$$W_1 = U_{k,i}^T N_{k,i,1} = \left[ \begin{array}{c} u_{k,i,1}^T N_{k,i,1} \\ U_{k,i,2}^T N_{k,i,1} \end{array} \right], \qquad u_{k,i,1}^T N_{k,i,1} \in \mathbb{R}^q, \qquad U_{k,i,2}^T N_{k,i,1} \in \mathbb{R}^{(r-1) \times q},$$

are also orthonormal. Also, the singular values of $u_{k,i,1}^T N_{k,i,1}$ are $\gamma_i = \cos\theta_i, i = 1, \ldots, q$, and it follows from Theorem 6.16 that the singular values of $U_{k,i,2}^T N_{k,i,1}$ are

$$\sigma_i = \sqrt{1 - \gamma_i^2} = \sin\theta_i, \qquad i = 1, \ldots, q.$$

Consider now the vector $W_2 \in \mathbb{R}^r$,

$$W_2 = N^T u_{k,i,1} = \begin{bmatrix} N_{k,i,1}^T u_{k,i,1} \\ N_{k,i,2}^T u_{k,i,1} \end{bmatrix}, \qquad N_{k,i,1}^T u_{k,i,1} \in \mathbb{R}^q, \qquad N_{k,i,2}^T u_{k,i,1} \in \mathbb{R}^{r-q}.$$

The singular values of $N_{k,i,1}^T u_{k,i,1}$ are $\cos\theta_i, i = 1, \ldots, q$, and thus it follows from Theorem 6.16 that the singular values of $N_{k,i,2}^T u_{k,i,1}$ and $u_{k,i,1}^T N_{k,i,2}$ are $\sin\theta_i, i = 1, \ldots, q$. □

Since the singular values of $u_{k,i,1}^T N_{k,i,2}$ and $U_{k,i,2}^T N_{k,i,1}$ are $\sigma_i = \sin\theta_i, i = 1, \ldots, q$, it follows that the principal angles are

$$\theta_i = \sin^{-1}\sigma_i, \qquad i = 1, \ldots, q,$$

and thus to first order,

$$\delta\theta_i = \frac{\delta\sigma_i}{\cos\theta_i}, \qquad i = 1, \ldots, q, \tag{6.16}$$

from which it follows that if $\theta_i \approx 0$, then $|\delta\theta_i| \approx |\delta\sigma_i|$. The principal angle $\theta_i$ is therefore stable with respect to changes in $\sigma_i$ when $\theta_i \approx 0$, which must be compared with the situation defined in (6.13). By contrast, if $\theta_i \approx \frac{\pi}{2}$, then (6.13) shows that $\theta_i$ can be calculated in a stable manner from $\mathcal{L}_{k,i}$ and $\mathcal{H}_{k,i}$, but it follows from (6.16) that its calculation from $\mathcal{L}_{k,i}^{\perp}$ and $\mathcal{H}_{k,i}^{\perp}$ is unstable.

The only issue that must still be addressed is the calculation of the matrices $U_{k,i,2}$ and $N_{k,i,2}$, whose columns define orthonormal bases for $\mathcal{L}_{k,i}^{\perp}$ and $\mathcal{H}_{k,i}^{\perp}$ respectively. It is recalled that $N_{k,i,1}$ is calculated from the QR decomposition of $A_{k,i}$, as shown in (6.10), with $u$ and $N$ replaced by $u_{k,i,1}$ and $N_{k,i,1}$ respectively, as noted above.

The unit vector $u_{k,i,1}$ lies in $\mathcal{L}_{k,i}$, and thus all vectors $x \in \mathbb{R}^r$ that satisfy

$$u_{k,i,1}^T x = 0,$$

are orthogonal to $u_{k,i,1}$, from which it follows that they lie in $\mathcal{L}_{k,i}^{\perp}$. It is necessary to choose an orthonormal set of vectors $x$ because an orthonormal basis for $\mathcal{L}_{k,i}^{\perp}$ is required.

If the SVD of $u_{k,i,1}$ is

$$u_{k,i,1} = P \begin{bmatrix} \sigma \\ 0 \end{bmatrix},$$

where $P \in \mathbb{R}^{r \times r}$ is orthogonal, $\sigma \in \mathbb{R}$ is the singular value of $u_{k,i,1}$, and the zero vector is of order $r - 1$, then

$$u_{k,i,1}^T p_k = \begin{bmatrix} \sigma & 0^T \end{bmatrix} P^T p_k,$$

where $p_k, k = 1, \ldots, r$, is the $k$th column of $P$. It is necessary to consider two situations, which are defined by $k = 1$ and $2 \leq k \leq r$.

If $k = 1$, then

$$u_{k,i,1}^T p_1 = \begin{bmatrix} \sigma & 0^T \end{bmatrix} e_1 = \sigma,$$

where $e_1$ is the first unit basis vector.

If $2 \leq k \leq r$, then

$$u_1^T p_k = \begin{bmatrix} \sigma & 0^T \end{bmatrix} e_k = 0,$$

where $e_k$ is the $k$th unit basis vector, and thus the last $r-1$ columns of the left singular matrix $P$ of $u_{k,i,1}$ provide an orthonormal basis for $\mathcal{L}_{k,i}^{\perp}$, that is,

$$U_{k,i,2} = \begin{bmatrix} p_2 & p_3 & \cdots & p_{r-1} & p_r \end{bmatrix}, \tag{6.17}$$

where

$$U_{k,i,2}^T U_{k,i,2} = I_{r-1}, \qquad u_{k,i,1}^T U_{k,i,2} = 0, \qquad U_{k,i,2}^T u_{k,i,1} = 0.$$

The calculation of an orthonormal basis for $\mathcal{H}_{k,i}^{\perp}$, that is, the columns of $N_{k,i,2}$, follows similarly. Specifically, if the SVD of $N_{k,i,1}$ is

$$N_{k,i,1} = P \begin{bmatrix} \Sigma \\ 0 \end{bmatrix} Q^T,$$

where $P \in \mathbb{R}^{r \times r}$, $Q \in \mathbb{R}^{q \times q}$, $\Sigma \in \mathbb{R}^{q \times q}$ is a diagonal matrix of the singular values $\sigma_i$ of $N_1$, arranged in non-increasing order, and the zero matrix is of order $(r-q) \times q$, then

$$N_{k,i,1}^T p_k = Q \begin{bmatrix} \Sigma^T & 0^T \end{bmatrix} P^T p_k,$$

where $p_k, k = 1, \ldots, r$, is the $k$th column of $P$. It is necessary to consider two

situations, which are defined by $1 \leq k \leq q$ and $q+1 \leq k \leq r$.

If $1 \leq k \leq q$, then

$$N_{k,i,1}^T p_k = Q \begin{bmatrix} \Sigma^T & 0^T \end{bmatrix} e_k = \sigma_k q_k,$$

where $q_k$ is the $k$th column of $Q$.

If $q+1 \leq k \leq r$, then

$$N_{k,i,1}^T p_k = Q \begin{bmatrix} \Sigma^T & 0^T \end{bmatrix} e_k = 0,$$

and thus the last $r-q$ columns of the left singular matrix $P$ of $N_{k,i,1}$ provide an orthonormal basis for $\mathcal{H}_{k,i}^{\perp}$, that is,

$$N_{k,i,2} = \begin{bmatrix} p_{q+1} & p_{q+2} & \cdots & p_{r-1} & p_r \end{bmatrix}, \tag{6.18}$$

where

$$N_{k,i,2}^T N_{k,i,2} = I_{r-q}, \qquad N_{k,i,1}^T N_{k,i,2} = 0, \qquad N_{k,i,2}^T N_{k,i,1} = 0.$$

It follows from (6.17) that $U_{k,i,2}$ is defined by the last $r-1$ columns of the left singular matrix of $u_{k,i,1}$, and similarly, it follows from (6.18) that $N_{k,i,2}$ is defined by the last $r-q$ columns of the left singular matrix of $N_{k,i,1}$.

### 6.2.3 Method 2: Residual

Let $x^*$ be an approximate solution for (6.5), and thus the residual of this approximation is

$$r_{k,i} = c_{k,i} - A_{k,i}x^*_{k,i}, \qquad x^*_{k,i} = A^\dagger_k c_{k,i}, \quad A^\dagger_k = (A^T_{k,i}A_{k,i})^{-1}A^T_{k,i}, \tag{6.19}$$

for $k = 1, \cdots, \min(m,n)$, and $i = 1, \cdots, m+n-2k+2$. For each value of $k$, the minimum value of $\|r_{k,i}\|$ is computed using (6.19), such that

$$r_k = \min_i\{\|r_{k,i}\| : i = 1, \cdots, m+n-2k+2\}, \qquad k = 1, \cdots, \min(m,n), \tag{6.20}$$

and the column index $i = q_{r,k}$ at which each minimum $r_k$ occurs is recorded to yield the optimal column vector $\mathbf{q}_r$,

$$\mathbf{q}_r = [q_{r,1}, q_{r,2}, \cdots, q_{r,\min(m,n)}], \tag{6.21}$$

where the subscript $r$ denotes that these column indices are computed using a criterion based on the residual. Let $d_r$ denote the computed degree of an AGCD. Although the values of $r_k$ for $k = 1, \cdots, d_r$ can not be zero because of the presence of inexact data, the values of $r_k$, for these values of $k$ are small compared to those values of $r_k$ for $k > d_r$. Thus the degree $d_r$ of an AGCD equals the index $k$ for which the change in $r_k$ between two successive values of $k$ is maximum,

$$d_r = \{k : (r_{k+1} - r_k) \longrightarrow \max; k = 1, \cdots, \min(m,n)\}, \tag{6.22}$$

and the index $i = q_{r,d_r}$ of the optimal column $c_{k,i}$ in (6.5) is the $d_r$th element in (6.21). Since the polynomials have been normalised before being involved in the AGCD computations, the non-normalised residual has been considered in (6.19).

## 6.3 Examples

This section discusses two examples that illustrate the methods explained in Section 6.2 for the computation of estimates for the degree of an AGCD.

Uniformly distributed random noise was first added, in a componentwise sense, to the coefficients of the theoretically exact polynomials. The resulting polynomials were then called the given polynomials. In particular, consider the exact polynomials $\hat{f}(y)$ and $\hat{g}(y)$ that are defined in (3.1). Adding componentwise uniformly distributed noise to the coefficients of $\hat{f}(y)$ and $\hat{g}(y)$ yields,

$$f(y) = \sum_{j=0}^{m} (\hat{a}_j + \delta\hat{a}_j) y^j,$$

$$g(y) = \sum_{j=0}^{n} (\hat{b}_j + \delta\hat{b}_j) y^j,$$

where $\delta\hat{a}_j = \hat{a}_j r_j \varepsilon_c$, $j = 0, \cdots, m$, and $\delta\hat{b}_j = \hat{b}_j r_j \varepsilon_c$, $j = 0, \cdots, n$, $r_j$ is a uniformly distributed random number in the interval $[-1, 1]$ and $\varepsilon_c^{-1}$ is the upper bound on the componentwise signal-to-noise ratio.

The polynomials $f(y)$ and $g(y)$ were then preprocessed according to the preprocessing operations given in Chapter 4 to have the scaled polynomial forms given in (4.8) and (4.9), respectively.

**Example 6.6.** Componentwise noise with signal-to-noise ratio $\varepsilon_c^{-1} = 10^4$ was added

to the coefficients of the polynomials whose roots and multiplicities are defined in Table 6.3. It can be seen that $\hat{f}(y)$ and $\hat{g}(y)$ have a GCD of degree $\hat{d} = 7$. The perturbed polynomials were then normalised by the geometric means of their coefficients and preprocessed by the optimal scaling factors $\alpha_0 = 0.920253$ and $\theta_0 = 2.905596$. Using Methods 1 and 2, the values of $\phi_k$ and $r_k$ defined in (6.7) and (6.20) respectively, were computed for $k = 1, \cdots, 28$, and the results of the variations of $\log \phi_k$ and $\log r_k$ with $k$ are shown in Figure 6.4. It can be seen that the maximum gradients (6.9) and (6.22) occur at $k = d_\phi = d_r = 7$, which suggests that the degree of the AGCD of $f_{\theta_0}(w)$ and $g_{\theta_0}(w)$ is $d = 7$, which is correct as $\hat{d} = 7$.

Furthermore, the results of computing the optimal columns of $S_k(f_{\theta_0}, \alpha_0 g_{\theta_0})$ for which the minimisations in (6.7) and (6.20) are achieved using Methods 1 and 2, for $k = 1, \cdots, 28$, are shown in Figure 6.5. It can be seen that Methods 1 and 2 do not necessarily have the same optimal column for each value of $k$. However, despite this difference, both methods meet the same value of $k = d_\phi = d_r = 7$ at which their criteria are achieved as shown in Figure 6.4. Figure 6.5 shows that at $k = 7$ the optimal columns $q_{\phi,7} = 26$ and $q_{r,7} = 26$. □

Table 6.3: The roots and multiplicities of $\hat{f}(y)$ and $\hat{g}(y)$ for Example 6.6.

| Root of $\hat{f}(y)$ | Multiplicity |
|---|---|
| 0.6290 | 5 |
| 2.6760 | 8 |
| -9.7181 | 4 |
| -0.5926 | 11 |

| Root of $\hat{g}(y)$ | Multiplicity |
|---|---|
| -9.7181 | 8 |
| -0.5926 | 3 |
| 7.7265 | 7 |
| -7.7194 | 10 |

**Example 6.7.** Consider the theoretically exact polynomials $\hat{f}(y)$ and $\hat{g}(y)$, that are specified by the roots and multiplicities given in Table 6.4. It can be seen that $\hat{f}(y)$

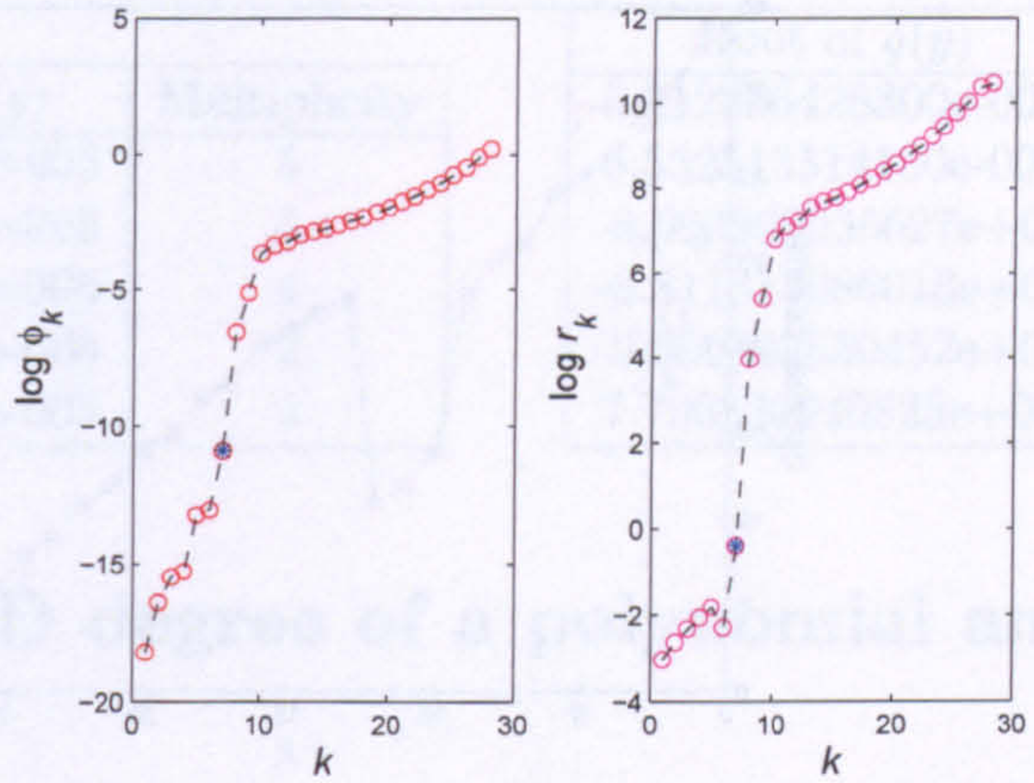



Figure 6.4: (a) The variations with $k$, of $\log \phi_k$ and $\log r_k$ for Example 6.6, where $*$ denotes the exact GCD degree $\hat{d}$.

and $\hat{g}(y)$ have a GCD of degree $\hat{d} = 7$. The polynomials were first perturbed by noise in a componentwise sense such that the signal-to-noise ratio was $\varepsilon_c^{-1} = 10^4$. The resulting polynomials were then normalised by the geometric means of their coefficients and scaled by the optimal preprocessing parameters $\alpha_0 = 2.387441 \times 10^2$ and $\theta_0 = 7.644097 \times 10^{-3}$. Using Methods 1 and 2, the values of $\phi_k$ and $r_k$ defined in (6.7) and (6.20) respectively, were computed for $k = 1, \cdots, 19$, and the results of the variations of $\log \phi_k$ and $\log r_k$ with $k$ are shown in Figure 6.6. It can be seen that the maximum gradients (6.9) and (6.22) occur at $k = d_\phi = d_r = 7$, which suggests that the degree of the GCD of $f_{\theta_0}(w)$ and $g_{\theta_0}(w)$ is $d = 7$, which is correct as $\hat{d} = 7$. Furthermore, the results of computing the optimal columns of $S_k(f_{\theta_0}, \alpha_0\, g_{\theta_0})$ for which the minimisations in (6.7) and (6.20) are achieved using Methods 1 and 2, for $k = 1, \cdots, 19$, are shown in Figure 6.7.

It can be seen that Methods 1 and 2 do not necessarily have the same optimal

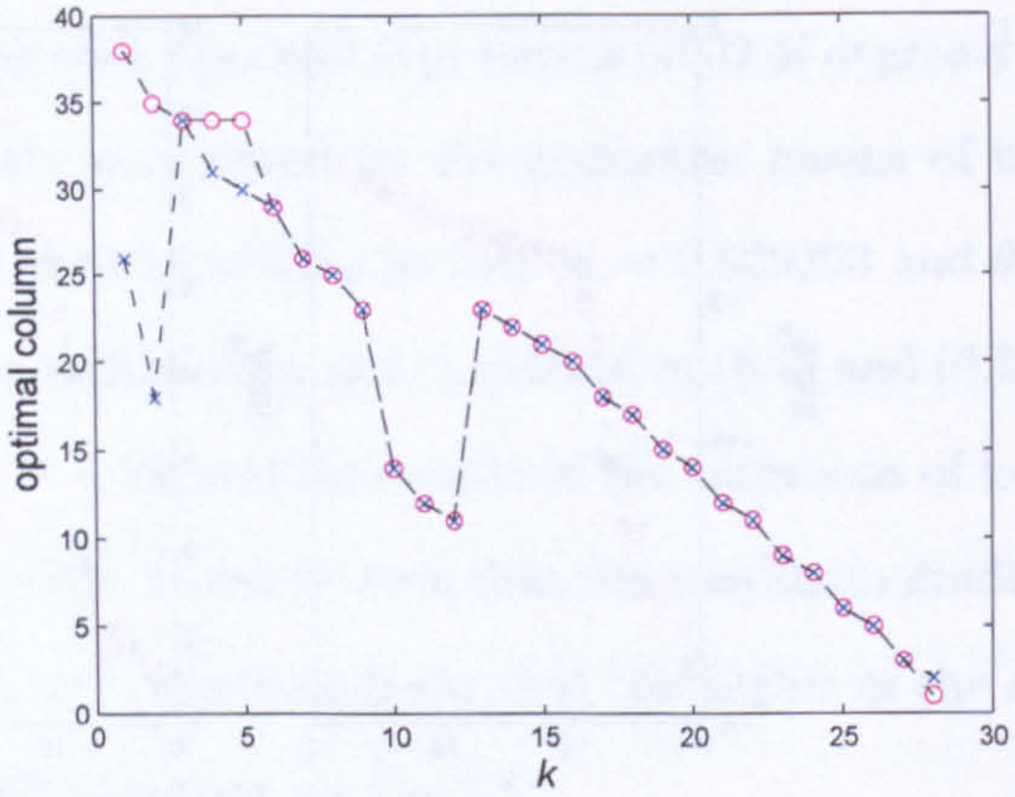



Figure 6.5: The optimal columns of $S_k(f_{\theta_0}, \alpha_0\, g_{\theta_0})$ for which the minimisations in (6.7) and (6.20) are achieved, using Method 1 $\times$, and Method 2 o, for Example 6.6.

column for each value of $k$. Both methods yield the same value of $k = d_\phi = d_r = 7$ at which their criteria are achieved, as shown in Figure 6.6. Figure 6.7 shows that at $k = d_r = d_\phi = 7$ the optimal columns $q_{\phi,7} = 14$ and $q_{r,7} = 14$. □

Experimental results show that Methods 1 and 2 described in Sections 6.2.2 and 6.2.3, respectively, do not necessarily yield the same optimal columns of the Sylvester matrix of $f_{\theta_0}(w)$ and $\alpha_0 g_{\theta_0}(w)$. However, the effect of this is negligible because it will be shown in the next chapter that the computed structured low rank approximations of $S(f_{\theta_0}, \alpha_0\, g_{\theta_0})$ differ slightly and always have a well defined rank drop at $k = \deg \mathrm{GCD}(f_{\theta_0}, \alpha_0\, g_{\theta_0})$. Thus the structured low rank approximations from both methods can be used for subsequent computations.

Table 6.4: The roots and multiplicities of $\hat{f}(y)$ and $\hat{g}(y)$ for Example 6.7.

| Root of $\hat{f}(y)$ | Multiplicity |
|---|---|
| 5.312896426e-005 | 5 |
| 6.532513514e-005 | 5 |
| 9.373846382e-005 | 4 |
| 7.098856547e-005 | 2 |
| 5.825141918e-005 | 3 |

| Root of $\hat{g}(y)$ | Multiplicity |
|---|---|
| 5.312896426300e-005 | 5 |
| 6.532513514100e-005 | 2 |
| -8.990899936627e+00 | 4 |
| -6.311317686013e+00 | 3 |
| -9.086833530452e+00 | 2 |
| 7.700830040825e+00 | 4 |

## 6.4 AGCD degree of a polynomial and its derivative

The methods considered so far for the computation of the degree of an AGCD are applicable to any pair of polynomials, and thus they can be applied to a polynomial and its derivative. However, the computation of an AGCD of a polynomial and its derivative provides a derivative constraint between them, from which another method for the computation of the degree of an AGCD is proposed.

This section introduces this constraint first and then a method that uses this constraint for the computation of the degree of an AGCD of a polynomial and its derivative, is explained. For simplicity, the exact polynomials are first considered and then the necessary modifications to accommodate the uncertainty of the inexact polynomials are discussed.

### 6.4.1 GCD degree of an exact polynomial and its derivative

Consider the exact polynomial $\hat{f} = \hat{f}(y)$ and its derivative $\hat{g}(y) = \hat{f}^{(1)}(y)$ defined in (3.1). It is recalled from Section 3.2 that if the degree of the GCD of $\hat{f}(y)$ and $\hat{f}^{(1)}(y)$

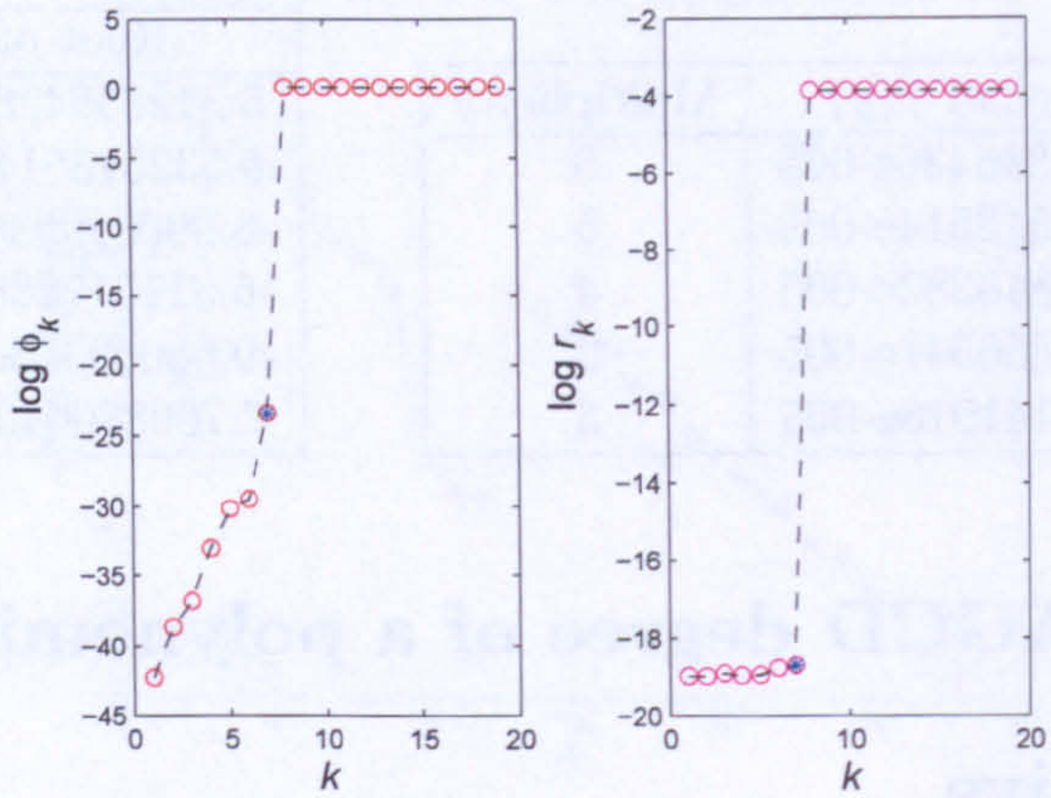



Figure 6.6: The variations with $k$, of $\log \phi_k$ and $\log r_k$ for Example 6.7, where $*$ denotes the exact GCD degree $\hat{d}$.

is equal to $d$, then there exist quotient polynomials $\hat{u}_k(y)$ and $\hat{v}_k(y)$, and a common divisor polynomial $\hat{d}_k(y)$, such that for $k = 1, \cdots, \hat{d}$,

$$\hat{f}(y) = \hat{u}_k(y)\hat{d}_k(y), \quad \text{and} \quad \hat{f}^{(1)}(y) = \hat{v}_k(y)\hat{d}_k(y), \tag{6.23}$$

and thus,

$$\hat{d}_k(y) = \frac{\hat{f}(y)}{\hat{u}_k(y)} = \frac{\hat{f}^{(1)}(y)}{\hat{v}_k(y)}, \tag{6.24}$$

where $\deg \hat{u}_k < \deg \hat{f} = m$, $\deg \hat{u}_k < \deg \hat{f}^{(1)} = n = m - 1$, and

$$\begin{aligned} \hat{u}_k(y) &= \textstyle\sum_{i=0}^{m-k} \hat{u}_{k,i} y^{m-k-i}, \\ \hat{v}_k(y) &= \textstyle\sum_{i=0}^{n-k} \hat{v}_{k,i} y^{n-k-i}, \\ \hat{d}_k(y) &= \textstyle\sum_{i=0}^{k} \hat{d}_{k,i} y^{k-i}. \end{aligned} \tag{6.25}$$

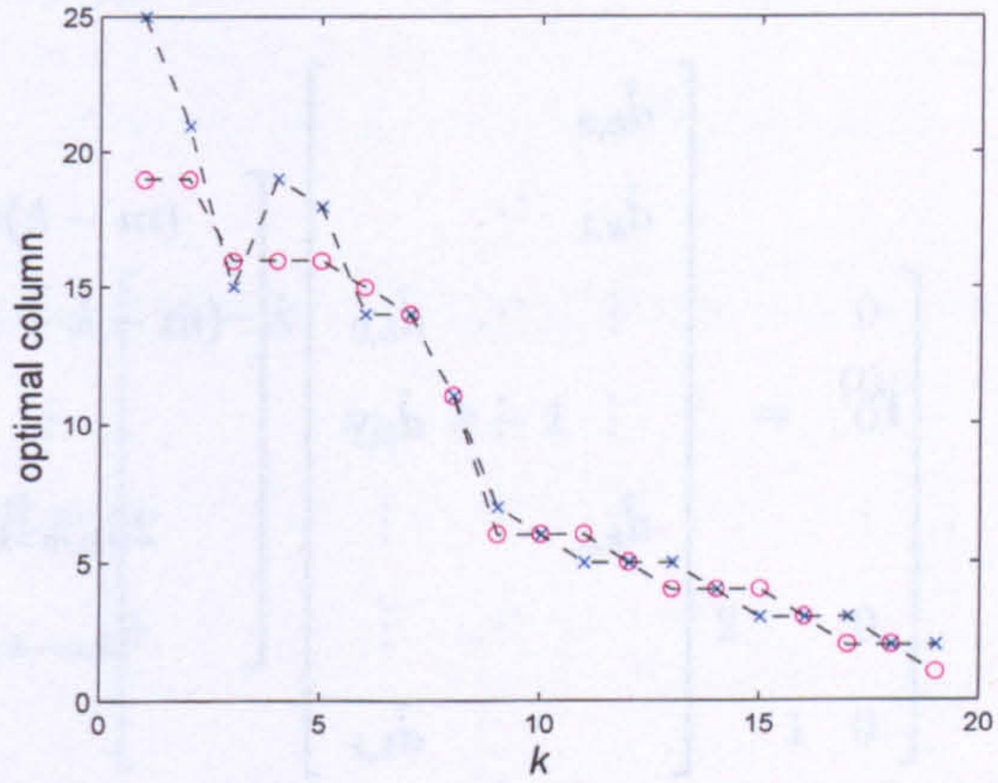



Figure 6.7: The optimal columns of $S_k(f_{\theta_0}, \alpha_0\, g_{\theta_0})$ for which the minimisations in (6.7) and (6.20) are achieved, using Method 1 $\times$, and Method 2 o, for Example 6.7.

