STRUCTURED MATRIX METHODS FOR A POLYNOMIAL ROOT SOLVER USING APPROXIMATE GREATEST COMMON DIVISOR COMPUTATIONS AND APPROXIMATE POLYNOMIAL FACTORISATIONS

by

XINYUAN LAO

A thesis submitted to the

Computer Science

in conformity with the requirements for

the degree of PhD

Sheffield University

England

June 2011

Abstract

LAO, XINYUAN. STRUCTURED MATRIX METHODS FOR A POLYNOMIAL ROOT SOLVER USING APPROXIMATE GREATEST COMMON DIVISOR COMPUTATIONS AND APPROXIMATE POLYNOMIAL FACTORISATIONS

This thesis discusses the use of structure preserving matrix methods for the numerical approximation of all the zeros of a univariate polynomial in the presence of noise. In particular, a robust polynomial root solver is developed for the calculation of the multiple roots and their multiplicities, such that the knowledge of the noise level is not required. This designed root solver involves repeated approximate greatest common divisor computations and polynomial divisions, both of which are ill-posed computations. A detailed description of the implementation of this root solver is presented as the main work of this thesis. Moreover, the root solver, implemented in MATLAB using 32-bit floating point arithmetic, can be used to solve non-trivial polynomials with a great degree of accuracy in numerical examples.

Acknowledgments

I am sincerely and heartily grateful to my supervisor, Joab Winkler, whose encouragement, supervision and support from the initial to the final level enabled me to complete the project. I am sure that it would have not been possible without his patience, guidance and help.

I would like to thank my parents who have been a source of strength during this period. Special thanks to Tangyun for his personal support and great patience at all times.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of the project.

Abbreviations and notation

GCD	...	greatest common divisor
LP	...	linear programming
LSE	...	least squares with equality
ML	...	maximum likelihood
NLLS	...	non-linear least squares
STLN	...	structured total least norm
SNTLN	...	structured non-linear total least norm
$S(f, g)$	...	Sylvester resultant matrix for the polynomials $f(x)$ and $g(x)$
$S_k(f, g)$	...	Sylvester subresultant matrix of order k for the polynomials $f(x)$ and $g(x)$
$B(f, g)$	...	Bézout resultant matrix for the polynomials $f(x)$ and $g(x)$
$f^{(1)}(x)$	...	first derivative of the polynomial $f(x)$
$\mathbf{f}$	...	vector of coefficients of the polynomial $f(x)$
$\hat{f}(x)$	...	exact form of the polynomial $f(x)$
$\tilde{f}(y)$	...	preprocessed form of the polynomial $f(x)$
α	...	a scale factor
α_{o}	...	the optimal value of α

θ	...	a scale factor
θ_o	...	the optimal value of θ
ε_c^{-1}	...	componentwise signal-to-noise ratio
$\eta_c(\tilde{x}_0)$	...	componentwise backward error of the approximate root $\tilde{x}_0$
$\kappa_c(x_0)$	...	componentwise condition number of the root x_0
$\rho(x_0)$	...	condition number of the root x_0 that preserves its multiplicity
$\log x$	...	$\log_{10} x$
$\lVert\cdot\rVert$	...	$\lVert\cdot\rVert_2$

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Historical review

Finding the solutions of a polynomial equation is among the oldest problems in mathematics. This problem was known to the Sumerians (third millennium B.C.), and it has deeply influenced the development of mathematics throughout the centuries and is of great practical importance in science and engineering presently [19, 49, 59]. In particular, solving a polynomial equation continues to be a major role in the highly important area of computing called computer algebra, especially for polynomials of high degree, in which case many computational tools from linear algebra, linear programming and fast Fourier transform (FFT) may require a solution of a polynomial equation [52]. Furthermore, many applications in computer algebra, robotics, computer graphics, computer vision, geometric and solid modeling and molecular modeling require a solution to a set of polynomial equations due to geometric operations [46]. However, the current viewpoint is that there are `no good, general solvers` for solving systems of more than one polynomial equation, as highlighted in [55].

Starting with the Sumerians and Babylonians, the study of univariate polynomial zero finding focused on small degree equations for specific coefficients. The solution formula for quadratic (second degree) polynomials has been known to the Babylonians (about 2000 B.C.) and the Egyptians (found in the Rhind or Ahmes papyrus of second millennium B.C.), and those for cubic (third degree) and quartic (fourth degree) polynomials were found successfully in the 16th century by Scipione del Ferro, Nicolo Tartaglia, Ludovico Ferrari and Geronimo Cardano. In 1824, the mathematician Niels Henrik Abel proved the striking result that there does not exist a formula for polynomials of degree 5 or those of higher degree. The absence of a solution formula requires the development of effective numerical methods for iteratively factoring polynomials of degree greater than 4. More details in historical review for solving a polynomial equation have been discussed by Pan [52].