Differentiating $\hat{f}(y)$ yields

$$\hat{f}^{(1)}(y) = \frac{d\Big(\hat{u}_k(y)\hat{d}_k(y)\Big)}{dy} = \hat{u}_k^{(1)}(y)\hat{d}_k(y) + \hat{u}_k(y)\hat{d}^{(1)}(y), \quad k = 1, \cdots, \hat{d}, \qquad (6.26)$$

which, in terms of a matrix-vector product, can be written as,

$$\hat{\mathbf{f}}^{(1)} = \begin{bmatrix} \hat{d}_{k,0} & & \\ \hat{d}_{k,1} & \ddots & \\ \vdots & \ddots & \hat{d}_{k,0} \\ \vdots & \vdots & \hat{d}_{k,1} \\ \hat{d}_{k,k} & \vdots & \vdots \\ & \ddots & \vdots \\ & & \hat{d}_{k,k} \end{bmatrix} \begin{bmatrix} (m-k)\hat{u}_{k,0} \\ (m-k-1)\hat{u}_{k,1} \\ \vdots \\ 2\hat{u}_{k,m-k-2} \\ \hat{u}_{k,m-k-1} \end{bmatrix} +$$

$$\begin{bmatrix} k\hat{d}_{k,0} & & & \\ (k-1)\hat{d}_{k,1} & \ddots & & \\ \vdots & \ddots & k\hat{d}_{k,0} & \\ \vdots & \vdots & (k-1)\hat{d}_{k,1} & \\ \hat{d}_{k,k} & \vdots & \vdots & \\ & \ddots & \vdots & \\ & & \hat{d}_{k,k} & \end{bmatrix} \begin{bmatrix} \hat{u}_{k,0} \\ \hat{u}_{k,1} \\ \vdots \\ \hat{u}_{k,m-k-2} \\ \hat{u}_{k,m-k-1} \end{bmatrix}$$

$$= F_k(\hat{d}_k)\hat{\mathbf{u}}_k^{(1)} + G_k(\hat{d}_k)\hat{\mathbf{u}}_k, \tag{6.27}$$

where $F_k(\hat{d}_k)$ and $G_k(\hat{d}_k)$ are Cauchy matrices whose entries are the coefficients of $\hat{d}_k(y)$, which is defined in (6.25), and its derivative, respectively, and $\hat{\mathbf{u}}_k$ and $\hat{\mathbf{u}}_k^{(1)}$ are the coefficient vectors of $\hat{u}_k(y)$, and its derivative, respectively, where $\hat{u}_k(y)$ is defined in (6.25),

$$\hat{\mathbf{u}}_k = \begin{bmatrix} \hat{u}_{k,0} & \hat{u}_{k,1} & \cdots & \hat{u}_{k,m-k} \end{bmatrix}^T \in \mathbb{R}^{m-k+1}, \tag{6.28}$$

$$\hat{\mathbf{u}}_k^{(1)} = \begin{bmatrix} (m-k)\hat{u}_{k,0} & (m-k-1)\hat{u}_{k,1} & \cdots & 2\hat{u}_{k,m-k-2} & \hat{u}_{k,m-k-1} \end{bmatrix}^T \in \mathbb{R}^{m-k}.$$

Let

$$R = \begin{bmatrix} m-k & & & & 0 \\ & m-k-1 & & & 0 \\ & & \ddots & & \vdots \\ & & & 2 & 0 \\ & & & 1 & 0 \end{bmatrix}, \tag{6.29}$$

and thus the vector $\hat{\mathbf{u}}_k^{(1)}$ can be expressed in terms of $\hat{\mathbf{u}}_k$ as follows,

$$\hat{\mathbf{u}}_k^{(1)} = R\hat{\mathbf{u}}_k. \tag{6.30}$$

Therefore, using (6.27) and (6.30), it can be verified that

$$\hat{\mathbf{f}}^{(1)} = \Big(F_k(\hat{d}_k)R + G_k(\hat{d}_k)\Big)\hat{\mathbf{u}}_k, \quad k = 1, \cdots, \hat{d}, \tag{6.31}$$

but it follows from (6.23) that

$$\hat{\mathbf{f}}^{(1)} = F_k(\hat{d}_k)\hat{\mathbf{v}}_k, \quad k = 1, \cdots, \hat{d}, \tag{6.32}$$

where $\hat{\mathbf{v}}_k$ is the vector of the coefficients of $\hat{v}_k(y)$ defined in (6.25),

$$\hat{\mathbf{v}}_k = \begin{bmatrix} \hat{v}_{k,0} & \hat{v}_{k,1} & \cdots & \hat{v}_{k,n-k} \end{bmatrix}^T \in \mathbb{R}^{n-k+1}. \tag{6.33}$$

Thus it follows from (6.31) and (6.32) that

$$F_k(\hat{d}_k)\hat{\mathbf{v}}_k - \Big(F_k(\hat{d}_k)R + G_k(\hat{d}_k)\Big)\hat{\mathbf{u}}_k = 0, \quad k = 1, \cdots, \hat{d}, \tag{6.34}$$

where $\hat{\mathbf{v}}_k$ and $\hat{\mathbf{u}}_k$ are defined in (6.33) and (6.28), respectively. It follows that if

$$\hat{e}_k = F_k(\hat{d}_k)\hat{\mathbf{v}}_k - \Big(F_k(\hat{d}_k)R + G_k(\hat{d}_k)\Big)\hat{\mathbf{u}}_k, \tag{6.35}$$

then for $k = 1, \cdots, m-1$

$$\begin{aligned} \|\hat{e}_k\| &= 0, \quad k = 1, \cdots, \hat{d}, \\ \|\hat{e}_k\| &> 0, \quad k = 1, \cdots, m-1. \end{aligned} \tag{6.36}$$

Since (6.34) is satisfied for $k = 1, \cdots, \hat{d}$ only, the value of $\hat{d}$ can be considered to be the largest value of $k$ for which (6.34) is satisfied.

An important difference between criterion (6.36), and the angle (6.3) and residual (6.4) criteria used by Methods 1 and 2 respectively, is that the entries of criterion (6.36), which is defined by (6.35), include the coefficients of $\hat{d}_k(y)$, and thus it requires initial estimates of the common divisors $\hat{d}_k(y)$. The computation of these initial estimates is now considered.

It follows from (6.24) that, for $k = 1, \cdots, \hat{d}$

$$\hat{v}_k(y)\hat{f}(y) = \hat{u}_k(y)\hat{f}^{(1)}(y), \tag{6.37}$$

where $\hat{u}_k(y)$, $\hat{v}_k(y)$ and $\hat{d}_k(y)$ are as defined in (6.25). The polynomial products in (6.37) can be written in matrix-vector form,

$$[C_k \quad D_k] \begin{bmatrix} \hat{\mathbf{v}}_k \\ -\hat{\mathbf{u}}_k \end{bmatrix} = S_k \begin{bmatrix} \hat{\mathbf{v}}_k \\ -\hat{\mathbf{u}}_k \end{bmatrix} = 0, \quad k = 1, \cdots, m-1, \tag{6.38}$$

where

$$\begin{aligned} \hat{\mathbf{v}}_k &= 0, \qquad k = \hat{d}+1, \cdots, m-1, \\ \hat{\mathbf{u}}_k &= 0, \qquad k = \hat{d}+1, \cdots, m-1, \end{aligned}$$

where $C_k = C_k(\hat{f}) \in \mathbb{R}^{(2m-k)(m-k)}$ and $D_k = D_k(\hat{f}^{(1)}) \in \mathbb{R}^{(2m-k)(m-k+1)}$ are Cauchy matrices whose entries are the coefficients of $\hat{f}(y)$ and $\hat{f}^{(1)}(y)$ respectively, $\hat{\mathbf{v}}_k$ and $\hat{\mathbf{u}}_k$ are defined in (6.33) and (6.28) respectively, and $S_k = S_k(\hat{f}, \hat{f}^{(1)}) \in \mathbb{R}^{(2m-k)\times(2m-2k+1)}$ is the $k$th Sylvester subresultant matrix of $\hat{f}(y)$ and $\hat{f}^{(1)}(y)$, which can be partitioned into a matrix $A_k \in \mathbb{R}^{(2m-k)\times(2m-2k)}$ and the vector $c_k \in \mathbb{R}^{(2m-k)}$,

$$S_k = \begin{bmatrix} c_k & A_k \end{bmatrix},$$

where $c_k$ is the first column of $S_k$ and $A_k$ is the matrix formed from the remaining columns of $S_k$. Since the degree of the GCD of $\hat{f}(y)$ and $\hat{f}^{(1)}(y)$ is equal to $\hat{d}$, $\hat{v}_{k,0} \neq 0$ for $k = 1, \cdots, \hat{d}$ and since exact data is being considered, $\hat{v}_{k,0}$ can be moved to the right hand side without loss of generality, that is, $\hat{v}_{k,0} = -1$. Thus using the partitioned form of $S_k$ and the condition $\hat{v}_{k,0} = -1$, allows (6.38) to be written as,

$$A_k x_k = c_k, \qquad k = 1, \cdots, \hat{d},$$

where

$$A_k x_k \neq c_k, \qquad k = \hat{d}+1, \cdots, m-1,$$

and for $k = 1, \cdots, m-1$,

$$x_k = A_k^{\dagger} c_k = \begin{bmatrix} \hat{v}_{k,1} & \cdots & \hat{v}_{k,m-k-1} & -\hat{u}_{k,0} & \cdots & -\hat{u}_{k,m-k} \end{bmatrix}^T \in \mathbb{R}^{2m-2k}. \quad (6.39)$$

Estimates for the vectors $\hat{\mathbf{u}}_k$ and $\hat{\mathbf{v}}_k$ can be calculated from (6.39), and estimates for $\hat{\mathbf{d}}_k$ can be obtained from $\hat{\mathbf{u}}_k$ and $\hat{\mathbf{v}}_k$. In particular, the equations in (6.23) can be combined in one matrix-vector form,

$$\begin{bmatrix} Q_{k,1} \\ Q_{k,2} \end{bmatrix} \hat{\mathbf{d}}_k = \begin{bmatrix} \hat{\mathbf{f}} \\ \hat{\mathbf{f}}^{(1)} \end{bmatrix}, \quad k = 1, \cdots, m-1,$$

where $Q_{k,1}$ and $Q_{k,2}$ are Cauchy matrices whose entries are the coefficients of $\hat{\mathbf{u}}_k$ and $\hat{\mathbf{v}}_k$ respectively, that are calculated from (6.39), and $\hat{\mathbf{f}}$ and $\hat{\mathbf{f}}^{(1)}$ are the vectors of the coefficients of $\hat{f}(y)$ and $\hat{f}^{(1)}(y)$, respectively. Thus, $\hat{\mathbf{d}}_k$ can be obtained from,

$$\hat{\mathbf{d}}_k = \begin{bmatrix} Q_{k,1} \\ Q_{k,2} \end{bmatrix}^{\dagger} \begin{bmatrix} \hat{\mathbf{f}} \\ \hat{\mathbf{f}}^{(1)} \end{bmatrix}, \quad k = 1, \cdots, m-1.$$

### 6.4.2 AGCD degree of an inexact polynomial and its derivative

This section extends Section 6.4.1 to the situation that occurs when the inexact polynomials $f(y)$ and its derivative $f^{(1)}(y)$, whose exact forms have a GCD of degree

$d$, are considered. It follows that, for $k = 1, \cdots, d$, (6.23) must be replaced by the approximations

$$f(y) \approx u_k(y)d_k(y), \quad \text{and} \quad f^{(1)}(y) \approx v_k(y)d_k(y),$$

and the derivative constraint (6.26) must be replaced by

$$f^{(1)}(y) \approx u_k^{(1)}(y)d_k(y) + u_k(y)d_k^{(1)}(y). \tag{6.40}$$

where

$$\begin{aligned} u_k(y) &= \textstyle\sum_{i=0}^{m-k} u_{k,i}y^{m-k-i}, \\ v_k(y) &= \textstyle\sum_{i=0}^{n-k} v_{k,i}y^{n-k-i}, \\ d_k(y) &= \textstyle\sum_{i=0}^{k} d_{k,i}y^{k-i}. \end{aligned}$$

It has been shown in Chapter 4 that it is necessary to process $f(y)$ and $f^{(1)}(y)$ before an AGCD is computed. In particular, it is required to normalise $f(y)$ and $f^{(1)}(y)$ by the geometric means of their coefficients, and thus

$$f(y) = \sum_{i=0}^{m} \bar{a}_i y^{m-i}, \quad \bar{a}_i = \frac{a_i}{\left(\prod_{j=0}^{m} |a_j|\right)^{\frac{1}{m+1}}}, \quad \prod_{i=0}^{m} |\bar{a}_i| = 1,$$

and

$$g(y) = \sum_{i=0}^{m-1} \bar{b}_i y^{m-i-1}, \quad \bar{b}_i = \frac{(m-i)a_i}{\left(\prod_{j=0}^{m-1} |(m-j)a_j|\right)^{\frac{1}{m}}}, \quad \prod_{i=0}^{m-1} |\bar{b}_i| = 1,$$

are considered, where $g(y)$ is proportional to, and not equal to, $f^{(1)}(y)$. Specifically, it can be verified that

$$\bar{b}_i = (m-i)\bar{a}_i\lambda, \qquad \lambda = \frac{\left(\frac{|a_m|}{m!}\right)^{\frac{1}{m}}}{\left(\prod_{j=0}^{m}|a_j|\right)^{\frac{1}{m(m+1)}}}, \quad i = 0, \cdots, m-1. \tag{6.41}$$

Thus,

$$g(y) = \lambda f^{(1)}(y).$$

Scaling $f(y)$ and $g(y)$ by the scaling factors $\alpha_0$ and $\theta_0$ as described in Chapter 4 yields the polynomials

$$f_{\theta_0}(w) = \sum_{i=0}^{m} a_i^* w^{m-i} \quad \text{and} \quad \alpha_0 g_{\theta_0}(w) = \alpha_0 \sum_{i=0}^{m-1} b_i^* w^{m-i-1}, \tag{6.42}$$

whose coefficients are

$$a_i^* = \bar{a}_i\theta_0^{m-i} \quad \text{and} \quad b_i^* = \bar{b}_i\theta_0^{m-i-1},$$

where $\alpha_0$ and $\theta_0$ are the optimal values of $\alpha$ and $\theta$, respectively, whose values are obtained by solving the LP problem (4.7). It also follows from (6.41) and (6.42) that

$$g_{\theta_0}(w) = \lambda f_{\theta_0}^{(1)}(w), \tag{6.43}$$

which establishes the relation between $g_{\theta_0}(w)$ and $f_{\theta_0}^{(1)}(w)$.

It is assumed that $f(y)$ is inexact, and thus for $k = 1, \cdots, d$, an approximate common

divisor $d_k(w)$ of $f(_{\theta_0}w)$ and $g_{\theta_0}(w)$, of degree $k$, satisfies

$$f_{\theta_0}(w) \approx u_{k,\theta_0}(w)d_{k,\theta_0}(w), \quad \alpha_0 g_{\theta_0}(w) \approx v_{k,\theta_0}(w)d_{k,\theta_0}(w), \quad k = 1, \cdots, d, \quad (6.44)$$

where,

$$u_{k,\theta_0}(w) = \sum_{i=0}^{m-k} \tilde{u}_{k,i} w^{m-k-i}, \quad \tilde{u}_{k,i} = \tilde{c}_{k,i}\theta_0^{m-k-i} \quad (6.45)$$

$$v_{k,\theta_0}(w) = \sum_{i=0}^{n-k} \tilde{v}_{k,i} w^{n-k-i}, \quad \tilde{v}_{k,i} = \tilde{e}_{k,i}\theta_0^{n-k-i} \quad (6.46)$$

$$d_{k,\theta_0}(w) = \sum_{i=0}^{k} \tilde{d}_{k,i} w^{k-i}, \quad \tilde{d}_{k,i} = \tilde{r}_{k,i}\theta_0^{k-i}. \quad (6.47)$$

Also, scaling (6.40) by $\theta_0$ yields,

$$\begin{aligned} f^{(1)}_{\theta_0}(w) &\approx \left(\sum_{i=0}^{k} \left(\tilde{r}_{k,i}\theta_0^{k-i}\right) w^{k-i}\right) u^{(1)}_{k,\theta_0}(w) + \\ &\quad \left(\sum_{i=0}^{k-1} \left((k-i)\tilde{r}_{k,i}\theta_0^{k-i-1}\right) w^{k-i-1}\right) u_{k,\theta_0}(w) \\ &= \left(\sum_{i=0}^{k} \left(\tilde{r}_{k,i}\theta_0^{k-i}\right) w^{k-i}\right) u^{(1)}_{k,\theta_0}(w) + \\ &\quad \left(\sum_{i=0}^{k-1} \left(\tilde{s}_{k,i}\theta_0^{k-i-1}\right) w^{k-i-1}\right) u_{k,\theta_0}(w), \end{aligned}$$

using (6.45), (6.46) and (6.47), where the coefficients of $d^{(1)}_{k,\theta_0}(w)$ are

$$\tilde{s}_{k,i} = (k-i)\tilde{r}_{k,i}, \qquad i = 0, \ldots, k-1.$$

In terms of a matrix-vector product, the coefficients of $f_{\theta_0}^{(1)}(w)$ can be written as,

$$
\begin{aligned}
\mathbf{f}_{\theta_0}^{(1)} \approx & \begin{bmatrix} \tilde{r}_{k,0}\theta_0^k & & \\ \tilde{r}_{k,1}\theta_0^{k-1} & \ddots & \\ \vdots & \ddots & \tilde{r}_{k,0}\theta_0^k \\ \vdots & \vdots & \tilde{r}_{k,1}\theta_0^{k-1} \\ \tilde{r}_{k,k} & \vdots & \vdots \\ & \ddots & \vdots \\ & & \tilde{r}_{k,k} \end{bmatrix} \begin{bmatrix} (m-k)\tilde{c}_{k,0}\theta_0^{m-k-1} \\ (m-k-1)\tilde{c}_{k,1}\theta_0^{m-k-2} \\ \vdots \\ 2\tilde{c}_{k,m-k-2}\theta_0 \\ \tilde{c}_{k,m-k-1} \end{bmatrix} + \\
& \begin{bmatrix} \tilde{s}_{k,0}\theta_0^{k-1} & & \\ \tilde{s}_{k,1}\theta_0^{k-2} & \ddots & \\ \vdots & \ddots & \tilde{s}_{k,0}\theta_0^{k-1} \\ \vdots & \vdots & \tilde{s}_{k,1}\theta_0^{k-2} \\ \tilde{s}_{k,k-1} & \vdots & \vdots \\ & \ddots & \vdots \\ & & \tilde{s}_{k,k-1} \end{bmatrix} \begin{bmatrix} \tilde{c}_{k,0}\theta_0^{m-k} \\ \tilde{c}_{k,1}\theta_0^{m-k-1} \\ \vdots \\ \tilde{c}_{k,m-k-1}\theta_0 \\ \tilde{c}_{k,m-k} \end{bmatrix} \\
= & \; L_{k,\theta_0}\tilde{\mathbf{c}}_{k,\theta_0}^{(1)} + M_{k,\theta_0}\tilde{\mathbf{c}}_{k,\theta_0}, \qquad (6.48)
\end{aligned}
$$

where $\tilde{\mathbf{c}}_{k,\theta_0}$ is the vector of coefficients of $u_{k,\theta_0}(w)$ defined in (6.45),

$$
\tilde{\mathbf{c}}_{k,\theta_0} = \begin{bmatrix} \tilde{c}_{k,0}\theta_0^{m-k} & \tilde{c}_{k,1}\theta_0^{m-k-1} & \cdots & \tilde{c}_{k,m-k-1}\theta_0 & \tilde{c}_{k,m-k} \end{bmatrix}^T \in \mathbb{R}^{m-k+1},
$$

and $\tilde{\mathbf{c}}^{(1)}_{k,\theta_0}$ is the vector of coefficients of the derivative with respect to $w$,

$$L_{k,\theta_0} = \begin{bmatrix} \tilde{r}_{k,0}\theta_0^k & & \\ \tilde{r}_{k,1}\theta_0^{k-1} & \ddots & \\ \vdots & \ddots & \tilde{r}_{k,0}\theta_0^k \\ \vdots & \vdots & \tilde{r}_{k,1}\theta_0^{k-1} \\ \tilde{r}_{k,k} & \vdots & \vdots \\ & \ddots & \vdots \\ & & \tilde{r}_{k,k} \end{bmatrix} \in \mathbb{R}^{m\times(m-k)},$$

and

$$M_{k,\theta_0} = \begin{bmatrix} \tilde{s}_{k,0}\theta_0^{k-1} & & \\ \tilde{s}_{k,1}\theta_0^{k-2} & \ddots & \\ \vdots & \ddots & \tilde{s}_{k,0}\theta_0^{k-1} \\ \vdots & \vdots & \tilde{s}_{k,1}\theta_0^{k-2} \\ \tilde{s}_{k,k-1} & \vdots & \vdots \\ & \ddots & \vdots \\ & & \tilde{s}_{k,k-1} \end{bmatrix} \in \mathbb{R}^{m\times(m-k+1)}.$$

It is readily verified that $\tilde{\mathbf{c}}^{(1)}_{k,\theta_0}$ and $\tilde{\mathbf{c}}_{k,\theta_0}$ are related by the diagonal matrix $R \in \mathbb{R}^{(m-k)\times(m-k+1)}$ which is defined in (6.29),

$$\theta_0\tilde{\mathbf{c}}^{(1)}_{k,\theta_0} = R\tilde{\mathbf{c}}_{k,\theta_0}. \tag{6.49}$$

It follows from (6.48) and (6.49) that

$$\mathbf{f}_{\theta_0}^{(1)} \approx \frac{1}{\theta_0}\Big(L_{k,\theta_0}R + \theta_0 M_{k,\theta_0}\Big)\tilde{\mathbf{c}}_{k,\theta_0}, \tag{6.50}$$

and, from (6.43) and (6.44) that

$$(\alpha_0\lambda)\, f_{\theta_0}^{(1)}(w) = v_{k,\theta_0}(w) d_{k,\theta_0}(w), \qquad \lambda = \frac{\left(\frac{|a_m|}{m!}\right)^{\frac{1}{m}}}{\left(\prod_{j=0}^{m}|a_j|\right)^{\frac{1}{m(m+1)}}},$$

and thus the vector of coefficients $(\alpha_0\lambda)\,\mathbf{f}_{\theta_0}^{(1)}$ of $(\alpha_0\lambda)\, f_{\theta_0}^{(1)}(w)$ can also be approximated by

$$(\alpha_0\lambda)\,\mathbf{f}_{\theta_0}^{(1)} \approx L_{k,\theta_0}\tilde{\mathbf{e}}_{k,\theta_0}, \tag{6.51}$$

where $\tilde{\mathbf{e}}_{k,\theta_0}$ is the vector of coefficients of $v_{k,\theta_0}(w)$ defined in (6.46),

$$\tilde{\mathbf{e}}_{k,\theta_0} = \begin{bmatrix} \tilde{e}_{k,0}\theta_0^{m-k-1} & \tilde{e}_{k,1}\theta_0^{m-k-2} & \cdots & \tilde{e}_{k,m-k-2}\theta_0 & \tilde{e}_{k,m-k-1} \end{bmatrix}^T \in \mathbb{R}^{m-k},$$

The combination of (6.50) and (6.51) yields

$$\left(\frac{\theta_0}{\alpha_0\lambda}\right) L_{k,\theta_0}\tilde{\mathbf{e}}_{k,\theta_0} - \Big(L_{k,\theta_0}R + \theta_0 M_{k,\theta_0}\Big)\tilde{\mathbf{c}}_{k,\theta_0} \approx 0. \tag{6.52}$$

It follows from (6.52) that

$$\begin{bmatrix} V_{k,\theta_0} & L_{k,\theta_0}R + U_{k,\theta_0} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{v}}_{k,\theta_0} \\ -\tilde{\mathbf{u}}_{k,\theta_0} \end{bmatrix} \approx 0, \tag{6.53}$$

where $V_{k,\theta_0} = \left(\frac{\theta_0}{\alpha\lambda}\right) L_{k,\theta_0}$ and $U_{k,\theta_0} = \theta_0 M_{k,\theta_0}$. This approximation allows the error measure,

$$e_k = \frac{\left\| V_{k,\theta_0}\tilde{\mathbf{v}}_{k,\theta_0} - \left(L_{k,\theta_0}R + U_{k,\theta_0}\right)\tilde{\mathbf{u}}_{k,\theta_0} \right\|}{\|V_{k,\theta_0}\tilde{\mathbf{v}}_{k,\theta_0}\| + \left\| \left(L_{k,\theta_0}R + U_{k,\theta_0}\right)\tilde{\mathbf{u}}_{k,\theta_0} \right\|}, \qquad k = 1, \cdots, m-1, \tag{6.54}$$

to be calculated for each value of $k$. The normalisation in (6.54) guarantees that $e_k$ is always finite and independent of any arbitrary scaling. The value of $k$, for which the error measure (6.54) achieves its minimum value is equal to the degree $d$ of the AGCD. The reason for this follows from the following observations (see Section 3.2 for the case when an exact polynomial is considered):

***Observation** 1: For $k = 1, \cdots, d-1$, the solutions $(\tilde{\mathbf{v}}_{k,\theta_0}, \tilde{\mathbf{u}}_{k,\theta_0})$ of (6.53) are, with high probability, coefficients of polynomial approximation to rational functions. Therefore, $e_k$ in (6.54) is large.*

***Observation** 2: For $k = d$, (6.53) is satisfied with a minimum error, since there is a unique approximate solution $(\tilde{\mathbf{v}}_{d,\theta_0}, \tilde{\mathbf{u}}_{d,\theta_0})$, corresponding to an AGCD. Therefore, $e_k$ in (6.54) is small.*

***Observation** 3: For $k = d+1, \cdots, m-1$, the coefficient matrix in (6.53) is far from singularity because there does not exist an AGCD of degree greater than $d$. Therefore, $e_k$ in (6.54) is large.*

It therefore follows that the index $k$ for which the error in (6.53) is a minimum is

equal to the degree $d$ of an AGCD of $f(y)$ and $g(y)$. Let $d_e = d$ denote the computed degree of an AGCD, using the error measure (6.54), then

$$d_e = \{k : e_k \longrightarrow \min; k = 1, \cdots, m-1\}.$$

The computation of the error measure (6.54) requires estimates of the common divisors $d_{k,\theta_0}(w)$, $k = 1, \cdots, m-1$. These estimates require that the approximation (6.5)

$$A_{k,i}x_{k,i} \approx c_{k,i}, \qquad k = 1, \cdots, m-1;\ i = 1, \cdots, 2m-2k+1, \tag{6.55}$$

be considered, where $c_{k,i}$ is the $i^{th}$ column of $S_k(f_{k,\theta_0}, \alpha_0 g_{k,\theta_0})$, $A_{k,i}$ is the matrix formed by the remaining columns of $S_k(f_{k,\theta_0}, \alpha_0 g_{k,\theta_0})$, and the vector $x_{k,i}$ contains the coefficients of the quotient polynomials $u_{k,\theta_0}(w)$ and $v_{k,\theta_0}(w)$ defined in (6.45) and (6.46), respectively. It follows from (6.44) that the estimates of the common divisors $d_{k,\theta_0}(w)$, $k = 1, \cdots, m-1$ can be obtained from the least squares solutions of

$$\begin{bmatrix} Q_{k,1} \\ Q_{k,2} \end{bmatrix} \mathbf{d}_{k,\theta_0} = \begin{bmatrix} \mathbf{f}_{\theta_0} \\ \alpha_0 \mathbf{g}_{\theta_0} \end{bmatrix} \quad k = 1, \cdots, m-1,$$

where $Q_{k,1}$ and $Q_{k,2}$ are Cauchy matrices whose entries are the coefficients of $u_{k,\theta_0}(w)$ and $v_{k,\theta_0}(w)$ respectively, that are calculated from $x_{k,i}$, and $\mathbf{f}_{\theta_0}$ and $\mathbf{g}_{\theta_0}$ are the vectors of the coefficients of $f_{\theta_0}(w)$ and $g_{\theta_0}(w)$, respectively. The indices $(k, i)$ must be calculated such that the error in (6.55) is small. It has been shown in Section 6.2 that the index $i$ of the optimal column of $S_k(f_{k,\theta_0}, \alpha_0 g_{k,\theta_0})$, for the computation of the degree of the AGCD of $f_{k,\theta_0}(w)$ and $\alpha_0 g_{k,\theta_0}$, for each value of $k = 1, \cdots, m-1$,

can be computed based on the angle between the subspace spanned by $c_{k,i}$ and the subspace spanned by $A_{k,i}$, or based on the residual of the approximation in (6.55), as shown in Sections 6.2.2 and 6.2.3, respectively. Therefore, the error measure $e_k$ in (6.54) can be computed using:

1. The criterion that is based on the first principal angle, to yield $e_{k,t}$.

2. The criterion that is based on the residual, to yield $e_{k,r}$.

Thus based on the use of the above two criteria, the error measure (6.54) provides two estimates $d_{e,t}$ and $d_{e,r}$ of $d$, respectively, and since it is assumed that the error is small, the desire is that both estimates are equal.

## 6.5 Examples

This section provides two examples that demonstrate the use of the three methods described in this chapter for the computation of the degree of an AGCD of an inexact polynomial and its derivative. In particular,

1. The first method uses first principal angle and it is described in Sections 6.2.2.

2. The second method uses the residual of an algebraic form derived from the Sylvester matrix of two inexact polynomials and it is described in Section 6.2.3.

3. The third method is discussed in the previous section. It uses a constraint between a polynomial and its derivative.

The first two methods are applicable to any pair of polynomials, whereas the third method is only applicable for a polynomial for a polynomial and its derivative. Moreover, it is noted above that the this method uses the error measure $e_k$ in (6.54),

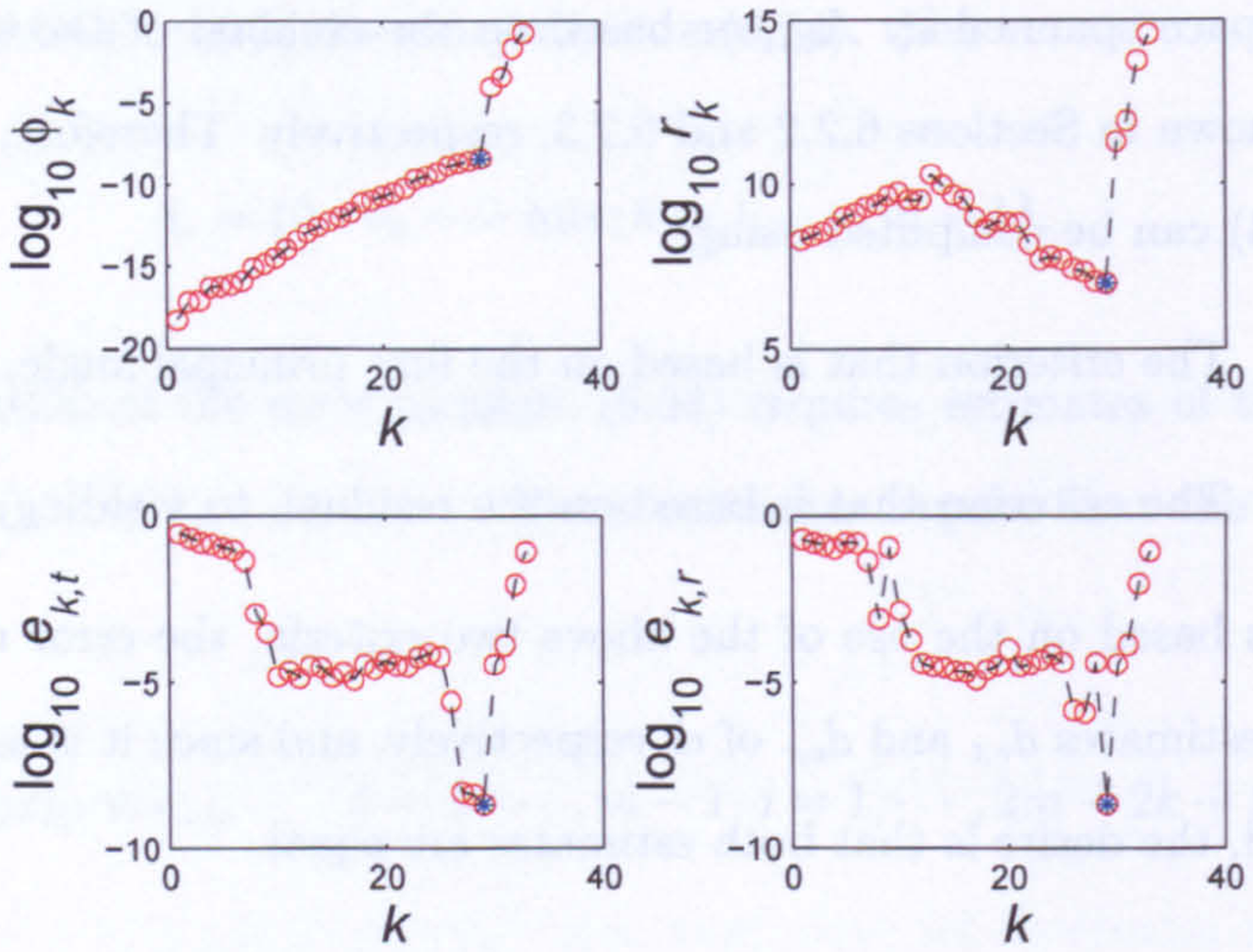



Figure 6.8: The variations with $k$, of $\log \phi_k$, $\log r_k$, $\log e_{k,t}$ and $\log e_{k,r}$ for Example 6.8, where $*$ denotes the exact GCD degree $\hat{d}$.

which have two forms $e_{k,t}$ and $e_{k,r}$, and the computation of both of these forms are considered in these examples.

**Example 6.8.** Consider the exact polynomial

$$\hat{f}(y) = (y - 9.2393)^{10}(y + 7.8313)^{8}(y + 9.2777)^{7}(y - 2.3618)^{6}(y - 1.3429)^{3},$$

for which the GCD of $\hat{f}(y)$ and $\hat{f}^{(1)}(y)$ is equal to

$$\hat{q}(y) = (y - 9.2393)^{9}(y + 7.8313)^{7}(y + 9.2777)^{6}(y - 2.3618)^{5}(y - 1.3429)^{2},$$

and the degree $\hat{d}$ of the GCD of $\hat{f}(y)$ and $\hat{f}^{(1)}(y)$ is equal to 29. Componentwise

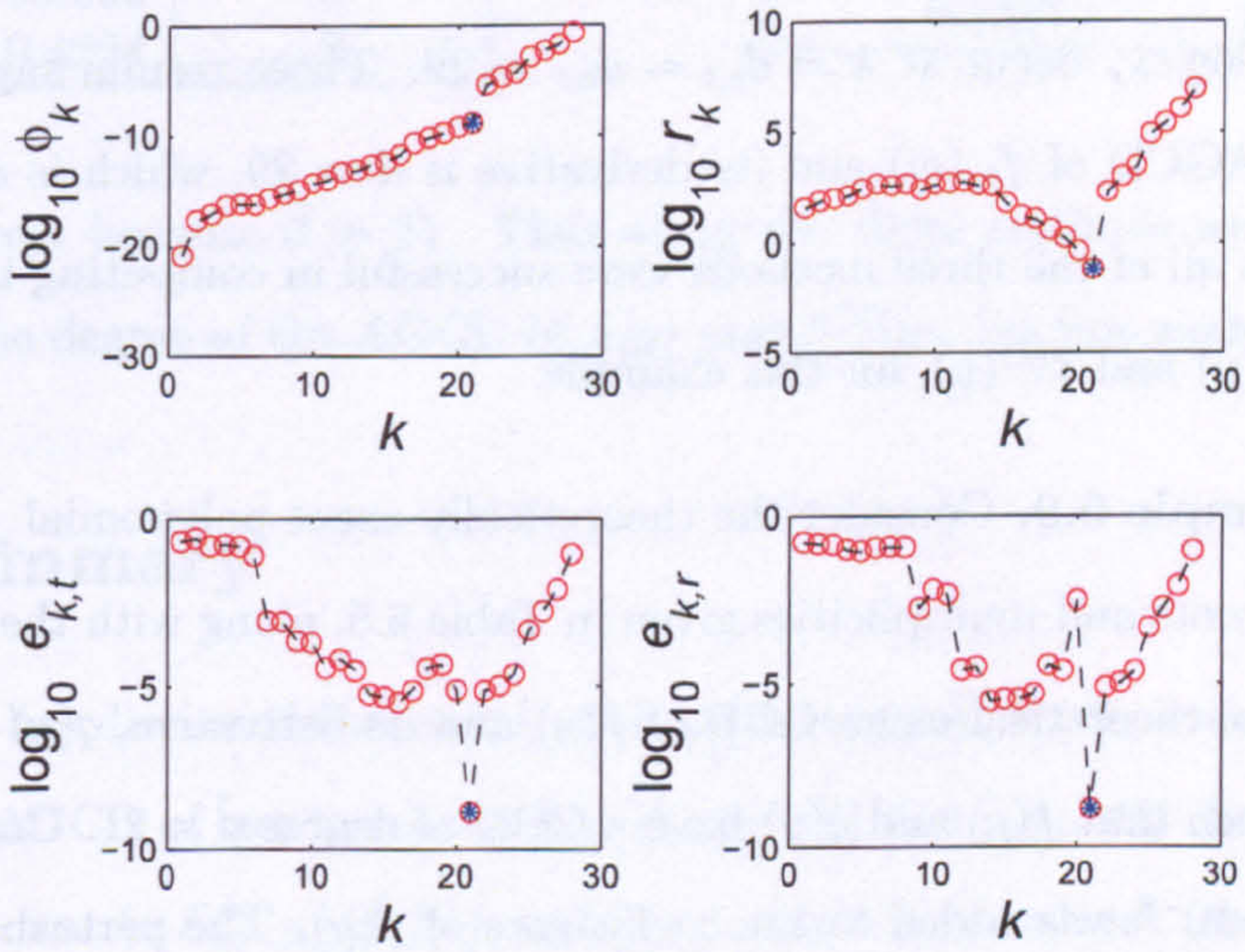



Figure 6.9: The variations with $k$, of $\log \phi_k$, $\log r_k$, $\log e_{k,t}$ and $\log e_{k,r}$ for Example 6.9, where $*$ denotes the exact GCD degree $\hat{d}$.

noise with $\varepsilon_c = 10^{-8}$ was added to the coefficients of $\hat{f}(y)$. The perturbed polynomial $f(y)$ and its derivative $f^{(1)}(y)$ were then normalised by the geometric means of their coefficients and preprocessed by the optimal scaling parameters $\alpha_0 = 1.9056$ and $\theta_0 = 5.8975$. The values of $\phi_k$, $r_k$, and $e_k$, defined in (6.7), (6.20) and (6.53) respectively, were computed for $k = 1, \cdots, 33$, and the results of the variations of $\log \phi_k$, $\log r_k$ and $\log e_k$ with $k$, are shown in Figure 6.8 It can be seen that the maximum changes in $\log \phi_k$ and $\log r_k$ occur at $k = d_\phi = d_r = 29$, and the minimum values of $\log e_{k,t}$ and $\log e_{k,r}$ occur at $k = d_{e,t} = d_{e,r} = 29$. These results suggest that the degree of the AGCD of $f_{\theta_0}(w)$ and its derivative is $d = 29$, which is correct because $\hat{d} = 29$. Thus all of the three methods were successful in computing the degree of the AGCD of $f(y)$ and $f^{(1)}(y)$, for this example. □

**Example 6.9.** Consider the theoretically exact polynomial $\hat{f}(y)$ that is specified by the roots and multiplicities given in Table 6.5, along with the roots and multiplicities of the theoretical exact GCD of $\hat{f}(y)$ and its derivative, $q(y) = \mathrm{GCD}(\hat{f}, \hat{f}^{(1)})$. It can be seen that $\hat{f}(y)$ and $\hat{g}(y)$ have a GCD of degree $\hat{d} = 21$. Componentwise noise with $\varepsilon_c = 10^{-8}$ was added to the coefficients of $\hat{f}(y)$. The perturbed polynomial $f(y)$ and its derivative $f^{(1)}(y)$ were then normalised by the geometric means of their coefficients and preprocessed by the optimal scaling parameters $\alpha_0 = 1.3862$ and $\theta_0 = 2.3838$. The values of $\phi_k$, $r_k$, and $e_k$, defined in (6.7), (6.20) and (6.53) respectively, were computed for $k = 1, \cdots, 28$, and the results of the variations of $\log \phi_k$, $\log r_k$ and $\log e_k$ with $k$, are shown in Figure 6.9.

It can be seen that the maximum changes in $\log \phi_k$ and $\log r_k$ occur at $k = d_\phi = d_r = 21$, and the minimum values of $\log e_{k,t}$ and $\log e_{k,r}$ occur at $k = d_{e,t} = d_{e,r} = 21$. These results suggest that the degree of the AGCD of $f_{\theta_0}(w)$ and its derivative is $d = 21$,

Table 6.5: The roots and multiplicities of $\hat{f}(y)$ and $\hat{q}(y) = \mathrm{GCD}(\hat{f}, \hat{f}^{(1)})$ for Example 6.9.

| Root of $\hat{f}(y)$ | Multiplicity |
|---|---|
| -7.5947 | 6 |
| 1.4923 | 5 |
| 0.63371 | 5 |
| 5.4862 | 4 |
| -3.3076 | 3 |
| -3.067 | 2 |
| 2.5090 | 2 |
| 0.4224 | 2 |

| Root of $\hat{q}(y)$ | Multiplicity |
|---|---|
| -7.5947 | 5 |
| 1.4923 | 4 |
| 0.63371 | 4 |
| 5.4862 | 3 |
| -3.3076 | 2 |
| -3.067 | 1 |
| 2.5090 | 1 |
| 0.4224 | 1 |

which is correct because $\hat{d} = 21$. Thus all of the three methods were successful in computing the degree of the AGCD of $f(y)$ and $f^{(1)}(y)$, for this example. □

## 6.6 Summary

This chapter has discussed three rank evaluation methods for the computation of the degree of an AGCD of two inexact polynomials $f(y)$ and $g(y)$. All three methods use the Sylvester matrix $S(f, g)$ of $f(y)$ and $g(y)$ and its subresultant matrices, but they differ in the criteria derived from the Sylvester resultant matrix.

The first method uses the first principal angle between the space spanned by one column of $S_k(f, g)$ and the space spanned by the remaining columns of $S_k(f, g)$, where $k$ denotes the order of the subresultant matrix. The second method uses the residual of an approximate linear algebraic equation derived from $S_k(f, g)$. These two methods differ in the criteria used for the computation of the degree $d$ of an AGCD of two polynomials, and they are applicable to any pair of polynomials. The third method, on the other hand, uses the constraint between a polynomial and its derivative and

it is only applicable for a polynomial $f(y)$ and its derivative $f^{(1)}(y)$. In this method an error measure that is derived from this constraint is used for the computation of $d$. It has been shown that this error measure provides two estimates of $d$ based on the criterion used to form $S_k(f, f^{(1)})$.

# Chapter 7

# The computation of an AGCD

The computation of the GCD of two polynomials has several applications, including the computation of multiple roots of a polynomial. However, it was noted in Chapter 5 that the GCD is not defined if the coefficients of the polynomial are only known within a limited accuracy or the computations are performed in a floating point environment, in which case only an AGCD can be defined. This chapter describes two methods for the computation of an AGCD of two inexact polynomials $f = f(y)$ and $g = g(y)$. These methods apply the method of structured non-linear total least norm (SNTLN) to the computation of a structured low rank approximation $S(\tilde{f}, \tilde{g})$ of the Sylvester matrix $S(f, g)$ of the inexact polynomials $f(y)$ and $g(y)$, from which an AGCD of $f(y)$ and $g(y)$ can be computed. The first method applies the method of SNTLN to the Sylvester matrix $S(f, g)$, whereas the second method applies the method of SNTLN to the approximate polynomial factorisation (APF) of $f(y)$ and $g(y)$. Both methods require that the degree of an AGCD be first determined, after which the coefficients of the AGCD are computed. The computation of the degree $d$ of an AGCD was discussed in Chapter 6, and it is therefore assumed that $d$ is known.

It is required to compute the coefficients of the AGCD, given $d$, and this topic is addressed in this chapter.

The first section in this chapter demonstrates the relation between the structured low rank approximation and the computation of an AGCD of two inexact polynomials. Section 7.2 describes the application of the method of SNTLN to the Sylvester matrix, for the computation of a structured low rank approximation of the Sylvester matrix of $f(y)$ and $g(y)$. Examples that demonstrate the theory in Section 7.2 are given in Section 7.3. Section 7.4 describes the application of the method of SNTLN to the APF of two polynomials for the computation of structured low rank approximations of the Sylvester matrix of $f(y)$ and $g(y)$. Examples that demonstrate the theory in Section 7.4 are given in Section 7.5.

## 7.1 Structured low rank approximation of the Sylvester matrix

It is recalled from Theorem 3.1 that the degree of the GCD of the exact polynomials $\hat{f} = \hat{f}(y)$ and $\hat{g} = \hat{g}(y)$, defined in (3.1), equals the rank deficiency of their Sylvester matrix $S(\hat{f}, \hat{g})$, and that the coefficients of the GCD lie in the last non-zero row of $S(\hat{f}, \hat{g})^T$, after reducing it to an upper triangular form.

The case is different when inexact polynomials $f(y)$ and $g(y)$ are considered, since their Sylvester matrix $S(f, g)$ has, with high probability, full rank. The coefficients of $f(y)$ and $g(y)$ can be perturbed such that their perturbed forms $\tilde{f}(y) = f(y) + \delta f(y)$

and $\tilde{g}(y) = g(y) + \delta g(y)$ have a non-constant GCD, that is

$$\text{rank}\, S(\tilde{f}, \tilde{g}) = \text{rank}\, (S(f, g) + S(\delta f, \delta g)) \; < \; (m + n), \tag{7.1}$$

where, the perturbations $\delta f$ and $\delta g$ can be obtained using the method of SNTLN. The GCD of $\tilde{f}(y)$ and $\tilde{g}(y)$ is an AGCD with respect to the inexact polynomials $f(y)$ and $g(y)$. The underlying principle of using the structured low rank approximation methods for computing an AGCD of two inexact polynomials can therefore be summarised as follows:

*Given the Sylvester matrix of inexact polynomials $f(y)$ and $g(y)$, compute the structured perturbation Sylvester matrix $S(\delta f, \delta g)$ such that (7.1) is satisfied.*

Once $S(\tilde{f}, \tilde{g})$ is calculated, an AGCD of $f(y)$ and $g(y)$ can be obtained from the corrected polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$ using the properties of the Sylvester matrix in Theorem 3.1. Specifically, an AGCD of $f(y)$ and $g(y)$ is, up to a non-zero multiplier, defined to be equal to the GCD of $\tilde{f}(y)$ and $\tilde{g}(y)$. Thus the discussion above suggests that there is a close relation between the computation of an AGCD of $f(y)$ and $g(y)$ and the computation of a structured low rank approximation $S(\tilde{f}, \tilde{g})$ of $S(f, g)$. The computation of an AGCD using the structured low rank approximation methods, however, requires that the degree of an AGCD be defined, and it is therefore assumed that the methods in Chapter 6 have been used.

The use of the method of SNTLN in constructing low rank approximations of $S(f, g)$ using both the Sylvester matrix and APF of two given inexact polynomials, is considered in this chapter.

## 7.2 Calculating an AGCD using the Sylvester matrix

This section considers the use of the method of SNTLN in constructing a low rank approximation of the Sylvester matrix $S(f,g)$ of the inexact polynomials $f(y)$ and $g(y)$, whose theoretical exact forms have a non-constant GCD.

It is assumed that the given polynomials have been preprocessed using the methods in Chapter 4, and thus the polynomials $f_\theta(w)$ and $g_\theta(w)$ defined in (4.5) and (4.6), respectively, are considered. However, the method of the SNTLN requires that the values of the parameters $\alpha$ and $\theta$ be calculated iteratively with initial values of $\alpha_0$ and $\theta_0$, respectively, whose values are obtained from solving the LP problem (4.7). The following forms are therefore used,

$$f_\theta(w) = \sum_{i=0}^{m} \left(\frac{a_i^*}{\theta_0^{m-i}}\right)\theta^{m-i}w^{m-i} \qquad \text{and} \qquad g_\theta(w) = \sum_{i=0}^{n} \left(\frac{b_i^*}{\theta_0^{n-i}}\right)\theta^{m-i}w^{n-i},$$

where

$$a_i^* = \bar{a}_i\theta_0^{m-i} \qquad \text{and} \qquad b_i^* = \bar{b}_i\theta_0^{n-i},$$

where $\bar{a}_i$ and $\bar{b}_i$ are defined in (4.2) and (4.3), respectively, and the value $\theta_0$ of $\theta$ is retained in the denominators of $a_i^*$ and $b_i^*$ to simplify the update process between the successive iterations in the method of SNTLN. The method of STLN can also be used in constructing a low rank approximation of $S(f,g)$, in which the parameters $\alpha$ and $\theta$ are hold constant. However, it shown in [79] that the method SNTLN provides better approximation of an AGCD of two inexact polynomials.

It is assumed that deg $\mathrm{AGCD}(f_\theta, \alpha g_\theta) = d$ where $d$ is calculated using the methods in Chapter 6. Since $f_\theta(w)$ and $\alpha\, g_\theta(w)$ are co-prime, and the perturbations are small, the $d$th Sylvester subresultant matrix of $f_\theta(w)$ and $\alpha\, g_\theta(w)$ is nearly singular. It is recalled from Section 6.2.1 that this property of $S_d(f_\theta, \alpha g_\theta)$ leads to the approximation (6.5), and for $k = d$ and $i = q$ this approximation yields,

$$A_{d,q}x \approx c_{d,q}, \qquad x = x_{d,q} \in \mathbb{R}^{m+n-2d+1}, \tag{7.2}$$

where the column $c_{d,q} \in \mathbb{R}^{m+n-d+1}$, the $q$th column of $S_d(f_\theta, \alpha\, g_\theta)$, is the optimal column for the computation of the degree of an AGCD of $f_\theta(w)$ and $\alpha\, g_\theta(w)$, as discussed in Section 6.2.1, and it has been shown in Sections 6.2.2 and 6.2.3 that the value of $q$ follows directly from the computations of $d$. The matrix $A_{d,q} \in \mathbb{R}^{(m+n-d+1)\times(m+n-2d+1)}$ is formed from the remaining columns of $S_d(f_\theta, \alpha g_\theta)$.

Structured perturbations must be applied to the approximation (7.2) to make it an equation that has an exact solution. In particular, let

$$z_{f_\theta} = \begin{bmatrix} z_0\theta^m, & \cdots, & z_{m-1}\theta, & z_m \end{bmatrix}^T \in \mathbb{R}^{m+1}, \tag{7.3}$$

and

$$\alpha z_{g_\theta} = \begin{bmatrix} \alpha z_{m+1}\theta^n, & \cdots, & \alpha z_{m+n}\theta, & \alpha z_{m+n+1} \end{bmatrix}^T \in \mathbb{R}^{n+1}, \tag{7.4}$$

be the vectors of the structured perturbations to be added to the coefficients of $f_\theta(w)$ and $\alpha g_\theta(w)$, respectively. The $d$th Sylvester subresultant structured perturbation

matrix $B_d = B_d(\alpha, \theta, z) \in \mathbb{R}^{(m+n-d+1)\times(m+n-2d+2)}$ of $S_d(f_\theta, \alpha g_\theta)$ is therefore,

$$B_d = \begin{bmatrix} z_0\theta^m & & & \alpha z_{m+1}\theta^n & & \\ z_1\theta^{m-1} & \ddots & & \alpha z_{m+2}\theta^{n-1} & \ddots & \\ \vdots & \ddots & z_0\theta^m & \vdots & \ddots & \alpha z_{m+1}\theta^n \\ z_{m-1}\theta & \ddots & z_1\theta^{m-1} & \alpha z_{m+n}\theta & \ddots & \alpha z_{m+2}\theta^{n-1} \\ z_m & \ddots & \vdots & \alpha z_{m+n+1} & \ddots & \vdots \\ & \ddots & z_{m-1}\theta & & \ddots & \alpha z_{m+n}\theta \\ & & z_m & & & \alpha z_{m+n+1} \end{bmatrix},$$

where

$$z = \begin{bmatrix} z_0, & \cdots, & z_m, & z_{m+1}, & \cdots, & z_{m+n+1} \end{bmatrix}^T \in \mathbb{R}^{m+n+2}. \tag{7.5}$$

Then the application of the method of SNTLN to the computation of an AGCD of $f_\theta(w)$ and $\alpha g_\theta(w)$ requires consideration of the equation

$$\Big(A_{d,q}(\alpha, \theta) + E_{d,q}(\alpha, \theta, z)\Big)x = c_{d,q}(\alpha, \theta) + h_{d,q}(\alpha, \theta, z), \tag{7.6}$$

which is the perturbed form of (7.2), where $h_{d,q}$ is the $q$th column of $B_d(\alpha, \theta, z)$, $E_{d,q}$ is the matrix formed by the remaining columns of $B_d(\alpha, \theta, z)$, and they have same structure as $A_{d,q}$ and $c_{d,q}$, respectively, and as noted before, it is assumed that the values of $d$ and $q$ are known. The quantities $\alpha, \theta$ and $z$ are to be computed using the method of SNTLN. Since the first $n - d + 1$ and the last $m - d + 1$ columns of $S_d(f_\theta, \alpha g_\theta)$ contain the coefficients of $f_\theta(w)$ and $\alpha g_\theta(w)$, respectively, the vectors $c_{d,q}$ and $h_{d,q}$ may or may not be dependent on $\alpha$, depending on the value of $q$. In

particular,

$$\begin{aligned} c_{d,q} &= c_{d,q}(\theta), \quad h_{d,q} = h_{d,q}(\theta, z) \quad \text{if} \quad 1 \leq q \leq n-d+1 \\ c_{d,q} &= c_{d,q}(\alpha, \theta), \quad h_{d,q} = h_{d,q}(\alpha, \theta, z) \quad \text{if} \quad n-d+2 \leq q \leq m+n-2d+2. \end{aligned}$$

The theory for the computation of $\alpha, \theta$ and $z$ is developed, considering the case for which, $n-d+2 \leq q \leq m+n-2d+2$, and the necessary modifications to derive the theory for $1 \leq q \leq n-d+1$ are obtained by setting $\alpha = 1$ and thus the derivative with respect to $\alpha$ is equal to zero.