There are some outstanding algorithms that have been proposed and used in the 20th century. Bairstow's method [21] is only valid for polynomials with real coefficients, and impractically slow in finding a double zero, as is Müller's method [21] which is based on approximating the polynomial in the neighborhood of the root by a quadratic polynomial when the order of multiplicity is three [19]. Newton's method [45] requires that the initial estimate is sufficiently near the exact root for convergence, and runs into trouble with multiple roots or closely spaced roots [59]. Laguerre's method [17, 26] is almost always guaranteed to converge to a root of the polynomial for all initial estimates and performs better for multiple roots. The computation is however very expensive as a general purpose polynomial root finder. The Jenkins - Traub algorithm [32, 33] involves three stages and is only valid for polynomials with real coefficients, but it is fast and globally convergent for all distributions of zeros.

These methods yield satisfactory results on a polynomial that has moderate degree and simple and well-distributed roots, with an assumption that a good starting point is used in the iterative scheme. Moreover, the quality of the results calculated by standard numerical methods deteriorates as the degree of the polynomial increases, the multiplicity of one or more of its roots increases, or the proximity of the roots decreases. According to the view of Dunaway and Turlington [15], these methods can fail when they encounter clustered or multiple roots and other types of ill-conditioned polynomials. Also, several principles can be used in testing polynomial zero finding programs, namely, program robustness, convergence difficulties, specific weakness of algorithms and program performance by statistical testing [34].

In recent years, some new methods were therefore developed for the numerical solution of polynomial equations, that is, determining all the zeros of a polynomial can be solved by factorization [11], matrix pencils [35] and structure matrix-based methods [20, 72].

1.2 Examples of errors

This section contains two examples that illustrate the problems of finding all the zeros of a polynomial that has multiple roots. Example 1.1 shows that roundoff errors can cause a significant deterioration in the computed roots, and Example 1.2 shows the effect of a perturbation in a coefficient of a polynomial of high degree.

Since the `roots` function in MATLAB is used in Examples 1.1 and 1.2 to compute the roots of a polynomial, it is important to explain this function in detail. The `roots` function uses the QR algorithm, which is a numerically stable method [28] to compute the eigenvalues of the companion matrix.

The companion matrix of the polynomial

$$f(\lambda) = \lambda^m + a_1\lambda^{m-1} + \cdots + a_{m-1}\lambda + a_m$$

is defined as

$$C = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ -a_m & -a_{m-1} & -a_{m-2} & \cdots & -a_1 \end{bmatrix}$$

in which the first superdiagonal consists entirely of ones and all other elements above the last row are zeros. The characteristic equation of A is equal to $f(\lambda)$ [30], pages 146 – 147,

$$f(\lambda) = \det(C - \lambda I),$$

that is, the eigenvalues of C are the roots of the polynomial $f(\lambda)$.

In numerical linear algebra, the QR algorithm is a procedure to calculate the eigenvalues and eigenvectors of a matrix. The basic idea is to perform a QR decomposition, writing the matrix as a product of an orthogonal matrix and an upper triangular matrix, multiply the factors in reverse order, and iterate.

Generally, suppose that A is the given matrix whose eigenvalues should be computed, and let $A_0 = A$. At the kth step (starting with $k = 0$), compute the QR decomposition of A_k, that is, $A_k = Q_k R_k$ where Q_k is a orthogonal matrix and R_k is an upper triangular matrix. Then form the matrix $A_{k+1} = R_k Q_k$ such that

$$A_{k+1} = R_k Q_k = Q_k^T Q_k R_k Q_k = Q_k^T A_k Q_k = Q_k^{-1} A_k Q_k.$$

All the matrices A_k are similar and thus they have the same eigenvalues. The algorithm is numerically stable because it proceeds by orthogonal similarity transformations [28]. More detail about the QR algorithm can be found in [24], pages 352 – 361.

It can be concluded, therefore, that the QR algorithm can be used to compute the eigenvalues of the companion matrix of the polynomial $f(\lambda)$ in order to obtain the roots of $f(\lambda)$. Hence the QR algorithm is used by the `roots` function in MATLAB to compute the roots of a polynomial.

Example 1.1. Consider the polynomial $(x-1)^{12}$ whose root is $x = 1$ with multiplicity 12. The `roots` function in MATLAB returns the roots

$$\begin{array}{llll}
1.0947 & 1.0804 + 0.0488i & 1.0804 - 0.0488i & 1.0433 + 0.0818i \\
1.0433 - 0.0818i & 0.9963 + 0.0905i & 0.9963 - 0.0905i & 0.9530 + 0.0753i \\
0.9530 - 0.0753i & 0.9233 + 0.0423i & 0.9233 - 0.0423i & 0.9128
\end{array}$$

which are shown in Figure 1.1.