It is recalled from Section 7.1 that constructing a low rank approximation $S(\tilde{f}_\theta, \alpha\tilde{g}_\theta)$ of $S(f,g)$ for the computation of an AGCD of $f(y)$ and $g(y)$ requires that $\tilde{f}_\theta(w) = f_\theta(w) + \delta z_{f_\theta}(w)$ and $\tilde{g}_\theta(w) = g_\theta(w) + \alpha\delta z_{g_\theta}(w)$ have a non-constant GCD, where the polynomials $\delta z_{f_\theta}(w)$ and $\alpha\delta z_{g_\theta}(w)$ are defined by the vectors of coefficients $z_{f_\theta}$ and $\alpha z_{g_\theta}$ defined in (7.3) and (7.4), respectively. However, it follows from Theorem 3.1 that the polynomials $\tilde{f}_\theta(w)$ and $\alpha\tilde{g}_\theta(w)$ have a non-constant GCD if and only if the non-linear equation (7.6) possesses a solution. This equation can be solved iteratively for the values of $\alpha, \theta, x$ and $z$. The associated residual of computing an approximate solution of (7.6) is,

$$r(\alpha, \theta, x, z) = c_{d,q}(\alpha, \theta) + h_{d,q}(\alpha, \theta, z) - \Big(A_{d,q}(\alpha, \theta) + E_{d,k}(\alpha, \theta, z)\Big)x, \tag{7.7}$$

and thus the residual that is associated with the successive iterations can be defined as $\tilde{r} = r(\alpha + \delta\alpha, \theta + \delta\theta, x + \delta x, z + \delta z)$. Using the Newton-Raphson method [69], the

first order approximation yields,

$$\begin{aligned}
\tilde{r} &= c_{d,q}(\alpha+\delta\alpha,\theta+\delta\theta)+h_{d,q}(\alpha+\delta\alpha,\theta+\delta\theta,z+\delta z)\\
&\quad -\Big(A_{d,q}(\alpha+\delta\alpha,\theta+\delta\theta)+E_{d,q}(\alpha+\delta\alpha,\theta+\delta\theta,z+\delta z)\Big)(x+\delta x)\\
&\approx c_{d,q}+\frac{\partial c_{d,q}}{\partial\alpha}\delta\alpha+\frac{\partial c_{d,q}}{\partial\theta}\delta\theta+h_{d,q}+\frac{\partial h_{d,q}}{\partial\alpha}\delta\alpha+\frac{\partial h_{d,q}}{\partial\theta}\delta\theta+\sum_{i=0}^{m+n+1}\frac{\partial h_{d,q}}{\partial z_i}\delta z_i\\
&\quad -A_{d,q}x-A_{d,q}\delta x-\left(\frac{\partial A_{d,q}}{\partial\alpha}x\right)\delta\alpha-\left(\frac{\partial A_{d,q}}{\partial\theta}x\right)\delta\theta\\
&\quad -E_{d,q}x-E_{d,q}\delta x-\left(\frac{\partial E_{d,q}}{\partial\alpha}x\right)\delta\alpha-\left(\frac{\partial E_{d,q}}{\partial\theta}x\right)\delta\theta\\
&\quad -\left(\sum_{i=0}^{m+n+1}\frac{\partial E_{q,d}}{\partial z_i}\delta z_i\right)x.
\end{aligned}$$

It follows from (7.7) that

$$\begin{aligned}
\tilde{r} &= r(\alpha,\theta,x,z)-\left(\left(\frac{\partial A_{d,q}}{\partial\theta}+\frac{\partial E_{d,q}}{\partial\theta}\right)x-\left(\frac{\partial c_{d,q}}{\partial\theta}+\frac{\partial h_{d,q}}{\partial\theta}\right)\right)\delta\theta\\
&\quad -(A_{d,q}+E_{d,q})\delta x-\left(\left(\frac{\partial A_{d,q}}{\partial\alpha}+\frac{\partial E_{d,q}}{\partial\alpha}\right)x-\left(\frac{\partial c_{d,q}}{\partial\alpha}+\frac{\partial h_{d,q}}{\partial\alpha}\right)\right)\delta\alpha\\
&\quad +\sum_{i=0}^{m+n+1}\frac{\partial h_{d,q}}{\partial z_i}\delta z_i-\sum_{i=0}^{m+n+1}\left(\frac{\partial E_{d,q}}{\partial z_i}\delta z_i\right)x,
\end{aligned}\tag{7.8}$$

where the last two terms can be simplified using the expressions of $h_{d,q}$ and $E_{d,q}x$, in terms of $z$. In particular, the vector $h_{d,q}$ can be written as,

$$h_{d,q}=\alpha P_d z=\alpha\begin{bmatrix}0_{q-n+d-2,m+1} & 0_{q-n+d-2,n+1}\\ 0_{n+1,m+1} & G\\ 0_{m+n-2d-q+2,m+1} & 0_{m+n-2d-q+2,n+1}\end{bmatrix}z,$$

where $P_d = P_d(\theta) \in \mathbb{R}^{(m+n-d+1)\times(m+n+2)}$,

$$G = G(\theta) = \text{diag}\begin{bmatrix} \theta^n & \theta^{n-1} & \cdots & \theta & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)},$$

and $z$ is defined in (7.5). Thus,

$$\delta h_{d,q} = \sum_{i=0}^{m+n+1} \frac{\partial h_{d,q}}{\partial z_i} \delta z_i = \alpha P_d \delta z.$$

The vector $E_{d,q}x$ can be expressed in terms of $z$ as follows,

$$E_{d,q}x = Y_d z,$$

where $Y_d = Y_d(\alpha, \theta, x) \in \mathbb{R}^{(m+n-d+1)\times(m+n+2)}$. Differentiating both sides of this equation with respect to $z$ yields,

$$\sum_{i=0}^{m+n+1} \left( \frac{\partial E_{d,q}}{\partial z_i} \delta z_i \right) x = \left( \delta E_{d,q} \mid_{\alpha,\theta:\text{const.}} \right) x = Y_d \delta z.$$

Replacing the last two terms of (7.8) with $\alpha P_d \delta z$ and $Y_d \delta z$, respectively, simplifies (7.8) to

$$\begin{aligned} \tilde{r} \;=\; & r(\alpha,\theta,x,z) - \left( \left( \frac{\partial A_{d,q}}{\partial \theta} + \frac{\partial E_{d,q}}{\partial \theta} \right) x - \left( \frac{\partial c_{d,q}}{\partial \theta} + \frac{\partial h_{d,q}}{\partial \theta} \right) \right) \delta\theta \\ & -(A_{d,q} + E_{d,q})\delta x - \left( \left( \frac{\partial A_{d,q}}{\partial \alpha} + \frac{\partial E_{d,q}}{\partial \alpha} \right) x - \left( \frac{\partial c_{d,q}}{\partial \alpha} + \frac{\partial h_{d,q}}{\partial \alpha} \right) \right) \delta\alpha \\ & -(Y_d - \alpha P_d)\delta z. \end{aligned} \tag{7.9}$$

For simplicity, let,

$$\begin{aligned} H_z &= Y_d - \alpha P_d \in \mathbb{R}^{(m+n-d+1)\times(m+n+2)}, \\ H_x &= A_{d,q} + E_{d,q} \in \mathbb{R}^{(m+n-d+1)\times(m+n-2d+1)}, \\ H_\alpha &= \left(\frac{\partial A_{d,q}}{\partial \alpha} + \frac{\partial E_{d,q}}{\partial \alpha}\right) x - \left(\frac{\partial c_{d,q}}{\partial \alpha} + \frac{\partial h_{d,q}}{\partial \alpha}\right) \in \mathbb{R}^{m+n-d+1}, \\ H_\theta &= \left(\frac{\partial A_{d,q}}{\partial \theta} + \frac{\partial E_{d,q}}{\partial \theta}\right) x - \left(\frac{\partial c_{d,q}}{\partial \theta} + \frac{\partial h_{d,q}}{\partial \theta}\right) \in \mathbb{R}^{m+n-d+1}. \end{aligned}$$

It follows from Definition 5.2 that the perturbations defined by the elements of the vector $z$ in (7.5), have to be minimised, that is, the given inexact polynomials are moved by the minimum amount such that the refined polynomials have a non-constant GCD. In terms of $H_z, H_x, H_\alpha$ and $H_\theta$, the $j$th iteration in the Newton-Raphson method for calculating $z, x, \alpha$ and $\theta$ is

$$\begin{bmatrix} H_z & H_x & H_\alpha & H_\theta \end{bmatrix}^{(j)} \begin{bmatrix} \delta z \\ \delta x \\ \delta \alpha \\ \delta \theta \end{bmatrix}^{(j)} = r^{(j)}, \tag{7.10}$$

where $r^{(j)} = r^{(j)}(\alpha, \theta, x, z)$, and the values of $z, x, \alpha$ and $\theta$ at the $(j+1)$th iteration are,

$$\begin{bmatrix} z \\ x \\ \alpha \\ \theta \end{bmatrix}^{(j+1)} = \begin{bmatrix} z \\ x \\ \alpha \\ \theta \end{bmatrix}^{(j)} + \begin{bmatrix} \delta z \\ \delta x \\ \delta \alpha \\ \delta \theta \end{bmatrix}^{(j)},$$

such that at $j = 0$, $z^{(0)} = 0$ because the given data is inexact, and $\alpha^{(0)}$ and $\theta^{(0)}$ are the solutions $\alpha_0$ and $\theta_0$ of the LP problem (4.7).

Clearly, it can be seen that (7.10) is of the form

$$Cy = b,$$

where $C \in \mathbb{R}^{(m+n-d+1)\times(2m+2n-2d+5)}$, $y \in \mathbb{R}^{2m+2n-2d+5}$, $b \in \mathbb{R}^{m+n-d+1}$,

$$C = \begin{bmatrix} H_z & H_x & H_\alpha & H_\theta \end{bmatrix}^{(j)}, \qquad y = \begin{bmatrix} \delta z \\ \delta x \\ \delta\alpha \\ \delta\theta \end{bmatrix}^{(j)}, \qquad b = r^{(j)}. \tag{7.11}$$

Since the nearest polynomial that has a multiple root is sought, it is required to minimise,

$$\left\| \begin{bmatrix} z^{(j+1)} - z^{(0)} \\ x^{(j+1)} - x^{(0)} \\ \alpha^{(j+1)} - \alpha_0 \\ \theta^{(j+1)} - \theta_0 \end{bmatrix} \right\| = \left\| \begin{bmatrix} z^{(j)} + \delta z^{(j)} \\ x^{(j)} + \delta x^{(j)} - x_0 \\ \alpha^{(j)} + \delta\alpha^{(j)} - \alpha_0 \\ \theta^{(j)} + \delta\theta^{(j)} - \theta_0 \end{bmatrix} \right\| := \|Ey - p\|,$$

where

$$E = I_{2m+2n-2d+5}, \quad p = -\begin{bmatrix} z^{(j)} & x^{(j)} - x_0 & \alpha^{(j)} - \alpha_0 & \theta^{j} - \theta_0 \end{bmatrix}^T, \tag{7.12}$$

and $y$ is defined in (7.11). The initial value $x_0$ of $x$ is obtained by setting $\alpha = \alpha_0$, $\theta = \theta_0$ and $z = 0$ in (7.7),

$$x_0 = \arg\min_{w} \|A_{d,q}(\alpha_0, \theta_0)w - c_{d,q}(\alpha_0, \theta_0)\| . \tag{7.13}$$

It follows from the discussion above that the method of SNTLN yields the following least squares equality (LSE) problem,

$$\min_{y} \|Ey - p\| \qquad \text{subject to} \qquad Cy = b. \tag{7.14}$$

The QR decomposition [28] can be used to solve this problem.

The following algorithm shows the application of the method of SNTLN to the Sylvester matrix of two inexact polynomials $f(y)$ and $g(y)$, for computing a structured low rank approximation of $S(f, g)$, where the QR decomposition is used to solve the LSE problem (7.14).

---

**Algorithm 7.2: A structured low rank approximation of the Sylvester matrix using the method of SNTLN**

**Input**

**(1)** $f = f(y)$ and $g = g(y)$, the inexact polynomials whose degrees are $m$ and $n$, respectively.

**(2)** $\alpha_0$ and $\theta_0$, the initial values of $\alpha$ and $\theta$, respectively.

**(3)** $d \leq \min(m, n)$, the degree of the AGCD of $f(y)$ and $g(y)$.

**(4)** $q$, the index of the optimal column for the computation of the degree of an AGCD.

**Output** A structured low rank approximation of $S(f, g)$ with rank $m+n-d$. **Begin**

% Initialisation

1. Set $z = 0$, and thus $E_{d,q} = \frac{\partial E_{d,q}}{\partial \alpha} = \frac{\partial E_{d,q}}{\partial \theta} = 0$ and $h_{d,q} = \frac{\partial h_{d,q}}{\partial \alpha} = \frac{\partial h_{d,q}}{\partial \theta} = 0$.

2. Calculate $A_{d,q}, Y_{d,q}, P_d, c_{d,q}, \frac{\partial A_{d,q}}{\partial \alpha}, \frac{\partial A_{d,q}}{\partial \theta}, \frac{\partial c_{d,q}}{\partial \alpha}$ and $\frac{\partial c_{d,q}}{\partial \theta}$ for $\alpha = \alpha_0, \theta = \theta_0$ and $x = x_0$, which is defined in (7.13). These initial values will also set $p$ in (7.12) to be equal to 0. Calculate the initial value of $b$, that is, the residual $r(\alpha_0, \theta_0, x_0, z = 0) = c_{d,q} - A_{d,q}x_0$.

3. Calculate the matrices $C$ and $E$ defined in (7.11) and (7.12), respectively.

4. % Solve the LSE problem (7.14), using the QR decomposition

   % Set iterations $= 0$.

   **Repeat**

   **(a)** Compute the $QR$ decomposition of $C^T$,

$$C^T = QR = Q \begin{bmatrix} R_1 \\ 0 \end{bmatrix}.$$

   **(b)** Set $w_1 = R_1^{-T} b$.

   **(c)** Partition $EQ$,

$$EQ = \begin{bmatrix} E_1 & E_2 \end{bmatrix},$$

such that $E_1 \in \mathbb{R}^{(2m+2n-2d+5)\times(m+n-d+1)}$ and $E_2 \in \mathbb{R}^{(2m+2n-2d+5)\times(m+n-d+4)}$.

**(d)** Compute

$$w_2 = E_2^{\dagger}\left(p - E_1 w_1\right).$$

**(e)** Compute the solution

$$y = Q \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}.$$

**(f)** Set $z := z + \delta z$, $x := x + \delta x, \alpha := \alpha + \delta\alpha$ and $\theta := \theta + \delta\theta$.

**(g)** Update $A_{d,q}$, $\frac{\partial A_{d,q}}{\partial \alpha}$, $\frac{\partial A_{d,q}}{\partial \theta}$, $E_{d,q}$, $\frac{\partial E_{d,q}}{\partial \alpha}$, $\frac{\partial E_{d,q}}{\partial \theta}$, $Y_d$, $P_d$, $c_{d,q}$, $\frac{\partial c_{d,q}}{\partial \alpha}$, $\frac{\partial c_{d,q}}{\partial \theta}$, $h_{d,q}$, $\frac{\partial h_{d,q}}{\partial \alpha}$, $\frac{\partial A_{d,q}}{\partial \theta}$ from $\alpha, \theta, x$ and $z$. Using the updated values, update $C$ and $p$.

**(h)** Calculate the updated value of $b$, that is the residual

$$r(\alpha, \theta, x, z) = (c_{d,q} + h_{d,q}) - (A_{d,q} + E_{d,q})x.$$

**(i)** Increment the iterations.

**Until** $\frac{\|r(\alpha,\theta,x,z)\|}{\|c_{d,q}+h_{d,q}\|} \leq 10^{-12}$ **or** iterations $< 50$

**End**

---

## 7.3 Examples

This section contains two examples that show the use of the method of SNTLN for the construction of a structured low rank approximation $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$, of $S(f, g)$ of two

inexact polynomials $f = f(y)$ and $g = g(y)$, by applying the method of SNTLN to the Sylvester matrix $S(f, g)$, where $\alpha^*$ and $\theta^*$ are the values of $\alpha$ and $\theta$, respectively, at the termination of the method of SNTLN, and $\tilde{f}_{\theta^*} = \tilde{f}_{\theta^*}(w)$ and $\tilde{g}_{\theta^*} = \tilde{g}_{\theta^*}(w)$. The following notation is used in these examples:

1. $\hat{f}(y)$ and $\hat{g}(y)$ are the theoretically exact polynomials.

2. $f(y)$ and $g(y)$ are the inexact polynomials whose coefficients are calculated from $\hat{f}(y)$ and $\hat{g}(y)$, respectively, after adding componentwise noise to their coefficients and normalising them by their geometric means.

3. $f_{\theta_0}(w)$ and $\alpha_0 g_{\theta_0}(w)$ are the preprocessed inexact polynomials that are used in the computation of the structured low rank approximation of the inexact polynomials $f(y)$ and $g(y)$.

4. $\tilde{f}_{\theta^*}(w)$ and $\tilde{g}_{\theta^*}(w)$ are the corrected polynomials whose coefficients are computed by the method of SNTLN.

5. The entries of the Sylvester matrices $S(\hat{f}, \hat{g})$, $S(f, g)$ and $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$, are calculated from the theoretically exact, inexact and corrected pairs of polynomials, respectively, after normalising them by the geometric means of their coefficients.

**Example 7.1.** Consider the polynomials

$$\begin{aligned}
\hat{f}(y) &= (y - 0.3396)^3(y + 0.5790)^3(y - 10.7712)^5 \times \\
&\quad (y - 5.8708)^4(y - 20.7633)^5, \\
\hat{g}(y) &= (y - 0.3396)^3(y + 0.5790)^5(y + 5.2495)^3 \times \\
&\quad (y - 5.8708)^2(y - 1.0777)^3,
\end{aligned}$$

whose Sylvester matrix is of order 36, and since the degree of their GCD is equal to 8,

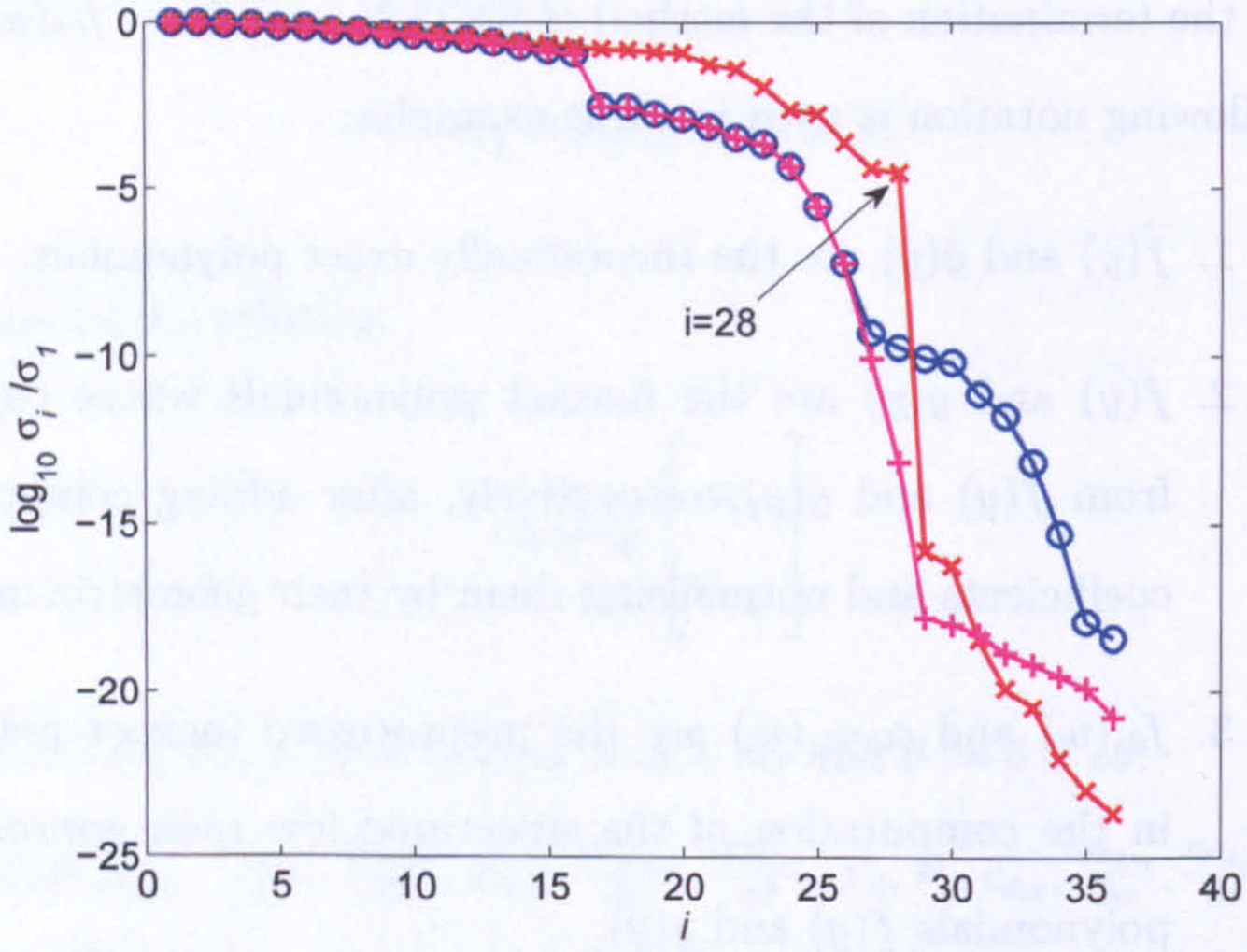



Figure 7.1: The normalised singular values of the Sylvvester matrices $S(f, g)$ o, $S(\hat{f}, \hat{g})+$ and $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})\times$, for Example 7.1.

the theoretically exact rank of $S(\hat{f}, \hat{g})$ is equal to $36 - 8 = 28$. Noise with componentwise signal-to-noise ratio of $10^6$ was added to the coefficients of these polynomials, which were then normalised, thereby yielding the polynomials $f(y)$ and $g(y)$.

The method of the first principal angle, which is discussed in Section 6.2.2, was used to calculate the degree $d = 8$ of the AGCD of $f(y)$ and $g(y)$, which is correct, and the index of the best column, $q = 3$, to form the approximation in (7.2). These results were then used in the implementation of Algorithm 7.2. It was found that two iterations were required for the solution of the LSE problem (7.14), and the following

values were obtained at the termination of Algorithm 7.2:

$$\|Ey - p\| = 10^{-7.7} \quad \text{and} \quad \|Cy - b\| = 10^{-15.4}.$$

Very similar results were obtained when the method that uses the residual, which is discussed in Section 6.2.3, was used to find the values of $d$ and $q$.

Figure 7.1 shows the normalised singular values of the Sylvester matrices $S(\hat{f}, \hat{g})$, $S(f, g)$ and $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$. It can be seen that the rank of $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$ is equal to 28, which is the correct value, whereas $S(\hat{f}, \hat{g})$ and $S(f, g)$ have full rank which suggests that $\hat{f}(y)$ and $\hat{g}(y)$ are co-prime, which is incorrect. □

**Example 7.2.** Consider the polynomials

$$\begin{aligned}
\hat{f}(y) &= (y - 3.671684 \times 10^{-5})^5 (y - 3.062163 \times 10^{-5})^2 (y - 5.724097 \times 10^{-5})^4 \times \\
&\quad (y - 3.981184 \times 10^{-5})^2 (y - 6.896876 \times 10^{-5})^3 (y - 1.151630 \times 10^{-4})^5, \\
\hat{g}(y) &= (y - 3.671684 \times 10^{-5})^5 (y - 3.062163 \times 10^{-5})^2 (y - 3.330558)^4 \times \\
&\quad (y + 6.437351)^4 (y + 7.439712)^3 (y - 9.981608)^4,
\end{aligned}$$

whose Sylvester matrix is of order 43, and since the degree of their GCD is equal to 7, the theoretically exact rank of $S(\hat{f}, \hat{g})$ is equal to $43 - 7 = 36$. Noise with componentwise signal-to-noise ratio of $10^8$ was added to the coefficients of these polynomials, which were then normalised, thereby yielding the polynomials $f(y)$ and $g(y)$.

The method of the first principal angle was used to calculate the degree $d = 7$ of the AGCD of $f(y)$ and $g(y)$, which is correct, and the index of the best column, $q$, to form (7.2). These results were then used in the implementation of Algorithm 7.2. It was found that one iteration was required for the solution of the LSE problem (7.14),

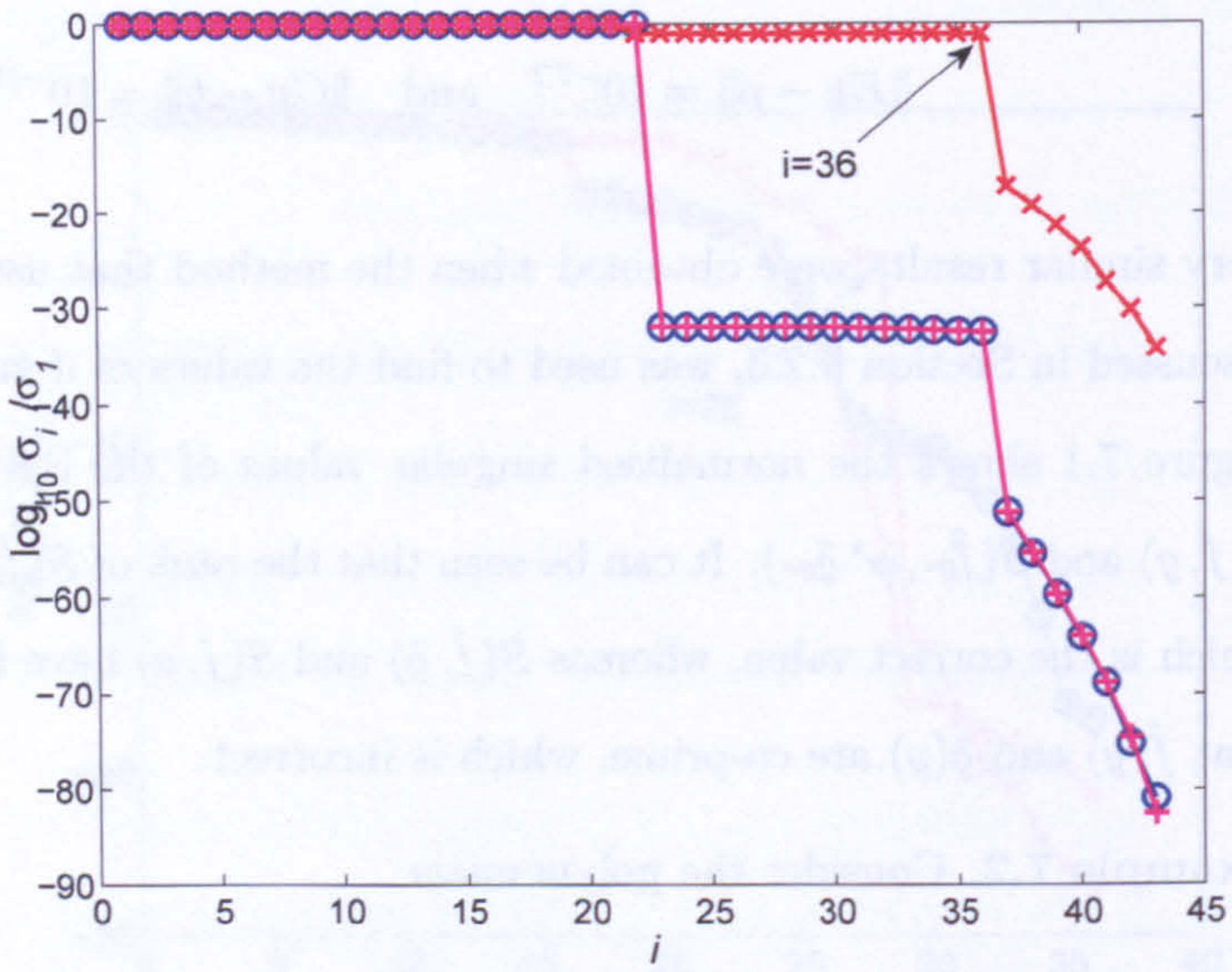



Figure 7.2: The normalised singular values of the Sylvvester matrices $S(f,g)$ o, $S(\hat{f},\hat{g})+$ and $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})\times$, for Example 7.2.

and the following values were obtained at the termination of Algorithm 7.2:

$$\|Ey - p\| = 10^{-11.4} \quad \text{and} \quad \|Cy - b\| = 10^{-15.1}.$$

Similar results were obtained when the method that uses the residual was used to find the values of $d$ and $q$.

Figure 7.2 shows the normalised singular values of the Sylvester matrices $S(\hat{f},\hat{g})$, $S(f,g)$ and $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$. It can be seen that the rank of $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$ is equal to 36, which is the correct value, whereas $\operatorname{rank} S(\hat{f},\hat{g}) = \operatorname{rank} S(f,g) = 22$, which is incorrect. □

## 7.4 Calculating an AGCD using APF

The method considered in this section computes a low rank approximation $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$, see Section 7.1 for more details, of $S(f, g)$ of the inexact polynomials $f = f(y)$ and $g = g(y)$, by applying the method of SNTLN to the approximate polynomial factorisation of $f(y)$ and $g(y)$, where $\alpha^*$ and $\theta^*$ are the values of $\alpha$ and $\theta$ respectively, at the termination of the method of SNTLN. This method differs from the method discussed in Section 7.2 because it explicitly computes the GCD of $\tilde{f}_{\theta^*}(w)$ and $\tilde{g}_{\theta^*}(w)$. On the other hand, the Sylvester matrix requires that $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$ be reduced to a triangular form, which can be unstable [2].

The application of the method of SNTLN to APF of two inexact polynomials $f(y)$ and $g(y)$ for computing $\mathrm{AGCD}(f, g)$ requires that their scaling forms $f_\theta(w)$ and $\alpha g_\theta(w)$, which are defined in (4.5) and (4.6), respectively, be considered. Specifically,

$$f_\theta(w) = \sum_{i=0}^{m} \left(\frac{a_i^*}{\theta_0^{m-i}}\right) \theta^{m-i} w^{m-i} \qquad \text{and} \qquad g_\theta(w) = \sum_{i=0}^{n} \left(\frac{b_i^*}{\theta_0^{n-i}}\right) \theta^{m-i} w^{n-i},$$

where

$$a_i^* = \bar{a}_i \theta_0^{m-i} \qquad \text{and} \qquad b_i^* = \bar{b}_i \theta_0^{n-i},$$

where $\bar{a}_i$ and $\bar{b}_i$ are defined in (4.2) and (4.3), respectively. It is assumed that the degree $d$ of the AGCD of $f(y)$ and $g(y)$, is known using the methods described in Chapter 6. The coefficients $a_i^*$ and $b_i^*$ form the entries of $S_d(f_\theta, \alpha g_\theta)$, and the parameters $\alpha$ and $\theta$ need to be iteratively refined starting from $\alpha_0$ and $\theta_0$, which are obtained from solving the LP problem (4.7).

The approximate factorisation of the inexact polynomials $f_\theta(w)$ and $g_\theta(w)$ can be

written as

$$f_\theta(w) \approx u_\theta(w)d_\theta(w) \quad \text{and} \quad \alpha g_\theta(w) \approx v_\theta(w)d_\theta(w), \tag{7.15}$$

where

$$d_\theta = d_\theta(w) = \sum_{i=0}^{d} \left(\frac{d_{\theta,i}}{\theta_0^{d-i}}\right) \theta^{d-i} w^{d-i} = \sum_{i=0}^{d} \left(r_i \theta^{d-i}\right) w^{d-i},$$

is an AGCD of degree $d$ of the inexact polynomials $f_\theta(w)$ and $g_\theta(w)$, and the quotient polynomials

$$u_\theta = u_\theta(w) \;=\; \sum_{i=0}^{m-d} \left(\frac{u_i}{\theta_0^{m-d-i}}\right) \theta^{m-d-i} w^{m-d-i} = \sum_{i=0}^{m-d} \left(c_i \theta^{m-d-i}\right) w^{m-d-i}$$

$$v_\theta = v_\theta(w) \;=\; \sum_{i=0}^{n-d} \left(\frac{v_i}{\theta_0^{n-d-i}}\right) \theta^{n-d-i} w^{n-d-i} = \sum_{i=0}^{n-d} \left(e_i \theta^{n-d-i}\right) w^{n-d-i},$$

are co-prime, where

$$r_i = \frac{d_{\theta,i}}{\theta_0^{d-i}}, \quad c_i = \frac{u_{\theta,i}}{\theta_0^{m-d-i}}, \quad \text{and} \quad e_i = \frac{v_{\theta,i}}{\theta_0^{n-d-i}}. \tag{7.16}$$

The value $\theta_0$ of $\theta$ is retained in the denominators of the coefficients of $d_\theta(w)$, $u_\theta(w)$ and $v_\theta(w)$ to simplify the update process between the successive iterations of the method of SNTLN. It follows that the full form of (7.15) is

$$\sum_{i=0}^{m} \left(\bar{a}_i \theta^{m-i}\right) w^{m-i} \approx \left(\sum_{i=0}^{m-d} \left(c_i \theta^{m-d-i}\right) w^{m-d-i}\right) \left(\sum_{i=0}^{d} \left(r_i \theta^{d-i}\right) w^{d-i}\right),$$

and

$$\alpha_0 \sum_{i=0}^{n} \left(\bar{b}_i \theta^{n-i}\right) w^{n-i} \approx \left(\sum_{i=0}^{n-d} \left(e_i \theta^{n-d-i}\right) w^{n-d-i}\right) \left(\sum_{i=0}^{d} \left(r_{d,i} \theta^{d-i}\right) w^{d-i}\right).$$

These approximations can be combined in matrix form as

$$\begin{bmatrix} C_1(c,\theta) \\ C_2(e,\theta) \end{bmatrix} \mathbf{r}(\theta) \approx \begin{bmatrix} \mathbf{f}(\theta) \\ \alpha_0 \mathbf{g}(\theta) \end{bmatrix}, \tag{7.17}$$

where

$$C_1(c,\theta) = \begin{bmatrix} c_0\theta^{m-d} & & & \\ c_1\theta^{m-d-1} & c_0\theta^{m-d} & & \\ c_2\theta^{m-d-2} & c_1\theta^{m-d-1} & \ddots & \\ \vdots & c_2\theta^{m-d-2} & \ddots & c_0\theta^{m-d} \\ \vdots & \vdots & \ddots & c_1\theta^{m-d-1} \\ c_{m-d-1}\theta & \vdots & \ddots & c_2\theta^{m-d-2} \\ c_{m-d} & c_{m-d-1}\theta & \ddots & \vdots \\ & c_{m-d} & \ddots & \vdots \\ & & \ddots & c_{m-d-1}\theta \\ & & & c_{m-d} \end{bmatrix} \in \mathbb{R}^{(m+1)\times(d+1)},$$

and

$$C_2(e,\theta) = \begin{bmatrix} e_0\theta^{n-d} & & & \\ e_1\theta^{n-d-1} & e_0\theta^{n-d} & & \\ e_2\theta^{n-d-2} & e_1\theta^{n-d-1} & \ddots & \\ \vdots & e_2\theta^{n-d-2} & \ddots & e_0\theta^{n-d} \\ \vdots & \vdots & \ddots & e_1\theta^{n-d-1} \\ e_{n-d-1}\theta & \vdots & \ddots & e_2\theta^{n-d-2} \\ e_{n-d} & e_{n-d-1}\theta & \ddots & \vdots \\ & e_{n-d} & \ddots & \vdots \\ & & \ddots & e_{n-d-1}\theta \\ & & & e_{n-d} \end{bmatrix} \in \mathbb{R}^{(n+1)\times(d+1)},$$

and

$$\begin{aligned} \mathbf{f}(\theta) &= \begin{bmatrix} \bar{a}_0\theta^m & \bar{a}_1\theta^{m-1} & \cdots & \bar{a}_{m-1}\theta & \bar{a}_m \end{bmatrix}^T \in \mathbb{R}^{m+1}, \\ \mathbf{g}(\theta) &= \begin{bmatrix} \bar{b}_0\theta^n & \bar{b}_1\theta^{n-1} & \cdots & \bar{b}_{n-1}\theta & \bar{b}_n \end{bmatrix}^T \in \mathbb{R}^{n+1}, \\ \mathbf{r}(\theta) &= \begin{bmatrix} r_0\theta^d & r_1\theta^{d-1} & \cdots & r_{d-1}\theta & r_d \end{bmatrix}^T \in \mathbb{R}^{d+1}. \end{aligned}$$

The application of the method of SNTLN requires that the coefficient matrix and the right hand side vector of (7.17) be perturbed such that the approximation is replaced by

$$\begin{bmatrix} C_1(c,\theta) + E_1(z,\theta) \\ C_2(e,\theta) + E_2(z,\theta) \end{bmatrix} \mathbf{r}(\theta) = \begin{bmatrix} \mathbf{f}(\theta) + \mathbf{s}(p,\theta) \\ (\alpha_0 + \beta_0)\,(\mathbf{g}(\theta) + \mathbf{t}(q,\theta)) \end{bmatrix}, \tag{7.18}$$

which has an exact solution, where

$$E_1(z_d,\theta) = \begin{bmatrix} z_0\theta^{m-d} & & \\ z_1\theta^{m-d-1} & \ddots & \\ z_2\theta^{m-d-2} & \ddots & z_0\theta^{m-d} \\ \vdots & \ddots & z_1\theta^{m-d-1} \\ \vdots & \ddots & z_2\theta^{m-d-2} \\ z_{m-d-1}\theta & \ddots & \vdots \\ z_{m-d} & \ddots & \vdots \\ & \ddots & z_{m-d-1}\theta \\ & & z_{m-d} \end{bmatrix} \in \mathbb{R}^{(m+1)\times(d+1)},$$

$$E_2(z_d,\theta) = \begin{bmatrix} z_{m-d+1}\theta^{n-d} & & \\ z_{m-d+2}\theta^{n-d-1} & \ddots & \\ z_{m-d+3}\theta^{n-d-2} & \ddots & z_{m-d+1}\theta^{n-d} \\ \vdots & \ddots & z_{m-d+2}\theta^{n-d-1} \\ \vdots & \ddots & z_{m-d+3}\theta^{n-d-2} \\ z_{m+n-2d}\theta & \ddots & \vdots \\ z_{m+n-2d+1} & \ddots & \vdots \\ & \ddots & z_{m+n-2d}\theta \\ & & z_{m+n-2d+1} \end{bmatrix} \in \mathbb{R}^{(n+1)\times(d+1)},$$

are the Cauchy matrices of the perturbations

$$z = \begin{bmatrix} z_0 & \cdots & z_{m-d} & z_{m-d+1} & \cdots & z_{m+n-2d+1} \end{bmatrix}^T \in \mathbb{R}^{m+n-2d+2},$$

that are added to the coefficients $c_i$ and $e_i$. The vectors $\mathbf{s} = \mathbf{s}(p, \theta) \in \mathbb{R}^{m+1}$ and $\mathbf{t} = \mathbf{t}(q, \theta) \in \mathbb{R}^{n+1}$,

$$\mathbf{s} = [\, p_0\theta^m \quad p_1\theta^{m-1} \quad \cdots \quad p_{m-1}\theta \quad p_m \,]^T \in \mathbb{R}^{m+1},$$
$$\mathbf{t} = [\, q_0\theta^n \quad q_1\theta^{n-1} \quad \cdots \quad q_{n-1}\theta \quad q_n \,]^T \in \mathbb{R}^{n+1},$$

where

$$p = [\, p_0 \quad p_1 \quad \cdots \quad p_{m-1} \quad p_m \,]^T \in \mathbb{R}^{m+1},$$
$$q = [\, q_0 \quad q_1 \quad \cdots \quad q_{n-1} \quad q_n \,]^T \in \mathbb{R}^{n+1},$$

are the vectors of the perturbations that are added to the coefficients of $f_\theta(w)$ and $g_\theta(w)$, respectively, and $\beta_0$ is the perturbation that is added to $\alpha_0$. The computations of the perturbation vectors $z, p$, and $q$, an estimate for $\mathbf{r}(\theta)$, and the scalars $\beta_0$ and $\theta$, require that (7.18) be solved. This equation is non-linear and it is solved iteratively using the method of Newton-Raphson.

An approximate solution for (7.18) yields the following residual

$$r(\beta_0, \theta, z, \mathbf{r}, p, q) = \begin{bmatrix} \mathbf{f}(\theta) + \mathbf{s}(p, \theta) \\ (\alpha_0 + \beta_0)\,(\mathbf{g}(\theta) + \mathbf{t}(q, \theta)) \end{bmatrix} - \begin{bmatrix} C_1(c, \theta) + E_1(z, \theta) \\ C_2(e, \theta) + E_2(z, \theta) \end{bmatrix} \mathbf{r}(\theta). \tag{7.19}$$

Thus, a first order Taylor expansion yields

$$\begin{aligned} & r(\beta_0+\delta\beta_0, \theta+\delta\theta, z+\delta z, \mathbf{r}+\delta\mathbf{r}, p+\delta p, q+\delta q) \\ &= \begin{bmatrix} \mathbf{f}(\theta+\delta\theta)+\mathbf{s}(p+\delta p, \theta+\delta\theta) \\ (\alpha_0+\beta_0+\delta\beta_0)\,(\mathbf{g}(\theta+\delta\theta)+\mathbf{t}(q+\delta q, \theta+\delta\theta)) \end{bmatrix} \qquad (7.20) \\ &- \begin{bmatrix} C_1(c, \theta+\delta\theta)+E_1(z+\delta z, \theta+\delta\theta) \\ C_2(e, \theta+\delta\theta)+E_2(z+\delta z, \theta+\delta\theta) \end{bmatrix} \mathbf{r}(\theta+\delta\theta). \qquad (7.21) \end{aligned}$$

To simplify the analysis of this expression, let us first consider (7.20):

To first order, the approximation of the first expression in (7.20) is

$$\mathbf{f}(\theta+\delta\theta)+\mathbf{s}(p_+\delta p, \theta+\delta\theta) \;\approx\; \mathbf{f}+\mathbf{s}+\frac{\partial \mathbf{f}}{\partial\theta}\delta\theta+\frac{\partial \mathbf{s}}{\partial\theta}\delta\theta+\sum_{i=0}^{m}\frac{\partial \mathbf{s}}{\partial p_i}\delta p_i,$$

and the approximation of the second expression in (7.20) is

$$\begin{aligned} & (\alpha_0+\beta_0+\delta\beta_0)\left(\mathbf{g}(\theta+\delta\theta)+\mathbf{t}(q+\delta q, \theta+\delta\theta)\right) \\ & \approx (\alpha_0+\beta_0)\,(\mathbf{g}+\mathbf{t})+(\alpha_0+\beta_0)\left(\tfrac{\partial \mathbf{g}}{\partial\theta}\delta\theta+\tfrac{\partial \mathbf{t}}{\partial\theta}\delta\theta+\textstyle\sum_{i=0}^{n}\tfrac{\partial \mathbf{t}}{\partial q_i}\delta q_i\right) \\ & +(\mathbf{g}+\mathbf{t})\,\delta\beta_0. \end{aligned}$$

The vectors $\mathbf{s}$ and $\mathbf{t}$ can be written as $\mathbf{s}=Sp$ and $\mathbf{t}=Tq$, respectively, where

$$\begin{aligned} S=S(\theta) &= \operatorname{diag}\begin{bmatrix} \theta^m & \theta^{m-1} & \dots & \theta & 1 \end{bmatrix} \in \mathbb{R}^{(m+1)\times(m+1)}, \\ T=T(\theta) &= \operatorname{diag}\begin{bmatrix} \theta^n & \theta^{n-1} & \dots & \theta & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}. \qquad (7.22) \end{aligned}$$

It follows that

$$\sum_{i=0}^{m} \frac{\partial \mathbf{s}}{\partial p_i} \delta p_i = S\delta p \qquad \text{and} \qquad \sum_{i=0}^{n} \frac{\partial \mathbf{t}}{\partial q_i} \delta q_i = T\delta q.$$

Thus

$$\mathbf{f}(\theta+\delta\theta) + \mathbf{s}(p+\delta p, \theta+\delta\theta) \;\approx\; \mathbf{f} + \mathbf{s} + \frac{\partial \mathbf{f}}{\partial \theta}\delta\theta + \frac{\partial \mathbf{s}}{\partial \theta}\delta\theta + S\delta p, \tag{7.23}$$

and

$$\begin{aligned} &(\alpha_0+\beta_0+\delta\beta_0)\left(\mathbf{g}(\theta+\delta\theta) + \mathbf{t}(q+\delta q, \theta+\delta\theta)\right) \\ &\approx (\alpha_0+\beta_0)(\mathbf{g}+\mathbf{t}) + (\alpha_0+\beta_0)\left(\frac{\partial \mathbf{g}}{\partial \theta}\delta\theta + \frac{\partial \mathbf{t}}{\partial \theta}\delta\theta + T\delta q\right) \\ &+ (\mathbf{g}+\mathbf{t})\,\delta\beta_0. \end{aligned} \tag{7.24}$$

Using (7.23) and (7.24), the expression in (7.20) can be written as

$$\begin{bmatrix} \mathbf{f}+\mathbf{s} \\ (\alpha_0+\beta_0)(\mathbf{g}+\mathbf{t}) \end{bmatrix} + \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial \theta}\delta\theta + \frac{\partial \mathbf{s}}{\partial \theta}\delta\theta + S\delta p \\ (\alpha_0+\beta_0)\left(\frac{\partial \mathbf{g}}{\partial \theta}\delta\theta + \frac{\partial \mathbf{t}}{\partial \theta}\delta\theta + T\delta q\right) + (\mathbf{g}+\mathbf{t})\,\delta\beta_0 \end{bmatrix}. \tag{7.25}$$

Consider now the expression in (7.21). The following substitutions

$$B = B(c, e, \theta) = \begin{bmatrix} C_1(c, \theta) \\ C_2(e, \theta) \end{bmatrix} \quad \text{and} \quad E = E(z, \theta) = \begin{bmatrix} E_1(z, \theta) \\ E_2(z, \theta) \end{bmatrix},$$

allow (7.21) to be rewritten as

$$-\Big(B(c,e,\theta+\delta\theta)+E(z+\delta z,\theta+\delta\theta)\Big)\mathbf{r}(\theta+\delta\theta),$$

whose first order approximation is

$$\begin{aligned} &= -\left(B+\frac{\partial B}{\partial\theta}\delta\theta+E+\frac{\partial E}{\partial\theta}\delta\theta+\sum_{i=0}^{m+n-2d+1}\frac{\partial E}{\partial z_i}\delta z_i\right)\left(\mathbf{r}(\theta)+\frac{d\left(\mathbf{r}(\theta)\right)}{d\theta}\delta\theta\right)\\ &= -(B+E)\,\mathbf{r}-\left(\frac{\partial B}{\partial\theta}\delta\theta+\frac{\partial E}{\partial\theta}\delta\theta+\delta E\right)\mathbf{r}-(B+E)\,\frac{d\mathbf{r}}{d\theta}\delta\theta, \end{aligned} \tag{7.26}$$

where

$$\delta E=\sum_{i=0}^{m+n-2d+1}\frac{\partial E}{\partial z_i}\delta z_i.$$

Also, let $Y_1\in\mathbb{R}^{(m+1)\times(m+n-2d+2)}$, $Y_2\in\mathbb{R}^{(n+1)\times(m+n-2d+2)}$, and

$$Y=Y(\mathbf{r},\theta)=\begin{bmatrix} Y_1(\mathbf{r},\theta)\\ Y_2(\mathbf{r},\theta)\end{bmatrix}, \tag{7.27}$$

where

$$Y_1(\mathbf{r},\theta) = \begin{bmatrix} C_3(\mathbf{r})\Theta_1 & 0_{m+1,n-d+1}\end{bmatrix},\quad C_3(\mathbf{r})\in\mathbb{R}^{(m+1)\times(m-d+1)},$$

$$Y_2(\mathbf{r},\theta) = \begin{bmatrix} 0_{n+1,m-d+1} & C_4(\mathbf{r})\Theta_2\end{bmatrix},\quad C_4(\mathbf{r})\in\mathbb{R}^{(n+1)\times(n-d+1)},$$

where the matrices $C_3(\mathbf{r})$ and $C_4(\mathbf{r})$ are the Cauchy matrices of $\mathbf{r}$, with different dimensions, and

$$\Theta_1 = \operatorname{diag}\begin{bmatrix} \theta^{m-d} & \theta^{m-d-1} & \dots & \theta & 1 \end{bmatrix} \in \mathbb{R}^{(m-d+1)\times(m-d+1)},$$

$$\Theta_2 = \operatorname{diag}\begin{bmatrix} \theta^{n-d} & \theta^{n-d-1} & \dots & \theta & 1 \end{bmatrix} \in \mathbb{R}^{(n-d+1)\times(n-d+1)}.$$

The expression in (7.26) can be simplified by differentiating, with respect to $z$, both sides of the equation

$$Y(\mathbf{r},\theta)z = E(z,\theta)\mathbf{r},$$

to give,

$$\delta E(z,\theta)\mathbf{r} = Y(\mathbf{r},\theta)\delta z.$$

Therefore, (7.21) can be written as

$$-(B+E)\,\mathbf{r} - (B+E)\,\frac{d\mathbf{r}}{d\theta}\delta\theta - Y\delta z - \left(\frac{\partial B}{\partial\theta}\mathbf{r} + \frac{\partial E}{\partial\theta}\mathbf{r}\right)\delta\theta. \tag{7.28}$$

Substituting for (7.25) and (7.28) into (7.20) and (7.21), respectively, yields

$$
\begin{aligned}
& r(\beta_0+\delta\beta_0, \theta+\delta\theta, z+\delta z, \mathbf{r}+\delta\mathbf{r}, p+\delta p, q+\delta q) \\
& \approx r(\beta_0, \theta, z, \mathbf{r}, p, q) \\
& + \begin{bmatrix} S & 0_{m+1,n+1} & 0_{m+1,1} & \frac{\partial \mathbf{f}}{\partial\theta}+\frac{\partial \mathbf{s}}{\partial\theta} \\ 0_{n+1,m+1} & (\alpha_0+\beta_0)T & \mathbf{g}+\mathbf{t} & (\alpha_0+\beta_0)\left(\frac{\partial \mathbf{g}}{\partial\theta}+\frac{\partial \mathbf{t}}{\partial\theta}\right) \end{bmatrix} \begin{bmatrix} \delta p \\ \delta q \\ \delta\beta_0 \\ \delta\theta \end{bmatrix} \\
& -(B+E)\frac{d\mathbf{r}}{d\theta}\delta\theta - Y\delta z - \left(\frac{\partial B}{\partial\theta}\mathbf{r}+\frac{\partial E}{\partial\theta}\mathbf{r}\right)\delta\theta.
\end{aligned}
$$

The $j$th iteration in the Newton-Raphson method for calculating $z, p, q, \beta_0$ and $\theta$ is

$$
\begin{aligned}
& \left[\begin{array}{c|ccc} Y & -S & 0_{m+1,n+1} & 0_{m+1,1} \\ & 0_{n+1,m+1} & -(\alpha_0+\beta_0)T & -(\mathbf{g}+\mathbf{t}) \end{array}\right. \\
& \qquad \left.\begin{array}{c} -\left(\frac{\partial \mathbf{f}}{\partial\theta}+\frac{\partial \mathbf{s}}{\partial\theta}\right)+\left(\frac{\partial C_1}{\partial\theta}+\frac{\partial E_1}{\partial\theta}\right)\mathbf{r}+(C_1+E_1)\frac{d\mathbf{r}}{d\theta} \\ -(\alpha_0+\beta_0)\left(\frac{\partial \mathbf{g}}{\partial\theta}+\frac{\partial \mathbf{t}}{\partial\theta}\right)+\left(\frac{\partial C_2}{\partial\theta}+\frac{\partial E_2}{\partial\theta}\right)\mathbf{r}+(C_2+E_2)\frac{d\mathbf{r}}{d\theta} \end{array}\right]^{(j)} \begin{bmatrix} \delta z \\ \delta p \\ \delta q \\ \delta\beta_0 \\ \delta\theta \end{bmatrix}^{(j)} \\
& = r^{(j)}(\beta_0, \theta, z, \mathbf{r}, p, q). \qquad (7.29)
\end{aligned}
$$

The values of $z, p, q, \beta_0$ and $\theta$ at the $(j+1)$th iteration are

$$\begin{bmatrix} z \\ p \\ q \\ \beta_0 \\ \theta \end{bmatrix}^{(j+1)} = \begin{bmatrix} z \\ p \\ q \\ \beta_0 \\ \theta \end{bmatrix}^{(j)} + \begin{bmatrix} \delta z \\ \delta p \\ \delta q \\ \delta \beta_0 \\ \delta \theta \end{bmatrix}^{(j)},$$

and the initial values in these iterations are

$$z^{(0)} = 0, \qquad p^{(0)} = 0, \qquad q^{(0)} = 0, \qquad \beta_0^{(0)} = 0, \qquad \theta^{(0)} = \theta_0.$$

Clearly it can be seen that (7.29) is of the form

$$Cy = g,$$

where $C \in \mathbb{R}^{(m+n+2)\times(2m+2n-2d+6)}$, $y \in \mathbb{R}^{2m+2n-2d+6}$, $g = \in \mathbb{R}^{m+n+2}$,

$$C = \left[\begin{array}{c|ccc} Y & -S & 0_{m+1,n+1} & 0_{m+1,1} \\ & 0_{n+1,m+1} & -(\alpha_0+\beta_0)T & -(\mathbf{g}+\mathbf{t}) \\ & & -\left(\frac{\partial \mathbf{f}}{\partial\theta}+\frac{\partial \mathbf{s}}{\partial\theta}\right)+\left(\frac{\partial C_1}{\partial\theta}+\frac{\partial E_1}{\partial\theta}\right)\mathbf{r}+(C_1+E_1)\frac{d\mathbf{r}}{d\theta} \\ & & -(\alpha_0+\beta_0)\left(\frac{\partial \mathbf{g}}{\partial\theta}+\frac{\partial \mathbf{t}}{\partial\theta}\right)+\left(\frac{\partial C_2}{\partial\theta}+\frac{\partial E_2}{\partial\theta}\right)\mathbf{r}+(C_2+E_2)\frac{d\mathbf{r}}{d\theta} \end{array}\right]^{(j)}, \tag{7.30}$$

$$y = \begin{bmatrix} \delta z^{(j)} \\ \delta p^{(j)} \\ \delta q^{(j)} \\ \delta \beta_0^{(j)} \\ \delta \theta^{(j)} \end{bmatrix} \quad \text{and} \quad g = r^{(j)}(\beta_0, \theta, z, \mathbf{r}, p, q).$$

Since it is required to move the given inexact polynomials the minimum amount, the function

$$\left\| \begin{bmatrix} z^{(j+1)} - z^{(0)} \\ p^{(j+1)} - p^{(0)} \\ q^{(j+1)} - q^{(0)} \\ \beta_0^{(j+1)} - \beta_0^{(0)} \\ \theta^{(j+1)} - \theta_0 \end{bmatrix} \right\| = \left\| \begin{bmatrix} z^{(j)} + \delta z^{(j)} \\ p^{(j)} + \delta p^{(j)} \\ q^{(j)} + \delta q^{(j)} \\ \beta_0^{(j)} + \delta \beta_0^{(j)} \\ \theta^{(j)} + \delta \theta^{(j)} - \theta_0 \end{bmatrix} \right\| := \|Ey - f\|,$$

must be minimised, where

$$E = I_{2m+2n-2d+6}, \quad \text{and} \quad f = -\begin{bmatrix} z^{(j)} & p^{(j)} & q^{(j)} & \beta_0^{(j)} & \theta^{(j)} - \theta_0 \end{bmatrix}^T. \tag{7.31}$$

It follows that the SNTLN method yields the following LSE problem

$$\min_y \|Ey - f\| \quad \text{subject to} \quad Cy = g. \tag{7.32}$$

An important issue that need to be addressed is the computation of the initial value of $\mathbf{r}(\theta)$. An estimate of this value can be obtained from the least squares solution of

(7.17),

$$\mathbf{r}^{(0)}(\theta_0) \approx \begin{bmatrix} C_1(c,\theta_0) \\ C_2(e,\theta_0) \end{bmatrix}^{\dagger} \begin{bmatrix} \mathbf{f}(\theta_0) \\ \alpha_0 \mathbf{g}(\theta_0) \end{bmatrix}. \tag{7.33}$$

This initial value $\mathbf{r}^{(0)}(\theta_0)$ of $\mathbf{r}(\theta)$, however, requires initial estimates of the coefficients of $u_\theta(w)$ and $v_\theta(w)$, in order to compute $c$ and $e$ whose coefficients are defined in (7.16). This issue is now considered.

It follows from (7.15) that

$$v_{\theta_0}(w)\, f_{\theta_0}(w) = u_{\theta_0}(w)\, g_{\theta_0}(w),$$

and these polynomial products can be written in matrix-vector form as,

$$\begin{bmatrix} C & D \end{bmatrix} \begin{bmatrix} \mathbf{v}_{\theta_0} \\ -\mathbf{u}_{\theta_0} \end{bmatrix} = S_d \begin{bmatrix} \mathbf{v}_{\theta_0} \\ -\mathbf{u}_{\theta_0} \end{bmatrix} \approx 0, \tag{7.34}$$

where $C \in \mathbb{R}^{(m+n-d+1)\times(n-d+1)}$ and $D \in \mathbb{R}^{(m+n-d+1)\times(m-d+1)}$ are the Cauchy matrices whose entries are the coefficients of $f_{\theta_0}(w)$ and $g_{\theta_0}(w)$, respectively, $\mathbf{v}_{\theta_0}$ and $\mathbf{u}_{\theta_0}$, are the vectors of the coefficients of $v_{\theta_0}$ and $u_{\theta_0}$, respectively, and $S_d = S_d(f_{\theta_0}, \alpha_0\, g_{\theta_0}) \in \mathbb{R}^{(m+n-d+1)\times(m+n-2d+2)}$ is the $d$th Sylvester subresultant matrix. The approximation in (7.34) can be written as

$$Ax \approx b, \tag{7.35}$$

where $A \in \mathbb{R}^{(m+n-d+1)\times(m+n-2d+1)}$, $b \in \mathbb{R}^{m+n-d+1}$ is the column of $S_d(f_{\theta_0}, \alpha_0 g_{\theta_0})$ that yields the minimum error in (7.35) [1]. Initial estimates for the vectors $\mathbf{u}_{\theta_0}$ and $\mathbf{v}_{\theta_0}$ can be obtained from the least squares solution of (7.35)

$$x = A^{\dagger} b. \tag{7.36}$$

Algorithm 7.4 shows the application of the method of SNTLN to the APF of two inexact polynomials $f(y)$ and $g(y)$ for the computation of a structured low rank approximation of their Sylveste matrix $S(f, g)$.

---

**Algorithm 7.4: A structured low rank approximation of the Sylvester matrix using APF**

**Input**

**(1)** $f = f(y)$ and $g = g(y)$, the inexact polynomials whose degrees are $m$ and $n$, respectively.

**(2)** $\alpha_0$ and $\theta_0$, the initial values of $\theta$ and $\alpha$, respectively.

**(3)** $d \leq \min(m, n)$, the degree of the AGCD of $f(y)$ and $g(y)$.

**(4)** $q$, the index of the optimal column for the computation of the degree of an AGCD.

**Output** A structured low rank approximation of $S(f, g)$ whose rank is equal to $m + n - d$, and an AGCD of $f(y)$ and $g(y)$.

---

[1] Sections 6.2.2 and 6.2.3 describe two methods for the selection of this column.

**Begin**

1. % Initialisation

**(a.1)** Initialise the following variables with zeros,

$$\beta_0^{(0)},\ z^{(0)},\ p^{(0)},\ q^{(0)},\ \mathbf{s},\ \mathbf{t},\ \frac{\partial \mathbf{s}}{\partial \theta}\ \frac{\partial \mathbf{t}}{\partial \theta},\ E,\ \frac{\partial E}{\partial \theta},\ f,$$

and set $\theta = \theta_0$ and $\alpha = \alpha_0$.

**(a.2)** Form the vectors $\mathbf{f}(\theta)$ and $\mathbf{g}(\theta)$.

**(a.3)** Calculate the coefficients of $u_\theta(w)$ and $v_\theta(w)$, from in (7.36).

**(a.4)** Form the matrices $C_1(c, \theta)$ and $C_2(e, \theta)$, and their derivatives.

**(a.5)** Calculate the initial values of the AGCD, $\mathbf{r}^{(0)}(\theta)$ from (7.33) and its derivative $\frac{dr}{d\theta}$, and the residual $b = r^{(0)}(0, \theta_0, 0, \mathbf{r}^{(0)}(\theta_0), 0, 0)$,

$$r(0, \theta_0, 0, \mathbf{r}^{(0)}(\theta_0), 0, 0) \ = \ \begin{bmatrix} \mathbf{f}(\theta_0) \\ \alpha_0 \mathbf{g}(\theta_0) \end{bmatrix} - \begin{bmatrix} C_1(\mathbf{c}, \theta_0) \\ C_2(\mathbf{e}, \theta_0) \end{bmatrix} \mathbf{r}^{(0)}(\theta_0).$$

**(a.6)** Calculate $Y(\mathbf{r}^{(0)}, \theta))$, defined in (7.27).

**(a.7)** Evaluate $\frac{\partial \mathbf{f}}{\partial \theta}$ and $\frac{\partial \mathbf{g}}{\partial \theta}$ at $\theta = \theta_0$.

**(a.8)** Form the diagonal matrices $S$ and $T$, defined in (7.22).

**(a.9)** Form $C$ and $E$, defined in (7.30) and (7.31), respectively.

**(a.10)** % Solve the LSE problem (7.32) using QR.

Initialise the iteration counter, iterations=0.

**Repeat**

**(b.1)** Compute the $QR$ decomposition of $C^T$,

$$C^T = QR = Q \begin{bmatrix} R_1 \\ 0 \end{bmatrix}.$$

**(b.2)** Set $w_1 = R_1^{-T} b$.

**(b.3)** Partition $EQ$ as

$$EQ = \begin{bmatrix} E_1 & E_2 \end{bmatrix},$$

such that

$$E_1 \in \mathbb{R}^{(2m+2n-2d+6)\times(m+n+2)}, \text{ and } E_2 \in \mathbb{R}^{(2m+2n-2d+6)\times(m+n-2d+4)}.$$

**(b.4)** Compute

$$w_2 = E_2^{\dagger} (f - E_1 w_1).$$

**(b.5)** Compute the solution

$$y = Q \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}.$$

**(b.6)** Set $z := z + \delta z$, $p := p + \delta p$, $q := q + \delta q$, and

$\beta_0 := \beta_0 + \delta\beta_0$, $\theta := \theta + \delta\theta$.

**(b.7)** Update $\mathbf{C}$ from the updated values of:

$\mathbf{f}(\theta), \mathbf{g}(\theta)$, $B(c, e, \theta)$, $\frac{\partial \mathbf{f}}{\partial \theta}$, $\frac{\partial \mathbf{g}}{\partial \theta}$ and $\frac{\partial B}{\partial \theta}$ from $\theta$.

$S$ and $T$ defined in (7.22) and thus $\mathbf{s}(p, \theta)$ and $\frac{\partial \mathbf{s}}{\partial \theta}$ from $p$ and $\theta$, and

$\mathbf{t}(q, \theta)$ and $\frac{\partial \mathbf{t}}{\partial \theta}$ from $q$ and $\theta$.

$E(z, \theta)$ and $\frac{\partial E(z,\theta)}{\partial \theta}$ from $z$ and $\theta$.

$\mathbf{r}(\theta)$ and its derivative $\frac{d\mathbf{r}(\theta)}{d\theta}$ from $\theta$.

$Y(\mathbf{r}, \theta)$ from $\mathbf{r}$ and $\theta$.

**(b.8)** Calculate the residual $r(\beta_0, \theta, z, \mathbf{r}, p, q)$, defined in (7.19), and thus update $g$.

**(b.9)** Update $f$, which is defined in (7.31), from $z, p, q, \theta$ and $\beta_0$.

**(b.10)** Calculate

$$e_r := \begin{bmatrix} \mathbf{f}(\theta) + \mathbf{s}(p, \theta) \\ (\alpha_0 + \beta_0)\,(\mathbf{g}(\theta) + \mathbf{t}(q, \theta)) \end{bmatrix}.$$

**Until** $\frac{\|r(\beta_0,\theta,z,\mathbf{r},p,q)\|}{\|e_r\|} \leq 10^{-12}$ **or** Iteration $> 50$.

**End**

---

## 7.5 Examples

This section contains two examples that show the use of the method of SNTLN for the construction of a structured low rank approximation $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$, of $S(f, g)$ of two inexact polynomials $f = f(y)$ and $g = g(y)$, using the APF of $f(y)$ and $g(y)$, where $\alpha^*$ and $\theta^*$ are the values of $\alpha$ and $\theta$ at the termination of Algorithm 7.4. The

same notation that was used in Section 7.3 is used in these examples.

**Example 7.3.** Consider the polynomials

$$\begin{aligned}\hat{f}(y) &= (y+9.2934)^8(y-4.8386)^3(y-2.8515)^8(y-3.0467)^5,\\ \hat{g}(y) &= (y+9.2934)^8(y-4.8386)^3(y+8.9904)^5(y+7.5947)^5,\end{aligned}$$

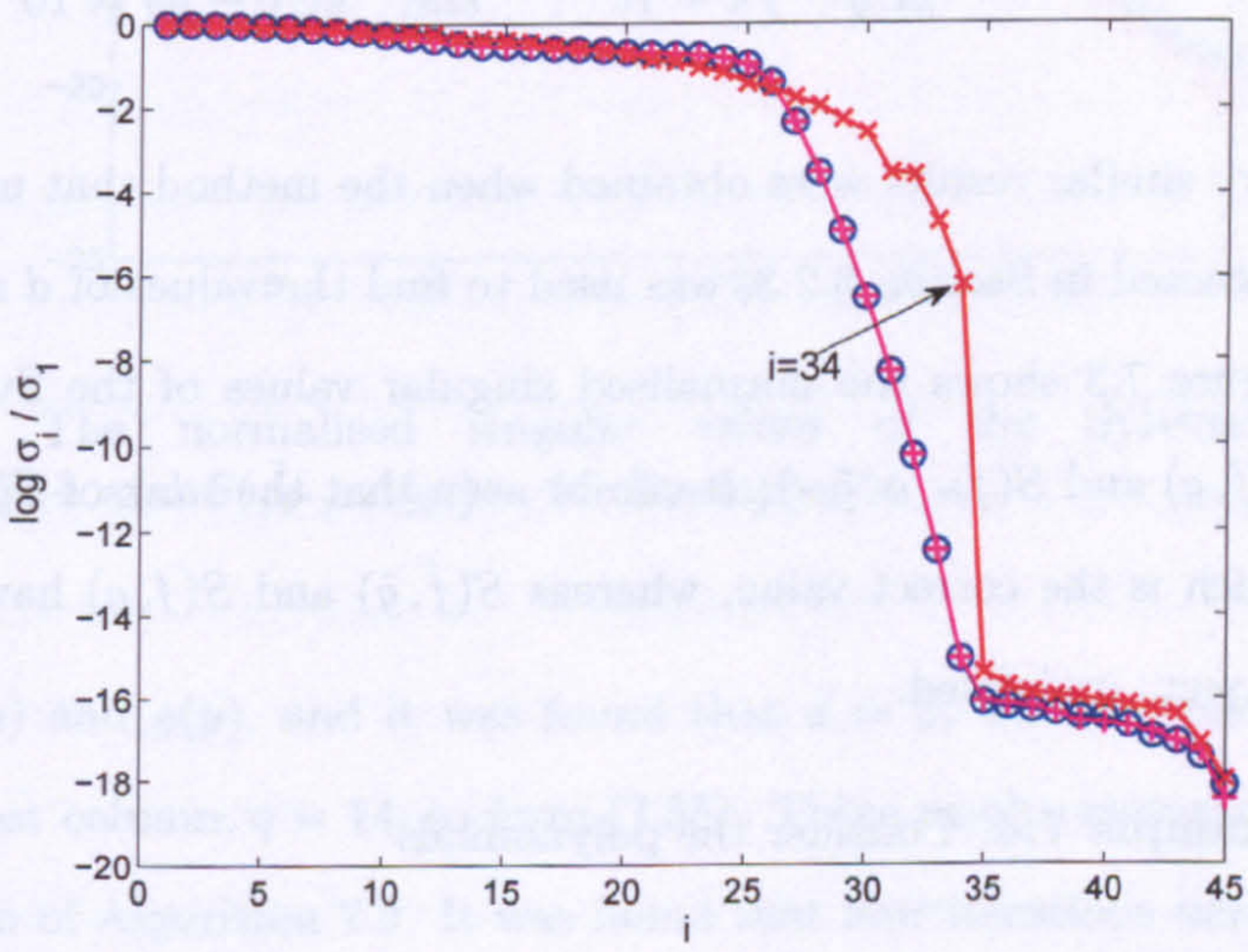



Figure 7.3: The normalised singular values of the Sylvvester matrices $S(f,g)$ o, $S(\hat{f},\hat{g})+$ and $S(\tilde{f}_{\theta^*},\alpha^*\tilde{g}_{\theta^*})\times$, for Example 7.3.

whose Sylvester matrix is of order 45, and since the degree of their GCD is equal to 11, the theoretically exact rank of $S(\hat{f},\hat{g})$ is equal to $45-11=34$. Noise with componentwise signal-to-noise ratio of $10^8$ was added to the coefficients of these polynomials, which were then normalised, thereby yielding the polynomials $f(y)$ and $g(y)$.

The method of the first principal angle, which is discussed in Section 6.2.2, was used to calculate the degree of an AGCD of $f(y)$ and $g(y)$, and it was found that $d = 11$, which is correct, and the index of the best column that formed (7.35), was 16. These results were then used in the implementation of Algorithm 7.2. It was found that two iterations were required for the solution of the LSE problem (7.32) to compute $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$, where the following values were obtained at the termination of Algorithm 7.2:

$$\|Ey - f\| = 10^{-2.1} \quad \text{and} \quad \|Cy - g\| = 10^{-13.1}.$$

Very similar results were obtained when the method that uses the residual, which is discussed in Section 6.2.3, was used to find the values of $d$ and $q$.

Figure 7.3 shows the normalised singular values of the Sylvester matrices $S(\hat{f}, \hat{g})$, $S(f, g)$ and $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$. It can be seen that the rank of $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$ is equal to 34, which is the correct value, whereas $S(\hat{f}, \hat{g})$ and $S(f, g)$ have full rank, which is not correct. undefined. □

**Example 7.4.** Consider the polynomials

$$\begin{aligned} \hat{f}(y) &= (y + 6.5914)^7 (y + 4.8442)^5 (y + 2.0640)^{10} (y + 8.5201)^5, \\ \hat{g}(y) &= (y + 6.5914)^2 (y + 4.8442)^4 (y - 5.1622)^2, \end{aligned}$$

whose Sylvester matrix is of order 35, and since the degree of their GCD is equal to 6, the theoretically exact rank of $S(\hat{f}, \hat{g})$ is equal to $35 - 6 = 29$. Noise with componentwise signal-to-noise ratio of $10^6$ was added to the coefficients of these polynomials, which were then normalised, thereby yielding the polynomials $f(y)$ and $g(y)$.

The method of the first principal angle was used to calculate the degree $d$ of an

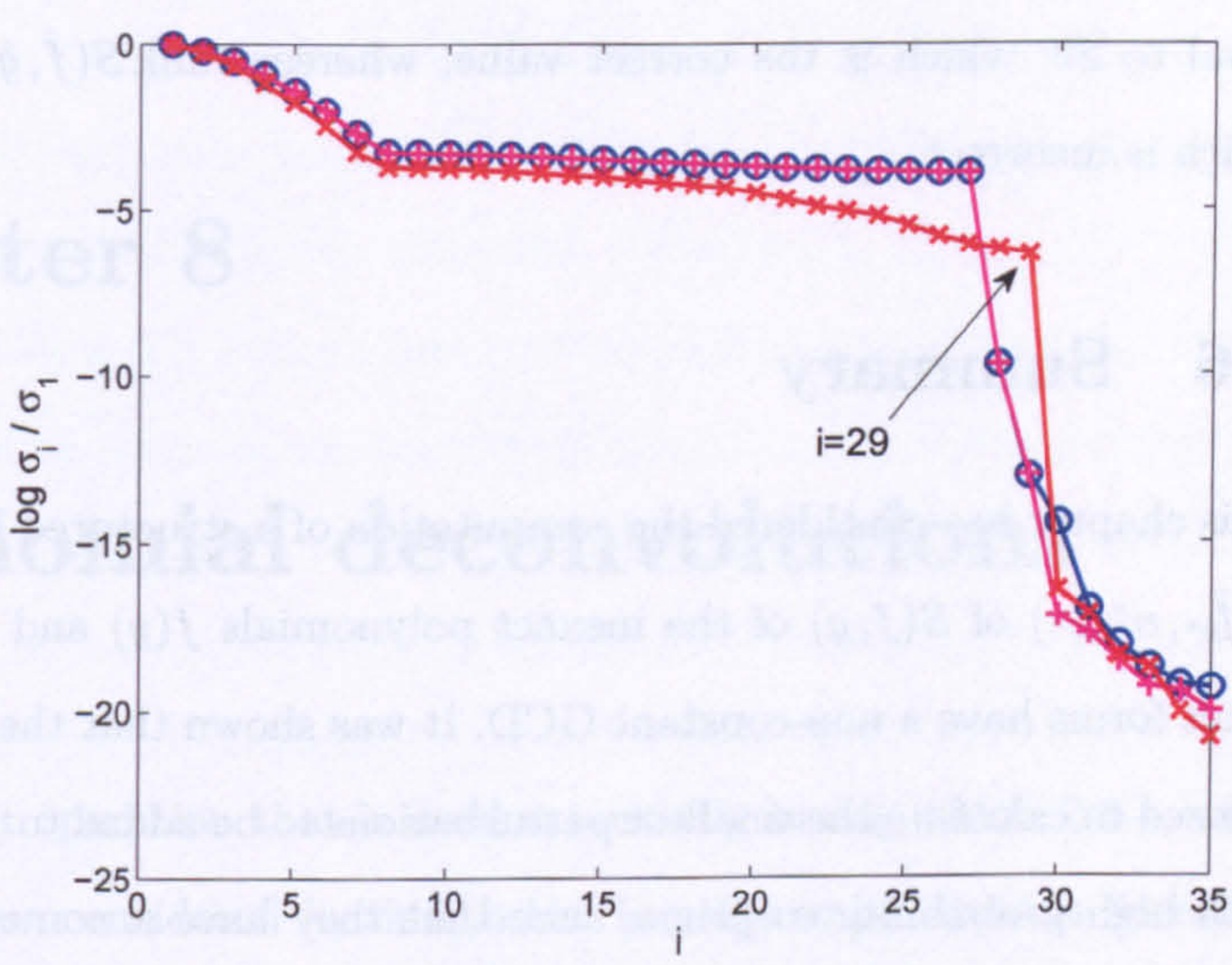



Figure 7.4: The normalised singular values of the Sylvvester matrices $S(f,g)$ o, $S(\hat{f},\hat{g})+$ and $S(\tilde{f}_{\theta^*},\alpha^*\tilde{g}_{\theta^*})\times$, for Example 7.4.

AGCD of $f(y)$ and $g(y)$, and it was found that $d = 6$, which is correct, and the index of the best column, $q = 14$, to form (7.35). These results were then used in the implementation of Algorithm 7.2. It was found that four iterations were required for the solution of the LSE problem (7.32), to compute $S(\tilde{f}_{\theta^*},\alpha^*\tilde{g}_{\theta^*})$, where the following values were obtained at the termination of Algorithm 7.2:

$$\|Ey - f\| = 10^{-0.8} \quad \text{and} \quad \|Cy - g\| = 10^{-15.8}.$$

Very similar results were obtained when the method that uses the residual was used to find the values of $d$ and $q$.

Figure 7.4 shows the normalised singular values of the Sylvester matrices $S(\hat{f}, \hat{g})$, $S(f, g)$ and $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$. Clearly, it can be seen that the rank of $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$ is equal to 29, which is the correct value, whereas $\operatorname{rank} S(\hat{f}, \hat{g}) = \operatorname{rank} S(f, g) = 27$, which is incorrect. □

## 7.6 Summary

This chapter has considered the computation of a structured low rank approximation $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$ of $S(f, g)$ of the inexact polynomials $f(y)$ and $g(y)$, whose theoretical exact forms have a non-constant GCD. It was shown that the method of SNTLN can be used to calculate the smallest perturbations to be added to $f(y)$ and $g(y)$, which are with high probability co-prime, such that they have a non-constant GCD of degree $d$. The perturbed polynomials were called the corrected polynomials $\tilde{f}_{\theta^*}(w)$ and $\tilde{g}_{\theta^*}(w)$, and they can be used to calculate an AGCD of $f(y)$ and $g(y)$. Two SNTLN-based methods for the computation of $S(\tilde{f}_{\theta^*}, \alpha^* \tilde{g}_{\theta^*})$, were given. The first applies the method of SNTLN to the Sylvester matrix of $f(y)$ and $g(y)$, after preprocessing them by the methods discussed in Chapter 4 to obtain $f_{\theta_0}(w)$ and $\alpha_0 g_{\theta_0}(w)$. The second method also considers the preprocessed polynomials but it applies the method of SNTLN to the approximate factorisation of these polynomials, instead of to their Sylvester matrix. The second method explicitly computes a GCD of the corrected polynomials, whereas the Sylvester matrix requires further computations to compute a GCD of the corrected polynomials. Both methods yield excellent results. However, if the GCD is required explicitly then it is better to use the second method. Moreover, more experiments must be performed in order to compare them for the computation of an AGCD.

# Chapter 8

# Polynomial deconvolutions

An important operation that is used extensively in Algorithm 2.3.1, which describes the root solver considered in this thesis, is polynomial division (deconvolution). It was noted in Section 2.3 that this operation is ill-posed because even if $f(y)/g(y)$ is a polynomial, the ratio $(f(y)+\delta f(y))/(g(y)+\delta g(y))$ is, with high probability, a rational function, for arbitrary $\delta f(y)$ and $\delta g(y)$. Algorithm 2.3.1 requires, however, that this ratio reduce to a polynomial and not a rational function. This problem can be solved by perturbing the coefficients of the polynomials $f(y)$ and $g(y)$ the minimum amount such that the polynomial in the denominator is an exact divisor of the polynomial in the numerator. This chapter discusses the numerical computation of the two sets of polynomial divisions,

$$\begin{aligned} h_i(y) &= \frac{q_{i-1}(y)}{q_i(y)}, \quad i = 1, \cdots, l, \\ w_i(y) &= \frac{h_i(y)}{h_{i+1}(y)}, \quad i = 1, \cdots, l-1, \end{aligned} \tag{8.1}$$

which are associated with the implementation of Algorithm 2.3.1, where $l$ is the highest root multiplicity. The method of structured total least norm (STLN) is used to calculate the smallest perturbations that must be added to the coefficients of the polynomials in the numerator and denominator such that the polynomial divisions yield a polynomial. Section 8.1 addresses the problem of computing several polynomial deconvolutions simultaneously, and Section 8.2 applies the method of STLN for the computation of the smallest perturbations to be added to the coefficients of the polynomials $q_{i-1}(y)$ and $q_i(y)$ such that the polynomial division yields a polynomial rather than a rational function. Since $h_i(y)/h_{i+1}(y)$ is of the same form as $q_{i-1}(y)/q_i(y)$, the same procedure is also applicable for $h_i(y)/h_{i+1}(y)$.

## 8.1 Problem statement

Consider the first set of polynomial deconvolutions in (8.1),

$$h_i(y) = \frac{q_{i-1}(y)}{q_i(y)}, \quad i = 1, \cdots, l. \tag{8.2}$$

If the ratio $q_{i-1}(y)/q_i(y)$ is a polynomial, a small level of noise in the coefficients of either polynomial is able to transform this ratio into a rational function. A good solution for this problem can be obtained by slightly perturbing $q_{i-1}(y)$ and $q_i(y)$ such that the perturbed form of $q_i(y)$ is an exact divisor of of the perturbed form of $q_{i-1}(y)$. Moreover, it is noted that $q_i(y)$ occurs in the $i$th and $(i+1)$ deconvolutions, in Algorithm 2.3.1, and computing the polynomial deconvolutions in (8.2) simultaneously allows this coupled form to be preserved. These simultaneous computations

require that the following variables be defined,

$$\begin{aligned} m_i &= \deg q_i(y), \quad i = 0, \cdots, l, \\ n_i &= \deg h_i(y), \quad i = 1, \cdots, l, \end{aligned}$$

such that

$$M = \sum_{i=0}^{l-1}(m_i + 1), \quad M_1 = \sum_{i=0}^{l}(m_i + 1), \quad M_1 = M + (m_l + 1),$$

and

$$N = \sum_{i=1}^{l}(n_i + 1).$$

The set of polynomial deconvolutions in (8.2) can be written in matrix form as

$$\mathbf{C}(\mathbf{q})\mathbf{h} = \mathbf{q},$$

$$\begin{bmatrix} C_1(\mathbf{q}_1) & & & & \\ & C_2(\mathbf{q}_2) & & & \\ & & \ddots & & \\ & & & C_{l-1}(\mathbf{q}_{l-1}) & \\ & & & & C_l(\mathbf{q}_l) \end{bmatrix} \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \vdots \\ \mathbf{h}_{l-1} \\ \mathbf{h}_l \end{bmatrix} = \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_{l-2} \\ \mathbf{q}_{l-1} \end{bmatrix}, \tag{8.3}$$

where $\mathbf{C} = \mathbf{C}(\mathbf{q}) \in \mathbb{R}^{M \times N}$, $\mathbf{h} \in \mathbb{R}^N$, $\mathbf{q} \in \mathbb{R}^M$, and

$$\begin{aligned} C_i(\mathbf{q}_i) &\in \mathbb{R}^{(m_{i-1}+1)\times(n_i+1)}, \quad i = 1, \cdots, l, \\ \mathbf{q}_i &\in \mathbb{R}^{m_{i-1}+1}, \quad i = 0, \cdots, l, \\ \mathbf{h}_i &\in \mathbb{R}^{n_i+1}, \quad i = 1, \cdots, l. \end{aligned}$$

The simplest solution for (8.3) is the least squares (LS) solution,

$$\mathbf{h} = \mathbf{C}^{\dagger}\mathbf{q}.$$

However, it is assumed the data is inexact and the residual that is associated with this LS solution is

$$r = \left\| \left( I - \mathbf{C}(\mathbf{q})\mathbf{C}(\mathbf{q})^{\dagger} \right) \mathbf{h} \right\| > 0,$$

from which it follows that the contents of the vector $\mathbf{h}$ are coefficients of polynomial approximations of rational functions because $q_{i-1}(y)/q_i(y)$, $i = 1, \cdots, l$ are not polynomials. A better solution is obtained when the coefficient matrix and the right hand side of (8.3) are slightly perturbed such that it has an exact solution. These perturbations can be calculated using the method of STLN, which is considered in the next section.

## 8.2 STLN for polynomial deconvolutions

The application of the method of STLN for solving (8.3) requires that a structured matrix be added to the coefficient matrix, and a structured vector be added to the right hand side of this equation.

Let $\mathbf{z}_i \in \mathbb{R}^{m_i+1}$ be the vector of perturbations added to the vector $\mathbf{q}_i$ that contains

the coefficients of the polynomial $q_i(y)$, $i = 0, \cdots, l$, that is,

$$\begin{aligned}
\mathbf{z}_0 &= [\, z_0, \ \cdots, \ z_{m_0} \,]^T \in \mathbb{R}^{m_0+1}, \\
\mathbf{z}_1 &= [\, z_{m_0+1}, \ \cdots, \ z_{m_0+m_1+1} \,]^T \in \mathbb{R}^{m_1+1}, \\
&\vdots \\
\mathbf{z}_l &= [\, z_M, \ \cdots, \ z_{M_1-1} \,]^T \in \mathbb{R}^{m_l+1}.
\end{aligned}$$

Each Cauchy matrix $C_i(\mathbf{q}_i)$, $i = 1, \cdots, l$, in (8.3) is added to a Cauchy matrix $E_i(\mathbf{z}_i) \in \mathbb{R}^{(m_{i-1}+1)\times(n_i+1)}$ of structured perturbations. Thus $C_i(\mathbf{q}_i) + E_i(\mathbf{z}_i)$, for $i = 0, \cdots, l$, form the entries of the matrix of the perturbed coefficients. Therefore, the perturbed form $B(\mathbf{z}_1, \cdots, \mathbf{z}_d) \in \mathbb{R}^{M\times N}$ of the coefficient matrix in (8.3) is

$$\begin{aligned}
B(\mathbf{z}_1, \cdots, \mathbf{z}_l) &= C(\mathbf{q}_1, \cdots, \mathbf{q}_l) + E(\mathbf{z}_1, \cdots, \mathbf{z}_l) \\
&= \begin{bmatrix} C_1(\mathbf{q}_1) & & & & \\ & C_2(\mathbf{q}_2) & & & \\ & & \ddots & & \\ & & & C_{l-1}(\mathbf{q}_{l-1}) & \\ & & & & C_l(\mathbf{q}_l) \end{bmatrix} + \\
&\quad \begin{bmatrix} E_1(\mathbf{z}_1) & & & & \\ & E_2(\mathbf{z}_2) & & & \\ & & \ddots & & \\ & & & E_{l-1}(\mathbf{z}_{l-1}) & \\ & & & & E_l(\mathbf{z}_l) \end{bmatrix}.
\end{aligned} \tag{8.4}$$

The perturbations added to the coefficients of the polynomial $q_i(y)$ in the matrix $C_i(\mathbf{q}_i)$ are also added to the vector of coefficients $\mathbf{q}_i$ of $q_i(y)$ on the right hand side. Therefore, the perturbed form of the right hand side of (8.3) is

$$\begin{bmatrix} \mathbf{q}_0 + \mathbf{z}_0 \\ \mathbf{q}_1 + \mathbf{z}_1 \\ \vdots \\ \mathbf{q}_{l-2} + \mathbf{z}_{l-2} \\ \mathbf{q}_{l-1} + \mathbf{z}_{l-1} \end{bmatrix} = \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_{l-2} \\ \mathbf{q}_{l-1} \end{bmatrix} + \begin{bmatrix} I_M & \vdots & 0 \end{bmatrix} \begin{bmatrix} \mathbf{z}_0 \\ \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_{l-1} \\ \mathbf{z}_l \end{bmatrix} = \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_{l-2} \\ \mathbf{q}_{l-1} \end{bmatrix} + P\mathbf{z}, \tag{8.5}$$

where

$$P = \begin{bmatrix} I_M & \vdots & 0 \end{bmatrix} \quad \text{and} \quad \mathbf{z} = [\, \mathbf{z}_0,\ \mathbf{z}_1,\ \cdots,\ \mathbf{z}_l \,]^T \in \mathbb{R}^{M_1}.$$

It therefore follows from (8.4) and (8.5) that the perturbed form of (8.3) is,

$$\Big(C(\mathbf{q}_1, \cdots, \mathbf{q}_l) + E(\mathbf{z}_1, \cdots, \mathbf{z}_l)\Big)\mathbf{h} = \mathbf{q} + P\mathbf{z}, \tag{8.6}$$

where

$$\mathbf{h} = \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \vdots \\ \mathbf{h}_{l-1} \\ \mathbf{h}_l \end{bmatrix} \in \mathbb{R}^N \quad \text{and} \quad \mathbf{q} = \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_{l-2} \\ \mathbf{q}_{l-1} \end{bmatrix} \in \mathbb{R}^M.$$

Equation (8.6) is non-linear and can be solved by the Newton-Raphson method [69] for the vector $\mathbf{h}$ and the vector of the structured perturbations $\mathbf{z}$. The residual that is associated with an approximate solution of (8.6) is

$$r(\mathbf{z}) = (\mathbf{q} + P\mathbf{z}) - \Big(C(\mathbf{q}_1, \cdots, \mathbf{q}_l) + E(\mathbf{z}_1, \cdots, \mathbf{z}_l)\Big)\mathbf{h}.$$

A first order Taylor expansion of $r(\mathbf{z})$ yields

$$\begin{aligned} r(\mathbf{z}+\delta\mathbf{z}) &= \Big(\mathbf{q} + P(\mathbf{z}+\delta\mathbf{z})\Big) \\ &\quad -\Big(C(\mathbf{q}_1, \cdots, \mathbf{q}_l) + E(\mathbf{z}_1+\delta\mathbf{z}_1, \cdots, \mathbf{z}_l+\delta\mathbf{z}_l)\Big)(\mathbf{h}+\delta\mathbf{h}) \\ &= r(\mathbf{z}) + P\delta\mathbf{z} - \Big(C(\mathbf{q}_1, \cdots, \mathbf{q}_l) + E(\mathbf{z}_1, \cdots, \mathbf{z}_l)\Big)\delta\mathbf{h} - \delta E(\mathbf{z}_1, \cdots, \mathbf{z}_l)\mathbf{h}, \end{aligned} \tag{8.7}$$

where

$$\delta\mathbf{E}(\mathbf{z}_1, \cdots, \mathbf{z}_l) = \begin{bmatrix} \delta E_1(\mathbf{z}_1) & & & & \\ & \delta E_2(\mathbf{z}_2) & & & \\ & & \ddots & & \\ & & & \delta E_{l-1}(\mathbf{z}_{l-1}) & \\ & & & & \delta E_l(\mathbf{z}_l) \end{bmatrix}.$$

There exist matrices $Y_i(\mathbf{h}_i) \in \mathbb{R}^{(m_i-1+1)\times(m_i+1)}$, $i = 1, \cdots, l$, such that

$$E_i(\mathbf{z}_i)\mathbf{h}_i = Y_i(\mathbf{h}_i)\mathbf{z}_i \quad \text{and thus} \quad \delta E_i(\mathbf{z}_i)\mathbf{h}_i = Y_i(\mathbf{h}_i)\delta\mathbf{z}_i.$$

It follows that

$$\delta E(\mathbf{z}_1, \cdots, \mathbf{z}_l)\mathbf{h} = Y\delta\mathbf{z}, \tag{8.8}$$

where

$$Y = Y(\mathbf{h}_1, \cdots, \mathbf{h}_l) = \begin{bmatrix} 0 & Y_1(\mathbf{h}_1) & & & \\ 0 & & Y_2(\mathbf{h}_2) & & \\ \vdots & & & \ddots & \\ 0 & & & Y_{l-1}(\mathbf{h}_{l-1}) & \\ 0 & & & & Y_l(\mathbf{h}_l) \end{bmatrix},$$

and

$$\delta\mathbf{z} = \begin{bmatrix} \delta\mathbf{z}_0 \\ \delta\mathbf{z}_1 \\ \vdots \\ \delta\mathbf{z}_{l-1} \\ \delta\mathbf{z}_l \end{bmatrix}.$$

Using (8.8), the expression for $r(\mathbf{z}+\delta\mathbf{z})$ in (8.7) can be simplified to

$$r(\mathbf{z}+\delta\mathbf{z}) = r(\mathbf{z}) - (C+E)\delta\mathbf{h} - (Y-P)\delta\mathbf{z},$$

and therefore it is required to solve

$$\begin{bmatrix} (C+E) & (Y-P) \end{bmatrix} \begin{bmatrix} \delta\mathbf{h} \\ \delta\mathbf{z} \end{bmatrix} = r,$$

which is an under-determine equation, with

$$\left[\begin{array}{cc} (C+E) & (Y-P) \end{array}\right] \in \mathbb{R}^{M\times(N+M_1)},$$

and $r = r(\mathbf{z})$. The $j$th iteration in the Newton-Raphson method for the calculation of $\mathbf{h}$ and $\mathbf{z}$ is,

$$\left[\begin{array}{cc} (C+E) & (Y-P) \end{array}\right]^{(j)} \left[\begin{array}{c} \delta\mathbf{h} \\ \delta\mathbf{z} \end{array}\right]^{(j)} = r^{(j)}. \tag{8.9}$$

Let $\mathbf{z}^{(0)} = 0$ be the initial value of $\mathbf{z}$ and $\mathbf{h}^{(0)}$ be the initial value of $\mathbf{h}$. Since the nearest solution is sought, it is required to minimise

$$\begin{aligned} \left\| \begin{array}{c} \mathbf{h}^{(j+1)} - \mathbf{h}^{(0)} \\ \mathbf{z}^{(j+1)} \end{array} \right\| &= \left\| \begin{array}{c} \mathbf{h}^{(j)} + \delta\mathbf{h}^{(j)} - \mathbf{h}^{(0)} \\ \mathbf{z}^{(j)} + \delta\mathbf{z}^{(j)} \end{array} \right\| \\ &= \left\| F \left[\begin{array}{c} \delta\mathbf{h}^{(j)} \\ \delta\mathbf{z}^{(j)} \end{array}\right] - \left[\begin{array}{c} -(\mathbf{h}^{(j)} - \mathbf{h}^{(0)}) \\ -\mathbf{z}^{(j)} \end{array}\right] \right\|, \end{aligned}$$

subject to (8.9), where $F = I_{N+M_1}$. This is an LSE problem of the form

$$\min \|Fy - s\| \quad \text{subject to} \quad Gy = t, \tag{8.10}$$

where

$$G = \left[ (C+E) \quad (Y-P) \right]^{(j)}, y = \begin{bmatrix} \delta \mathbf{h}^{(j)} \\ \delta \mathbf{z}^{(j)} \end{bmatrix},$$

$$s = \begin{bmatrix} -(\mathbf{h}^{(j)} - \mathbf{h}^{(0)}) \\ -\mathbf{z}^{(j)} \end{bmatrix}, t = r^{(j)},$$

and $\mathbf{h}^{(0)}$ is obtained from the least squares solution of (8.3),

$$\mathbf{h}^{(0)} = \begin{bmatrix} C_1(\mathbf{q}_1) & & & & \\ & C_2(\mathbf{q}_2) & & & \\ & & \ddots & & \\ & & & C_{l-1}(\mathbf{q}_{l-1}) & \\ & & & & C_l(\mathbf{q}_l) \end{bmatrix}^{\dagger} \begin{bmatrix} \mathbf{q}_0 \\ \mathbf{q}_1 \\ \vdots \\ \mathbf{q}_{l-2} \\ \mathbf{q}_{l-1} \end{bmatrix}. \tag{8.11}$$

The application of the QR decomposition to solve the LSE problem (8.10) is shown in Algorithm 8.2.

---

**Algorithm 8.2: $QR$ decomposition for polynomial deconvolution**

**Input** The polynomials $q_i(y), i = 0, \cdots, l$.

**Output** The polynomials $h_i(y), i = 1, \cdots, l$.

**Begin**

1. Set $\mathbf{z} = 0$ and compute $\mathbf{h}^{(0)}$ from (8.11).

2. **repeat**

**(a)** Compute the $QR$ decomposition of $G^T$,

$$G^T = QR = Q \begin{bmatrix} R_1 \\ 0 \end{bmatrix}.$$

**(b)** Set $w_1 = R_1^{-T} r$.

**(c)** Partition $FQ$, into $F_1 \in \mathbb{R}^{(N+M_1)\times M}$ and $F_2 \in \mathbb{R}^{(N+M_1)\times(N+M_1-M)}$

$$FQ = \begin{bmatrix} F_1 & F_2 \end{bmatrix}.$$

**(d)** Compute

$$w_2 = F_2^{\dagger}\Big(s - F_1 w_1\Big).$$

**(e)** Compute the solution

$$y = Q \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}.$$

**(f)** Set $\mathbf{h} := \mathbf{h} + \delta\mathbf{h}$ and $\mathbf{z} := \mathbf{z} + \delta\mathbf{z}$.

**(g)** Update $G, s$ and $t$, then evaluate the residual

$$res = (\mathbf{q} + P\mathbf{z}) - \Big(C(\mathbf{q}_1, \cdots, \mathbf{q}_l) + E(\mathbf{z}_1, \cdots, \mathbf{z}_l)\Big)\mathbf{h}.$$

**until** $\frac{\|res\|}{\|\mathbf{q}+P\mathbf{z}\|} \leq 10^{-12}$.

**End**

---

An important issue that should be addressed is that the polynomials $q_{i+1}(y)$, $i = 0, \cdots, l$, are the AGCDs of the successive polynomials $q_i(y)$ and $q_i^{(1)}(y)$, and each AGCD is computed with different value of $\theta$. For example, if the given polynomial is in the variable $y$, and $\theta_1$ and $\theta_2$ are obtained from the LP problem (4.7), then

$$y = \theta_1 w_1 \qquad w_1 = \theta_2 w_2. \tag{8.12}$$

and

$$\begin{aligned} q_1 &= \text{AGCD}(q_0, q_0^{(1)}), \qquad q_1 = q_1(w_1), \\ q_2 &= \text{AGCD}(q_1, q_1^{(1)}), \qquad q_2 = q_2(w_2). \end{aligned}$$

It is therefore important to transform the polynomials $q_1(w_1)$ and $q_2(w_2)$ to the same independent variable. For example, it follows from (8.12) that

$$y = \theta_1\theta_2 w_2, \quad q_1 = q_1\Big(\frac{y}{\theta_1}\Big) \quad \text{and} \quad q_2 = q_2\Big(\frac{y}{\theta_2\,\theta_2}\Big).$$

The same process is repeated for all $q_i(w_i)$.

## 8.3 Summary

This chapter has discussed the computational implementation of polynomial deconvolution, which is used extensively in the proposed root solver. It has been shown

that this problem is ill-posed because even if this division yields a polynomial, a small random perturbation added to the coefficients of the polynomials in the numerator and denominator, yields, with high probability, a rational function. It has been shown that the method of STLN can be used to impose the requirement that polynomial division yields a polynomial.

# Chapter 9

# Polynomial root solver

This chapter considers the application of the developed root solver, which follows the method described in Algorithm 2.3.1, for the computation of the roots of inexact polynomials, whose exact forms contain multiple roots. This root solver involves the following problems

1. The computation of successive AGCDs.

2. The computation of successive polynomial divisions.

3. The computation of several polynomials, all of whose roots are simple.

The first problem involves two stages:

(a) The computation of the degree of an AGCD.

(b) The computation of the coefficients of this AGCD.

These two stages have been considered in Chapters 6 and 7, respectively. The second problem has been considered in Chapter 8.

The last stage of Algorithm 2.3.1 requires the solution of several polynomial equations, all of whose roots are simple. These roots can be refined by the method of non-linear least squares. The task of computing the simple roots and refining them is considered in the Section 9.1, and Section 9.2 presents some examples.

## 9.1 The computation of the roots and their refinement

The developed root solver that is described in Algorithm 2.3.1 follows the Divide and Conquer Strategy, by which a polynomial that has multiple roots is broken down into several polynomials, each of which only has simple roots. Thus, the last stage of this algorithm requires solving several polynomial equations, all of whose roots are simple and can be computed using classical root solving methods (e.g those described in Chapter 1). The multiplicities of these roots are determined by the successive AGCD computations in Algorithm 2.3.1, and it has been shown in Section 2.3 that the values of these multiplicities follow directly from the index of the second set of polynomial deconvolutions $w_j$, $j = 1, 2, \cdots, r_{max}$, where $r_{max}$ is the highest root multiplicity. Thus, in addition to the computation of the roots of a polynomial, Algorithm 2.3.1 computes the multiplicity structure of this polynomial.

Once the roots and their associated multiplicities are known, the values of the roots can be refined under the constraints of the multiplicity structure. In particular, consider the polynomial $f(y)$, which is defined in (2.7), and let the initial estimates of the $l$ distinct roots of $f(y)$ be $\mathbf{y}_0 = [y_{0,1}, y_{0,2}, \cdots, y_{0,l}]$ and the associated multiplicities be defined by the vector $\mathbf{m} = [m_1, m_2, \cdots, m_l]$. It follows from Kahan's observations,

which are illustrated in Section 2.2, that this multiplicity structure defines the pejorative manifold on which the polynomial $f(y)$ lies. Furthermore, a small displacement of $f(y)$ on this manifold yields a small change in the roots of $f(y)$ under the constraint that the multiplicities of the roots are preserved. A geometrical interpretation of this process is given in Section 2.3.3.

It was shown in Section 2.2 that the pejorative manifold of the polynomial $f(y)$, which is defined in (2.7), is given by Vieta's system (2.7) $\mathbf{P_m(y) = a}$,

$$\mathbf{P_m(y)} = \begin{cases} p_1(\mathbf{y}) & = a_1 \\ p_2(\mathbf{y}) & = a_2 \\ \quad \vdots & \\ p_n(\mathbf{y}) & = a_n \end{cases}, \tag{9.1}$$

and it is required to find the vector $\mathbf{y} = [\ y_1, \ \cdots, y_l\ ]$, with minimum error. Thus, the problem of computing the roots $y_j$, $j = 1, 2, \cdots, l$, is reduced to the minimisation problem,

$$\min_{\mathbf{y}\in\mathbb{R}^l} \|\mathbf{P_m(y)} - \mathbf{a}\|_2^2 = \min_{\mathbf{y}\in\mathbb{R}^l} \sum_{j=1}^{n} (p_j(\mathbf{y}) - a_j)^2,$$

which is a non-linear least squares problem, and can be solve iteratively, using the Gauss-Newton method [11].

Let $J = J(\mathbf{y})$ be the Jacobian matrix of $\mathbf{P_m(y)}$, and the entries of $J(\mathbf{y})$ are $J_{ij} = \frac{\partial p_i(\mathbf{y})}{\partial y_j}$,

$i = 1, \cdots, n$, and $j = 1, \cdots, l$, that is,

$$J = \begin{bmatrix} \frac{\partial p_1(\mathbf{y})}{\partial y_1} & \frac{\partial p_1(\mathbf{y})}{\partial y_2} & \cdots & \frac{\partial p_1(\mathbf{y})}{\partial y_l} \\ \frac{\partial p_2(\mathbf{y})}{\partial y_1} & \frac{\partial p_2(\mathbf{y})}{\partial y_2} & \cdots & \frac{\partial p_2(\mathbf{y})}{\partial y_l} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial p_n(\mathbf{y})}{\partial y_1} & \frac{\partial p_n(\mathbf{y})}{\partial y_2} & \cdots & \frac{\partial p_n(\mathbf{y})}{\partial y_l} \end{bmatrix}.$$

The $(k+1)$th iteration in the Gauss-Newton method is given by,

$$\mathbf{y}_{k+1} = \mathbf{y}_k - (J_k^T J_k)^{-1} J^T \mathbf{r}(\mathbf{y}_k), \tag{9.2}$$

where $\mathbf{r}(\mathbf{y}_k) = \mathbf{P_m}(\mathbf{y}_k) - \mathbf{a}$. The matrix inverse in the iteration (9.2) exists only if $J_k = J(\mathbf{y}_k)$ is non-singular, (i.e. $J_k$ has full rank for all $k$). It is shown in [85] that $J_k$ is non-singular, since the roots $y_1, y_2, \cdots, y_l$ in $\mathbf{P_m}(\mathbf{y})$ are all distinct.

The initial estimates of the distinct roots $y_j$, $j = 1, 2, \cdots, l$ of $f(y)$ are calculated by solving the polynomial equations $w_i(y) = 0$, $i = 1, 2, \cdots, r_{max}$, in Algorithm 2.3.1, all of whose roots are simple, where $r_{max}$ is the highest root multiplicity. Given these initial estimates of the roots of an inexact polynomial $f(y)$, and the associated multiplicity structure, Algorithm 9.1 refines these estimates using the method of non-linear least squares.

---

**Algorithm 9.1: The refinement of the roots**

**Input**

1. The vector $\mathbf{a} \in \mathbb{R}^{n+1}$ of the normalised coefficients of $f(y)$.

2. The initial estimates of the distinct roots $\mathbf{y}_0 = [y_{0,1}, y_{0,2}, \cdots, y_{0,l}]$ of $f(y)$, and the multiplicity $m_j$ of each distinct root $y_{0,j}$, $j = 1, \cdots, l$.

**Output** The refined roots $\mathbf{y} = [y_1, y_2, \cdots, y_l]$ of $f(y)$.

**Begin**

1. Set $k = 0$ and $\mathbf{y}_k = \mathbf{y}_0$.

2. **Repeat**

   (a) Calculate the vector $P_{\mathbf{m}}(\mathbf{y}_k)$ defined in (9.1), and the residual vector,

$$\mathbf{r}_k = P(\mathbf{y}_k) - \mathbf{a}.$$

   (b) Calculate the Jacobian matrix $J_k = J(\mathbf{y}_k)$.

   (c) Calculate $\mathbf{y}_{k+1}$ defined in (9.2).

   (d) Calculate $\mathbf{r}_{k+1}$,

$$\mathbf{r}_{k+1} = P_{\mathbf{m}}(\mathbf{y}_{k+1}) - \mathbf{a}.$$

(e) Calculate the error,

$$e_{k+1} = \frac{\|\mathbf{r}_{k+1} - \mathbf{r}_k\|}{\|\mathbf{r}_k\|}.$$

(f) Increment $k$.

**Until** $e_{k+1} \leq 10^{-14}$.

3. Set $\mathbf{y} = \mathbf{y}_{k+1}$.

---

## 9.2 Results

This section contains some examples that show the results of computing the roots of inexact polynomials using the root solver described in this work. The computation of AGCDs of several pairs of polynomials forms the most crucial stage in this root solver. The degrees of the AGCDs were determined by applying the majority voting principle to the set of the results produced by the methods discussed in Chapter 6. It was shown that an AGCD of two inexact polynomials can be be computed using either the Sylvester matrix or APF of these two inexact polynomials. Although both methods can be used to compute a structured low rank approximation of the Sylvester matrix, the method of APF is superior for the calculation of an AGCD because it yields the AGCD explicitly, and no extra calculation is required. By contrast, the Sylvester matrix must be reduced to an upper triangular form using either the LU or QR decomposition, but this may cause numerical problems.

It was found experimentally that the first AGCD computation must be performed using APF, and that subsequent AGCD computations can be performed using the Sylvester matrix. It is believed that this is because the AGCD computed from the APF in the first AGCD computation is of sufficiently high quality, such that the Sylvester matrix can be used for all subsequent AGCD computations.

Two root solvers were used to test the results produced by the developer root solver; the function roots() in MATLAB and the suite of MATLAB programmes MULTROOT which is developed by Zeng [85], and it is called with the function multroot=(poly, threshold). The first argument poly is the vector of the coefficients of the polynomial, and the second argument threshold is the error tolerance. If the second argument is omitted, then the default value threshold = $10^{-10}$ is used.

The examples included in this section contains three sets of polynomials:

1. The first set considers some of the test polynomials from the test collection in [86], after adding componentwise noise to the coefficients of the polynomials. The developed root solver and MULTROOT work well with this set. MULTROOT requires, however, the noise level in order to produce good results, whereas the developed root solver does not require this information.

2. The second set considers test polynomials including some hard classes of polynomials suggested by the author of this thesis, and the results of each example in this set are compared with the results from MULTROOT, and the function roots(). The developed root solver performs well with the examples given in this set. In contrast to Set 1, it is shown in this set that MULTROOT does not always work well even if the noise level is specified. The function roots() fails also to compute the correct answers of the examples in this set.

3. The last set considers the case when the signal-to-noise ratio of each coefficient of a given polynomial is a random variable between $a$ and $b$, where $b/a = 10^3, 10^4$ or $10^5$. The results of each example in this set are also compared with the results from MULTROOT, and the function roots(). It is shown that MULTROOT fails to compute the roots of the polynomials in this set for all threshold values in the range $a \leq \text{threshold} \leq b$. Similar results were obtained when the function roots() was used. On the other hand, the developed root solver works perfectly as it does not require any knowledge about the noise level.

**Polynomial Set 1:** This set contains three examples from [86]. The roots of the polynomials in this set were computed using both the developed root solver and MULTROOT. Both root finders yield satisfactory results. However the noise level was required to be given for MULTROOT as an input argument threshold, in order to provide good results, whereas it was not required by the developed root solver.

**Example 9.1.** Consider the exact polynomial $\hat{f}_1(y)$ whose roots and multiplicities are given in the first and second columns of Table 9.1, respectively.

Table 9.1: The roots and multiplicities of $\hat{f}_1(y)$ for Example 9.1.

| exact root | exact mult. | computed root | computed mult. | relative error |
|---|---|---|---|---|
| 1 | 8 | 1.000000000e+000 | 8 | 7.129630220e-012 |
| 2 | 16 | -2.000000000000463e+000 | 16 | 2.315925229e-013 |
| 3 | 24 | 2.999999999991630e+000 | 24 | 2.790064476e-012 |

Componentwise noise with $\varepsilon_c = 10^{-10}$ was added to the coefficients of this polynomial to create the inexact form $f_1(y)$ of $\hat{f}_1(y)$. The results of computing the roots of $f_1(y)$, and their corresponding multiplicities, using the root solver described in this work

are given in the third and fourth columns of Table 9.1, respectively. The fifth column of Table 9.1 shows that the relative error in computing each distinct root is between two and three order of magnitude smaller than $\varepsilon_c = 10^{-10}$. Similar results were also found by MULTROOT with the argument threshold $= 10^{-10}$. However, it is shown in Set 2 that in contrast to the developed root solver, if $f_1(y)$ is perturbed by $\varepsilon_c = 10^{-8}$, MULTROOT fails to compute the roots of $f_1(y)$ even if threshold is set equal to $10^{-8}$.

□

**Example 9.2.** Consider the exact polynomial $\hat{f}_2(y)$ whose roots and multiplicities are given in the first and second columns of Table 9.2, respectively.

Table 9.2: The roots and multiplicities of $\hat{f}_2(y)$ for Example 9.2.

| exact root | exact mult. | computed root | computed mult. | relative error |
|---|---|---|---|---|
| 2.727272727e+000 | 2 | 2.727271785e+000 | 2 | 3.455067788e-007 |
| 1.818181818e+000 | 3 | 1.818182510e+000 | 3 | 3.806444900e-007 |
| 9.090909090e-001 | 5 | 9.090908204e-001 | 5 | 9.751830201e-008 |

Componentwise noise with $\varepsilon_c = 10^{-8}$ was added to the coefficients of this polynomial to create the inexact form $f_2(y)$ of $\hat{f}_2(y)$. The results of computing the roots of $f_2(y)$, and their corresponding multiplicities, using the root solver described in this work are given in the third and fourth columns of Table 9.2. The fifth column of Table 9.2 shows that the relative error in computing each distinct root is between one and two order of magnitude 1arger than $\varepsilon_c = 10^{-8}$. Similar results were also found by MULTROOT with the argument threshold $= 10^{-8}$. □

**Example 9.3.** Consider the exact polynomial $\hat{f}_3(y)$ whose roots and multiplicities are given in the first and second columns of Table 9.3, respectively.

Table 9.3: The roots and multiplicities of $\hat{f}_3(y)$ for Example 9.3.

| exact root | exact mult. | computed root | computed mult. | relative error |
|---|---|---|---|---|
| 2.35 | 3 | 2.559999568e+00 | 3 | 1.6855607569e-07 |
| 2.56 | 1 | 2.350000137e+00 | 1 | 5.774327796e-08 |

Componentwise noise with $\varepsilon_c = 10^{-8}$ was added to the coefficients of this polynomial to create the inexact form $f_3(y)$ of $\hat{f}_3(y)$. The results of computing the roots of $f_3(y)$, and their corresponding multiplicities, using the root solver described in this work are given in the third and fourth columns of Table 9.3, respectively. The fifth column of Table 9.3 shows that the relative error in computing each distinct root is acceptable with respect to $\varepsilon_c = 10^{-8}$. Similar results were also found by MULTROOT with the argument threshold $= 10^{-8}$. □

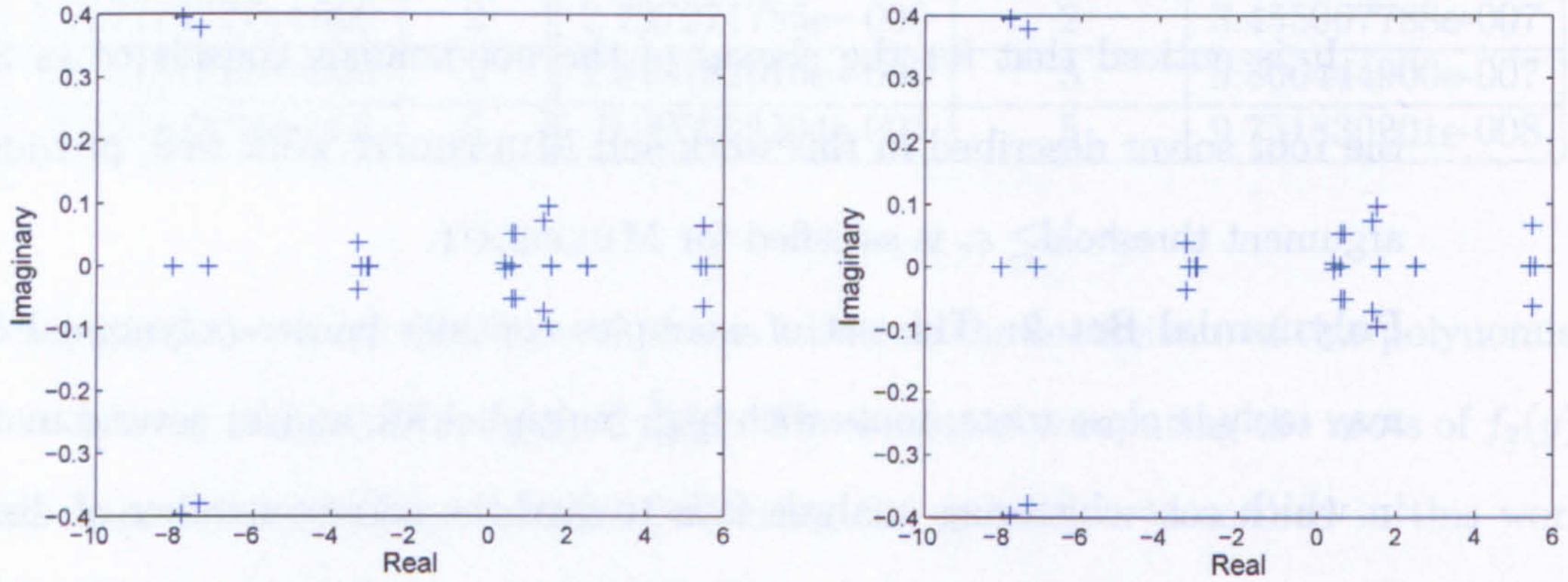
It is noticed that for the classes of the polynomials considered in Set 1, both the root solver described in this work and MULTROOT work well, provided that the argument threshold$\geq \varepsilon_c$ is satisfied for MULTROOT.

**Polynomial Set 2:** This set of examples consider harder polynomial classes that may include close roots, roots with high multiplicities, and/or several multiple roots, in which case clustering analysis fails to give the correct number of distinct roots. This set contains four examples. The first three examples are suggested by this work and the fourth example considers the polynomial $f_1(y)$ in Example 9.1, but with a lower signal-to-noise ratio. Unlike the results in the first set of examples, this set of examples demonstrates that MULTROOT does not always work well if the noise level is specified. On the other, the root solver developed in this thesis performs very well without prior knowledge of the noise level.

**Example 9.4.** Let the exact polynomial $\hat{f}_4(y)$ be defined by the roots and multiplicities given in the first and second columns of Table 9.4, respectively.

Table 9.4: The roots and multiplicities of $\hat{f}_4(y)$ for Example 9.4.

| exact root | exact mult. | computed root | computed mult. | relative error |
|---|---|---|---|---|
| -7.5947e+00 | 6 | -7.59343498e+00 | 6 | 1.66566613e-004 |
| 6.3371e-01 | 5 | 6.33767034e-01 | 5 | 9.00006631e-005 |
| 1.4923e+00 | 5 | 1.49217406e+00 | 5 | 8.43930316e-005 |
| 5.4862e+00 | 4 | 5.48668014e+00 | 4 | 8.75174996e-005 |
| -3.3076e+00 | 3 | -3.10954147e+00 | 3 | 5.98798325e-002 |
| -3.0670e+00 | 2 | -3.36592459e+00 | 2 | 9.74648171e-002 |
| 4.2244e-01 | 2 | 4.22380231e-01 | 2 | 1.41486013e-004 |
| 2.5090e+00 | 2 | 2.50915603e+00 | 2 | 6.21875010e-005 |




(a) (b)

Figure 9.1: The computed roots of $f_4(y)$ in Example 9.4, using (a) MULTROOT, and (b) the MATLAB function roots().

Noise with signal-to-noise ratio $10^6$ was added to the coefficients of this polynomial to create the inexact form $f_4(y)$ of $\hat{f}_4(y)$. The results of computing the roots of $f_4(y)$, and their corresponding multiplicities, using the root solver described in this work

are given in the third and fourth columns of Table 9.4, respectively. The fifth column of Table 9.4 shows the relative error in computing each distinct root. The results of computing the roots of $f_4(y)$, using MULTROOT with threshold $= 10^{-6}$, and roots() are plotted in Figures 9.1 (a) and (b), respectively.

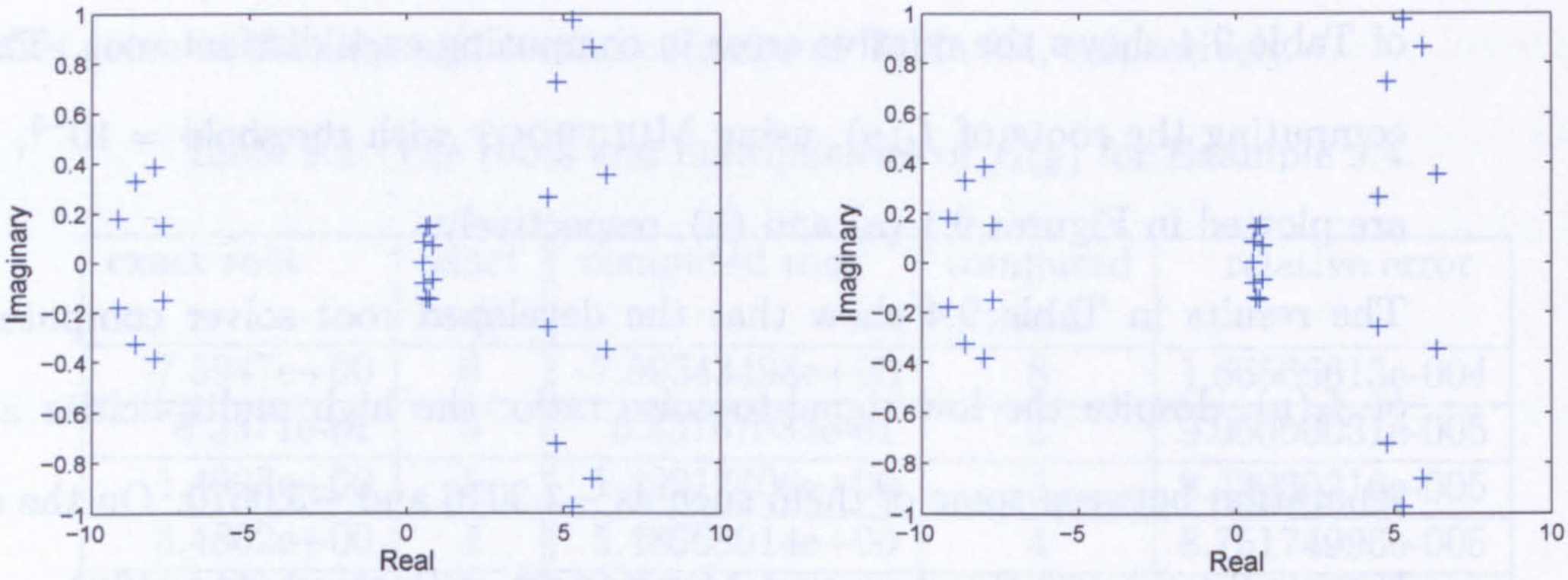
The results in Table 9.4 show that the developed root solver computes the roots of $f_4(y)$, despite the low signal-to-noise ratio, the high multiplicities and the low separation between some of them such as $-3.3076$ and $-3.0670$. On the other hand, Figures 9.1 (a) and (b) show that MULTROOT, with threshold $= 10^{-6}$, and MATLAB return simple roots, and thus the multiplicities of the roots are lost. Considering lower signal to noise ratios such as $\epsilon_c^{-1} \leq 10^5$, however, causes the developed root solver to fail as well. □

**Example 9.5.** This example considers the polynomial $f_5(y)$ whose whose roots and multiplicities are given in the first and second columns of Table 9.5, respectively. Noise with componentwise signal-to-noise ratio $\epsilon_c^{-1} = 10^7$ was added to the coefficients of $f_5(y)$. The results in Table 9.5 show that the roots of $f_5(y)$ were computed with

Table 9.5: The roots and multiplicities of $\hat{f}_5(y)$ for Example 9.5.

| exact root | exact mult. | computed root | computed mult. | relative error |
|---|---|---|---|---|
| 5.36868065e+000 | 10 | 5.36868078e+000 | 10 | 2.32121468e-008 |
| 6.66252854e-001 | 9 | 6.66252846e-001 | 9 | 1.26828432e-008 |
| -8.00831536e+000 | 5 | -8.00831726e+000 | 5 | 2.37076153e-007 |
| -8.99239567e+000 | 3 | -8.99239169e+000 | 3 | 4.43280353e-007 |

good accuracy, despite the high multiplicities and the low signal-to-noise ratio. The results of computing the roots and associated multiplicities of $f_5(y)$ using MULTROOT with threshold $= 10^{-7}$, and MATLAB are shown in Figures 9.2 (a) and (b). As in the




(a) (b)

Figure 9.2: The computed roots of $f_5(y)$ in Example 9.5, using (a) MULTROOT, and (b) the MATLAB function roots().

previous example, it is noted that both MULTROOT and MATLAB fail to compute the correct roots because they return simple roots. Considering lower signal to noise ratio such as $\varepsilon_c^{-1} = 10^6$, however, causes the developed root solver to fail as well. □

**Example 9.6.** Example 9.1 has consider the exact polynomial $\hat{f}_1(y)$ from Zeng's set in [86]. The same polynomial is considered in this example with threshold $= 10^{-8}$ instead of $10^{-10}$.

Table 9.6: The roots and multiplicities of $\hat{f}_1(y)$ for Example 9.6.

| exact root | exact mult. | computed root | computed mult. | relative error |
|---|---|---|---|---|
| 1 | 8 | 1.00000000e+000 | 8 | 1.826900853e-009 |
| 2 | 16 | 2.00000000e+000 | 16 | 5.466993524e-010 |
| 3 | 24 | 2.999999999e+000 | 24 | 2.317150916e-010 |

Componentwise noise with $\varepsilon_c = 10^{-8}$ was added to the coefficients of this polynomial to create the inexact form $f_1(y)$ of $\hat{f}_1(y)$. The results of computing the roots of $f_1(y)$,

and their corresponding multiplicities, using the root solver described in this work are given in the third and fourth columns of Table 9.6, respectively. The fifth column of Table 9.6 shows that the relative error in computing each distinct root is between one and two order of magnitude smaller than $\epsilon_c = 10^{-8}$. On the other hand, both the root solver MULTROOT with threshold $= 10^{-8}$, and the function roots() returned simple complex conjugate pairs of roots. □

Despite the difficulty of the polynomial classes and the low signal-to-noise ratios, used in Set 2, it was shown that the developed root solver performs very well. On the other hand both MULTROOT, with threshold$= \epsilon_c$, and root() return simple complex conjugate pairs of roots. This shows that the structured polynomial root solver described in this work provides more reliable computations, in the presence of noise, than MULTROOT, and this is due to its robust structured methods that allow lower levels of signal-to-noise ratios to be handled.

**Polynomial Set 3:** The examples in Sets 1 and 2 added noise to the coefficients of the given polynomial such that the componentwise signal-to-noise ratio $\epsilon_c^{-1}$ is constant. The examples in this set allow the $\epsilon_c^{-1}$ to vary between the coefficients, that is, the signal-to-noise ratio of each coefficient varies between $a$ and $b$, where $b/a = 10^3, 10^4$ or $10^5$.

**Example 9.7.** Let the exact polynomial $\hat{f}_6(y)$ be defined by the roots and multiplicities given in the first and second columns of Table 9.7, respectively. The different coefficients of $\hat{f}_6(y)$ were perturbed independently with variable $\epsilon_c$ whose value ranges between $10^{-10}$ and $10^{-5}$.

The results of computing the roots of $f_6(y)$, and their corresponding multiplicities, using the root solver described in this work are given in the third and fourth columns

Table 9.7: The roots and multiplicities of $\hat{f}_6(y)$ for Example 9.7.

| exact root | exact mult. | computed root | computed mult. | relative error |
|---|---|---|---|---|
| 2 | 5 | 2.00002785e+000 | 5 | 1.39246339e-005 |
| 3 | 3 | 2.99970072e+000 | 3 | 9.97587653e-005 |
| 6 | 2 | 5.99968552e+000 | 2 | 5.24133777e-005 |
| 4 | 1 | 4.00133963e+000 | 1 | 3.34906515e-004 |

of Table 9.7, respectively. The fifth column of Table 9.7 shows that the relative error in computing each distinct root is adequate with respect to $10^{-10} \leq \epsilon_c \leq 10^{-5}$. MULTROOT yields unsatisfactory answers for threshold $= 10^{-10}, 10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}$ and $10^{-5}$, because it returned simple complex conjugate pairs of roots and similar results were obtained using roots(). □

**Example 9.8.** Consider the exact polynomial $\hat{f}_7(y)$ which is defined by the roots and multiplicities given in the first and second columns of Table 9.8, respectively. The different coefficients of $\hat{f}_7(y)$ were perturbed by variable $\epsilon_c$ whose value ranges between $10^{-9}$ and $10^{-6}$.

Table 9.8: The roots and multiplicities of $\hat{f}_7(y)$ for Example 9.8.

| exact root | exact mult. | computed root | computed mult. | relative error |
|---|---|---|---|---|
| -7.5947e+00 | 6 | -7.59470511e+00 | 6 | 6.73425542e-07 |
| 6.3371e-01 | 5 | 6.33713762e-01 | 5 | 5.93626159e-06 |
| 1.4923e+00 | 5 | 1.49228952e+00 | 5 | 7.02533502e-06 |
| 5.4862e+00 | 4 | 5.48617804e+00 | 4 | 4.00225089e-06 |
| -3.3076e+00 | 3 | -3.30759693e+00 | 3 | 9.29136428e-07 |
| -3.0670e+00 | 2 | -3.06700249e+00 | 2 | 8.12519187e-07 |
| 4.2244e-01 | 2 | 4.22435986e-01 | 2 | 9.50092906e-06 |
| 2.5090e+00 | 2 | 2.50904664e+00 | 2 | 1.85908575e-05 |

The results of computing the roots of $f_7(y)$, and their corresponding multiplicities, using the root solver described in this work are given in the third and fourth columns of Table 9.8, respectively. The fifth column of Table 9.8 shows the relative error in computing each distinct root. MULTROOT() yielded unsatisfactory answers for threshold $= 10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}$, because it returned simple complex conjugate pairs of roots and similar results were obtained using roots(). □

**Example 9.9.** The procedure described in the previous two examples was repeated for the polynomial whose roots and multiplicities are given in the first and second columns of Table 9.9. Consider the exact polynomial $\hat{f}_8(y)$ which is defined by the roots and multiplicities given in the first and second columns of Table 9.9, respectively. The different coefficients of $\hat{f}_8(y)$ have been perturbed independently with variable randomly by $\varepsilon_c$ whose value ranges between $10^{-10}$ and $10^{-8}$.

Table 9.9: The roots and multiplicities of $\hat{f}_8(y)$ for Example 9.9.

| exact root | exact mult. | computed root | computed mult. | relative error |
|---|---|---|---|---|
| -8.79070000e+00 | 9 | -8.79070047e+00 | 6 | 5.39267513e-08 |
| -1.99840000e+00 | 4 | -1.99839999e+00 | 5 | 2.51676037e-09 |
| 6.63740000e+00 | 4 | 6.63740102e+00 | 5 | 1.53471615e-07 |
| 4.44700000e+00 | 3 | 4.44699986e+00 | 4 | 3.04789679e-08 |
| 9.01830000e+00 | 2 | 9.01829716e+00 | 3 | 3.14759166e-07 |
| -7.31320000e+00 | 1 | -7.31319737e+00 | 2 | 3.59548548e-07 |

The results of computing the roots of $f_8(y)$, and their corresponding multiplicities, using the root solver described in this work are given in the third and fourth columns of Table 9.9, respectively. The fifth column of Table 9.9 shows the relative error in computing each distinct root. The root solver MULTROOT yielded unsatisfactory answers for threshold $= 10^{-10}, 10^{-9}, 10^{-8}$, because it returned simple complex conjugate

pairs of roots and similar results were obtained using roots(). □

It is shown in this section that the developed root solver and MULTROOT work well in the presence of noise. However, MULTROOT requires that the argument threshold$\geq \varepsilon_c$ be satisfied. On the other hand, the developed root solver does not require that the noise level be known. This is due to the fact that the developed root solver uses data-driven methods. In particular, the determination of the degree of an AGCD of two inexact polynomials is done using (6.9) and (6.22), which depend on the given data rather than a fixed threshold value. This property in the developed root solver allows it to handle harder classes of polynomials with a lower signal-to-noise ratio. In such cases, MULTROOT does not always provide satisfactory results even if the exact $\varepsilon_c$ is known and the argument threshold is set equal to $\varepsilon_c$. Furthermore, in practice, different coefficients of a polynomial may have different values of signal-to-noise ratio. When such situations were tested, it was shown that the developed root solver provided better results than MULTROOT, and it is suggested that, with this class of polynomials, it is hard to define a threshold value for MULTROOT.

## 9.3 Summary

In this chapter the final stage of Algorithm 2.3.1 has been discussed. It has been shown that it requires the computation of the solution of several polynomial equations, all of whose roots are simple, distinct and can be computed using classical root solving methods. The method of non-linear least squares has been used to refine the values of these roots under the constraint that their multiplicities are preserved. Experimental results of applying the developed root solver on different polynomial

classes have been presented, in the presence of noise. These results have been compared with the results obtained from MULTROOT which is developed by Zeng [85] and the MATLAB function roots().

While the developed root solver provides excellent results, both MULTROOT and the function roots() perform badly, in the presence of noise. In particular, MULTROOT does not provide good results if the noise level is greater than the default setting of the threshold, and even if the threshold is set at the known signal-to-noise ratio, it does not necessarily preserve the multiplicities of the theoretically exact roots.

# Chapter 10

# Conclusions and future work

## 10.1 Conclusion

The work presented in this thesis has described a polynomial root solver that computes multiple roots of inexact polynomials. Due to the ill-posed nature of this problem, a small perturbation is sufficient to break up the multiple roots into simple roots, and many of the root solving methods in the literature fail to compute the correct values of the multiple roots. The algorithm used in the root solver developed in this work first computes the multiplicity structure of the given polynomial through successive AGCD computations. It then uses two sets of polynomial divisions to break up the given polynomial into several polynomials whose roots are simple and distinct. Finally, a refinement stage is used to improve the accuracy of the results. thereby yielding superior results. Following this procedure in computing the roots has shown significant improvements in the results with respect to the previous work.

The computational implementation of this algorithm involves three main operations:

1. The computation of successive AGCDs.

2. The computation of successive polynomial divisions.

3. The computation of several polynomials whose roots are simple and distinct.

The first operation *AGCD computation* involves two stages

(a) The computation of the degree of the AGCD.

(b) The computation of the coefficients of the AGCD.

Three methods are used to compute the degree of the AGCD. The first and second methods are applicable to any pair of polynomials, whereas the third method is only applicable to a polynomial and its derivative. All three methods use the Sylvester matrix $S(f,g)$ of $f(y)$ and $g(y)$ and its subresultant matrices, but they differ in the criteria used to evaluate the error in a linear algebraic equation.

The first method uses the first principal angle between the space spanned by one column of $S_k(f,g)$ and the space spanned by the remaining columns of $S_k(f,g)$, where $k$ denotes the order of the subresultant matrix. The second method uses the residual of an approximate linear algebraic equation. The third method uses the relation between a polynomial and its derivative and therefore it is only applicable for a polynomial and its derivative. The majority voting principle is then used to determine the degree of the AGCD.

Once the degree of the AGCD is known, its coefficients are computed in the second stage, using the method of non-linear structured total least norm. Two different algorithms are developed for the computation of the coefficients of an AGCD. Both algorithms use the method of SNTLN. In particular, the first method applies the

method of SNTLN to the Sylvester matrix of the inexact polynomials, and the coefficients of the AGCD can either be taken from the last non-zero of the Sylvester matrix after reducing its transpose to upper triangular form, or they can be taken from the null space of the Sylvester matrix. On the other hand, the second method computes the AGCD explicitly without the need for extra computation as it applies the method of SNTLN to the approximate factorisation of two inexact polynomials. The examples in Chapter 7 show that both methods give excellent results for hard classes of polynomials.

The second operation *polynomial division* is an ill-posed computation and thus it is treated with care to provide more accurate results. In particular, the coefficients of the polynomials that are involved in the divisions $q_i(y)/q_{(i+1)}(y)$, are perturbed with structured perturbations, using the method of STLN, such that the perturbed form of $q_{(i+1)}(y)$ is an exact divisor of the perturbed form of $q_i(y)$. A similar procedure is applied to the second set of division $h_i(y)/h_{i+1}(y)$.

In Chapter 9, the first and second operations are combined and applied successively, and this yields several polynomials, each of which only has simple distinct roots. The MATLAB function roots() has been used to solve these polynomial equations. Improved results are obtained when the method of non-linear least squares is used to refine the values of these roots. This refinement process is done under the constraint of the multiplicity structure of the given polynomial, such that the polynomial defined by the refined roots is kept on the same manifold as that of the polynomial formed by the initial roots estimates.

The developed method has been used to compute the multiple roots of hard classes of polynomials and it is shown in Chapter 9 that it gives very good results.

To summarise, in addition to developing a root finding method that computes the multiple roots of inexact polynomials, the work in this thesis shows that:

1. The computation of the multiplicity structure of the given inexact polynomial is the most crucial stage in the computation of its multiple roots.

2. Based on the geometric interpretation described in this thesis, the computation of the multiple roots of inexact polynomial is well conditioned if the multiplicity structure of the polynomial is preserved.

3. Preprocessing the given polynomial, using the preprocessing operations described in this work, has a significant effect in providing more reliable computations.

4. The computation of the optimal values of the scaling parameters $\alpha$ and $\theta$ can be performed by solving a linear programming problem.

5. The numerical rank of the Sylvester matrix can be computed directly from the given data without the need for prior knowledge about the level of the noise.

6. The method of SNTLN can be used to compute an AGCD of inexact polynomials.

7. The method of STLN can be used to impose a constraint on the polynomial division in order to induce a polynomial rather than a rational function as the solution.

8. The method of non-linear least squares is efficient in refining the values of the roots.

## 10.2 Future work and improvements

The work in this research only considers the feasibility of using structured methods for computing multiple roots of an inexact polynomial and it is shown that it provides encouraging results. However, further work is needed to improve its computational efficiency by developing fast algorithms that exploit the structure nature of the developed methods. In particular, the proposed methods used for the computation of the rank of the Sylvester matrix of inexact polynomial require the computation of the SVD for each subresultant matrix, which is expensive computationally. Since two successive subresultant matrices differ only in one column, an update procedure should be used for computational efficiency, and thus the QR decomposition [27, 57] can be used. Moreover, the method that uses the APF for the computation of an AGCD of two polynomials uses two Cauchy matrices. A fast algorithm can be developed to exploit the structure of a matrix that contains two Cauchy matrices.

Curves and surfaces in geometric modelling are represented as polynomials in the Bernstein basis, and thus intersection problems reduce to the solution of one or more polynomial equations. This application requires extending the work presented in this thesis to the Bernstein basis. Another application is blind image deconvolution, in which two noisy images of the same scene are used to obtain an improved image (high signal-to-noise ratio) of the scene. Although this is a bivariate problem, Fourier transforms enable this problem to be reduced to a univariate GCD problem [43, 62], and thus the methods discussed in this thesis are appropriate.

# Bibliography


[1] H. L. Porteous A. C. Baker. *Linear Algebra and Differential Equations.* Ellis Horwood Limited, New York; London, 1990.

[2] Dario A.Bini. Structured matrix-based methods for polynomials $\epsilon$-gcd: Analysis and comparisons. *Adv. Comp. Math.*, 2007.

[3] A. Aliphas, S. Narayan, and A. Peterson. Finding the zeros of linear phase fir frequency sampling digital filters. *IEEE Transaction on Acoustics, Speech and Signal Processing*, 31:729–734, 1983.

[4] Marc Van Barel, Raf Vandebril, and Paul Van Dooren. Implicit double shift QR-algorithm for companion matrices, 2008.

[5] S. Barnett. *Polynomials and Linear Control Systems.* Marcel Dekker, New York, USA, 1983.

[6] Bernhard Beckermann and George Labahn. A fast and numerically stable euclidean-like algorithm for detecting relatively prime numerical polynomials. *J. Symb. Comput.*, 26(6):691–714, 1998.

[7] D. A. Bini, Y. Eidelman, L. Gemignani, and I. Gohberg. Fast QR eigenvalue algorithms for Hessenberg matrices which are rank-one perturbations of unitary

matrices. *SIAM Journal on Matrix Analysis and Applications*, 29(2):566–585, 2007.

[8] D.A. Bini and P. Boito. Structured matrix-based methods for polynomial $\epsilon$-gcd: analysis and comparisons. In *Proceedings of the 2007 international symposium on Symbolic and algebraic computation*, pages 9–16. ACM, 2007.

[9] D.A. Bini, P. Boito, Y. Eidelman, L. Gemignani, and I. Gohberg. A fast implicit QR eigenvalue algorithm for companion matrices. *Linear Algebra and its Applications*, 432(8):2006 – 2031, 2010. Special issue devoted to the 15th ILAS Conference at Cancun, Mexico, June 16-20, 2008.

[10] Christian H. Bischof. Incremental condition estimation. *SIAM J. Matrix Anal. Appl.*, 11(2):312–322, 1990.

[11] Å. Björck. *Numerical Methods for Least Squares Problems*. SIAM, Philadelphia, USA, 1996.

[12] A. Bjork and Gene H. Golub. Numerical methods for computing angles between linear subspaces. *Mathematics of Computation*, 27:579–579.

[13] W. S. Brown. On euclid's algorithm and the computation of polynomial greatest common divisors. *J. ACM*, 18:478–504, October 1971.

[14] Paulina Chin, Robert M. Corless, and George F. Corliss. Optimization strategies for the approximate GCD problem. In *ISSAC '98: Proceedings of the 1998 international symposium on Symbolic and algebraic computation*, pages 228–235, New York, NY, USA, 1998. ACM.

[15] George E. Collins. Subresultants and reduced polynomial remainder sequences. *J. ACM*, 14:128–142, January 1967.

[16] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt. The singular value decomposition for polynomial systems. In *Proc. Int. Symp. Symbolic and Algebraic Computation*, pages 195–207. ACM Press, New York, 1995.

[17] R. M. Corless, S. M. Watt, and L. Zhi. $QR$ factoring to compute the GCD of univariate approximate polynomials. *IEEE Trans. Signal Processing*, 52(12):3394–3402, 2004.

[18] D.A. Cox, J. Little, and D. O'Shea. Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra, 3/e (Undergraduate Texts in Mathematics). 2007.

[19] Germund Dalhquist, Åke Björck, and Ned Anderson. *Numerical Methods.* Prentice-Hall Series in Automatic Computation. 1974.

[20] J. Demmel. On condition numbers and the distance to the nearest ill-posed problem. *Numerische Mathematik*, 51:251–289, 1987.

[21] D.K Dunaway. *A Composite Algorithm for Finding Zeros of Real Polynomials.* PhD thesis, Southern Methodist University, Texas, USA, 1972.

[22] A. Edelman and H. Murakami. Polynomial roots from companion matrix eigenvalues. *Mathematics of Computation*, 64(210):763–776, 1995.

[23] I. Emiris, A. Galligo, and H. Lombardi. Numerical univariate polynomial GCD. In J. Renegar, M. Schub, and S. Smale, editors, *The Mathematics of Numerical*

*Analysis. Volume 32 of Lecture Notes in Applied Mathematics*, pages 323–343. AMS, 1996.

[24] I. Emiris, A. Galligo, and H. Lombardi. Certified approximate univariate GCDs. *J. Pure and Applied Algebra*, 117,118:229–251, 1997.

[25] C. F. Gerald and P. O. Wheatley. *Applied Numerical Analysis.* Addison-Wesley, USA, 1994.

[26] S. Ghaderpanah and S. Klasa. Polynomial scaling. *SIAM Journal on Numerical Analysis*, 27(1):117–135, 1990.

[27] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, San Diego, USA, 1995.

[28] G. H. Golub and C. F. Van Loan. *Matrix Computations.* John Hopkins University Press, Baltimore, USA, 1996.

[29] Richard W Hamming. *Numerical methods for scientists and engineers (2nd ed.).* Dover Publications, Inc., New York, NY, USA, 1986.

[30] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, USA, 2002.

[31] D. G. Hough. *Explaining and Ameliorating the Ill Condition of Zeros of Polynomials.* PhD thesis, Department of Computer Science, University of California, Berkeley, USA, 1977.

[32] V. Hribernig. *Sensitivity of Algebraic Algorithms.* PhD thesis, Technical University of Vienna, 1994.

[33] V. Hribernig and H. J. Stetter. Detection and validation of clusters of polynomial zeros. *Journal of Symbolic Computation*, 24:667–681, 1997.

[34] W. Kahan. Conserving confluence curbs ill-condition. Technical report, Department of Computer Science, University of California, Berkeley, USA, 1972.

[35] W. Kahan. The improbability of probabilistic error analyses for numerical computations. `http://www.cs.berkeley.edu/~wkahan/improber.ps`, 1996.

[36] E. Kaltofen, Z. Yang, and L. Zhi. Structured low rank approximation of a Sylvester matrix. *Symbolic-numeric computation*, pages 69–83, 2007.

[37] N. Karmarkar and Y. N. Lakshman. Approximate polynomial greatest common divsior and nearest singular polynomials. In *Proc. Int. Symp. Symbolic and Algebraic Computation*, pages 35–39. ACM Press, New York, 1996.

[38] N. K. Karmarkar. On approximate GCDs of univariate polynomials. *J. Symb. Comput.*, 26(6):653–666, 1998.

[39] S.R. Khare, H.K. Pillai, and M.N. Belur. Computing approximate GCD of univariate polynomials. In *Control Automation (MED), 2010 18th Mediterranean Conference on*, pages 437 –441, 2010.

[40] D. E. Knuth. *The Art of Computer Programming, Vol 2.* Addison-Wesley, Reading, USA, 1969.

[41] Bingyu Li, Zhuojun Liu, and Lihong Zhi. A fast algorithm for solving the sylvester structured total least squares problem. *Signal Process.*, 87(10):2313–2319, 2007.

[42] Zijia Li, Zhengfeng Yang, and Lihong Zhi. Blind image deconvolution via fast approximate GCD. In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ISSAC '10, pages 155–162, New York, NY, USA, 2010. ACM.

[43] Ben Liang and S.U. Pillai. Blind image deconvolution using a robust 2-d GCD approach. In *Circuits and Systems, 1997. ISCAS '97., Proceedings of 1997 IEEE International Symposium on*, volume 2, pages 1185 –1188 vol.2, June 1997.

[44] D. Manocha and J. Demmel. Algorithms for intersecting parametric and algebraic curves II: Multiple intersections. *Graphical Models and Image Processing*, 57(2):81–100, 1995.

[45] C. Moler. Cleve's corner: roots of polynomials. *The MathWorks Newsletter*, 5(1):8–9, 1991.

[46] Arnold Neumaier. Enclosing clusters of zeros of polynomials. *J. Comput. Appl. Math.*, 156(2):389–401, 2003.

[47] B. Nobel and J. W. Daniel. *Applied Linear Algebra.* Prentice Hall, Englewood Cliffs, New Jersey, USA, 1988.

[48] M. T. Noda and T. Sasaki. Approximate GCD and its application to ill-conditioned linear algebraic equations. *Journal of Computational and Applied Mathematics*, 38:335–351, 1991.

[49] C.C. Paige and M. Wei. History and generality of the cs decomposition. *Linear Algebra and its Applications*, 208-209:303 – 326, 1994.

[50] V. Y. Pan. Solving a polynomial equation: Some history and recent progress. *SIAM Review*, 39(2):187–220, 1997.

[51] V. Y. Pan. Computation of approximate polynomial GCDs and an extension. *Information and Computation*, 167:71–85, 2001.

[52] Victor Y. Pan. Optimal (up to polylog factors) sequential and parallel algorithms for approximating complex polynomial zeros. In *STOC '95: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 741–750, New York, NY, USA, 1995. ACM.

[53] Victor Y. Pan. Approximate polynomial GCDs, padé approximation, polynomial zeros and bipartite graphs. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 68–77, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[54] V.Y. Pan. Optimal and nearly optimal algorithms for approximating polynomial zeros. *Computers & Mathematics with Applications*, 31(12):97–138, 1996.

[55] Nicholas M. Patrikalakis and Takashi Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing.* Springer, 2009.

[56] D.H. Pham and J.H. Manton. A subspace algorithm for guard interval based channel identification and source recovery requiring just two received blocks. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, volume 4, pages IV – 317–20 vol.4, 2003.

[57] William H. Press, Saul A. Teukolsky, and Brian P. Vetterling, William T.and Flannery. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge University Press, New York, NY, USA, 2nd edition, 1992.

[58] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical recipes in C (2nd ed.): the art of scientific computing.* Cambridge University Press, New York, NY, USA, 1992.

[59] W. Qiu and Y. Hua. A gcd method for blind channel identification,. *Digital Signal Processing*, 7(3):199 – 205, 1997.

[60] J. Ben Rosen, H. Park, and J. Glick. Total least norm formulation and solution for structured problems. *SIAM J. Mat. Anal. Appl.*, 17(1):110–128, 1996.

[61] J. Ben Rosen, H. Park, and J. Glick. Structured total least norm for nonlinear problems. *SIAM J. Mat. Anal. Appl.*, 20(1):14–30, 1998.

[62] Ben Liang S. Unnikrishna Pillai. Blind image deconvolution using a robust GCD approach. *IEEE Transaction on image processing*, 8(2), 1999.

[63] M. Sanuki and T. Sasaki. Computing approximate GCDs in ill-conditioned cases. In *Proceedings of the 2007 international workshop on Symbolic-numeric computation*, pages 170–179. ACM, 2007.

[64] T. Sasaki and F. Kako. An algebraic method for separating close-root clusters and the minimum root separation. *Symbolic-Numeric Computation*, pages 149–166, 2007.

[65] Tateaki Sasaki and Akira Terui. Computing clustered close-roots of univariate polynomials. In *SNC '09: Proceedings of the 2009 conference on Symbolic numeric computation*, pages 177–184, New York, NY, USA, 2009. ACM.

[66] SS Sastry. *Introductory methods of numerical analysis.* PHI Learning Pvt. Ltd., 2006.

[67] C. E. Schmidt and L. R. Rabiner. A study of techniques for finding the zeros of linear phase FIR digital filters. *IEEE Trans. Acoustics, Speech and Signal Processing*, 27:96–98, 1977.

[68] A. Schönhage. Quasi-gcd computations. *Journal of Complexity*, 1:118–137, 1985.

[69] C.G. Small and J. Wang. *Numerical methods for nonlinear estimating equations.* Oxford statistical science series. Oxford University Press, 2003.

[70] K. Steiglitz and B. Dickinson. Phase unwrapping by factorization. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 30(6):984–991, 1982.

[71] Petre Stoica and Torsten Söderström. Common factor detection and estimation. *Automatica*, 33(5):985–989, 1997.

[72] Borching Su. Blind channel estimation using redundant precoding : new algorithms, analysis, and theory. 2008.

[73] Nai-Kuan Tsao and Franklin F. Kuo. On machine precision, computation error and condition number in solving linear algebraic systems.

[74] J. V. Uspensky. *Theory of Equations.* McGraw-Hill, New York, USA, 1948.

[75] D. S. Watkins. *Fundamentals of Matrix Computations.* John Wiley and Sons, New York, USA, 1991.

[76] J. Wilkinson. *Rounding Errors In Algebraic Processes.* Prentice-Hall, Englewood Cliffs,N.J., USA, 1963.

[77] J. R. Winkler. Condition numbers of a nearly singular simple root of a polynomial. *Applied Numerical Mathematics*, 38:275–285, 2001.

[78] Joab R. Winkler and John D. Allan. Structured total least norm and approximate gcds of inexact polynomials. *J. Comput. Appl. Math.*, 215(1):1–13, 2008.

[79] Joab R. Winkler and Madina Hasan. A non-linear structure preserving matrix method for the low rank approximation of the sylvester resultant matrix. *Journal of Computational and Applied Mathematics*, 234:3226–3242, 2010.

[80] Joab R. Winkler and Xin Lao. The calculation of the degree of an approximate greatest common divisor of two polynomials. *Journal of Computational and Applied Mathematics*, 2010.

[81] Xinyuan Wu. On zeros of polynomial and vector solutions of associated polynomial system from vieta theorem. *Applied Numerical Mathematics*, 44(3):415 – 423, 2003.

[82] C. J. Zarowski. The MDL criterion for rank determination via effective singular values. *IEEE Trans. Signal Processing*, 46(6):1741–1744, 1998.

[83] C. J. Zarowski, X. Ma, and F. W. Fairman. $QR$-factorization method for computing the greatest common divisor of polynomials with inexact coefficients. *IEEE Trans. Signal Processing*, 48(11):3042–3051, 2000.

[84] Z. Zeng. The approximate GCD of inexact polynomials. Part 1: A univariate algorithm, 2004. Preprint.

[85] Z. Zeng. Computing multiple roots of inexact polynomials. *Mathematics of Computation*, 74:869–903, 2005.

[86] Zhonggang Zeng. Algorithm 835: Multroot—a matlab package for computing polynomial roots and multiplicities. *ACM Transactions on Mathematical Software*, 30:218–236, 2004.

[87] L. Zhi. Displacement structure in computing approximate GCD of univariate polynomials. In *Proc. Sixth Asian Symposium on Computer Mathematics (ASCM 2003*, pages 288–298, 2003.