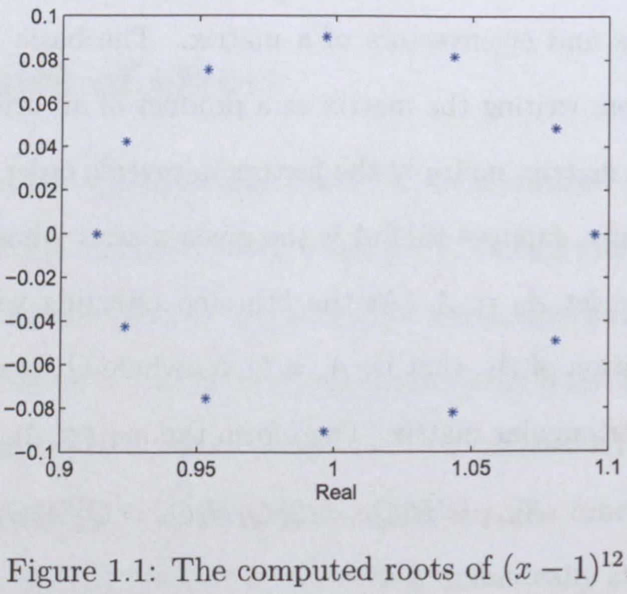


Figure 1.1: The computed roots of $(x-1)^{12}$.

It is clear that the multiple root has split up into 12 distinct roots because of roundoff errors. Roundoff errors due to floating point arithmetic of $O(10^{-16})$ are sufficient to cause a relative error in the solution of about 9×10^{-2}, and thus it is unsatisfactory for the computation of multiple root. □

Example 1.2. Consider the effect of perturbing the constant coefficient of the polynomial $(x-1)^{12}$ by $-\epsilon$ where $|\epsilon| \ll 1$. The roots of the perturbed polynomial are the solutions of $(x-1)^{12} - \epsilon = 0$, that is,

$$x = 1 + \epsilon^{\frac{1}{12}}. \tag{1.1}$$

Euler's formula states that, for any real number θ,

$$e^{i\theta} = \cos\theta + i\sin\theta, \qquad \theta \in \mathbb{R}.$$

If $\theta = 2\pi k, k \in \mathbb{Z}$, it follows that

$$e^{i2\pi k} = \cos 2\pi k + i \sin 2\pi k = 1. \tag{1.2}$$

If $\epsilon = 2^{-12}$, then from (1.1) and (1.2) the solution is

$$\begin{aligned} x &= 1 + (2^{-12}e^{i2\pi k})^{\frac{1}{12}} \\ &= 1 + \frac{1}{2}e^{\frac{i2\pi k}{12}}, \qquad k = 0, \ldots, 11, \\ &= 1 + \frac{1}{2}\left(\cos\frac{\pi k}{6} + i\sin\frac{\pi k}{6}\right), \qquad k = 0, \ldots, 11. \end{aligned}$$

The roots are shown in Figure 1.2, and it is seen that they lie on a circle in the complex plane, with centre at (1,0) and radius 1/2, that is, a perturbation as small as 2^{-12} to the constant coefficient can result in a relative error of 50% in the solution. Hence, an error in one coefficient is small enough to cause a huge error in the computation of a multiple root. □

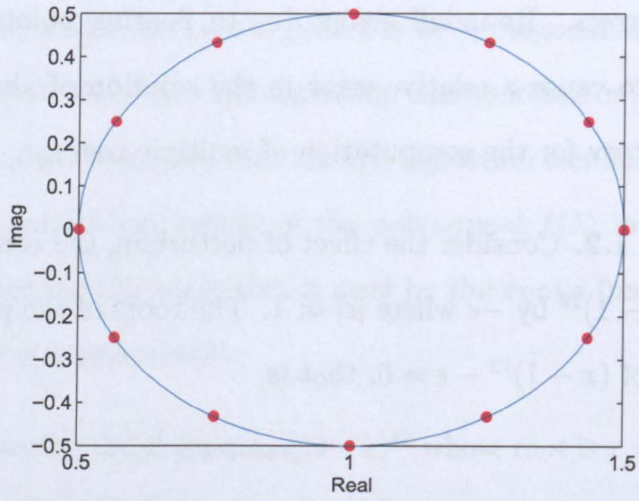


Figure 1.2: Perturbation region, in the complex plane, of the roots of $(x-1)^{12}$ when the constant term is perturbed by 2^{-12}.

These two simple examples show that roundoff errors due to floating point arithmetic and errors in polynomial coefficients, that are present in most practical examples, are sufficient to cause an incorrect and unacceptable solution.

According to the remarks of Goedecker [23], this kind of polynomial $x^m - 1$ in these examples is particularly difficult for the QR algorithm, as it is applied to the `roots` function in MATLAB, although the QR algorithm has considerable advantages over other standard algorithms such as the Jenkins-Traub algorithm and a modified version of Laguerre's algorithm to find the zeros of a polynomial in numerical tests. Goedecker notes on page 1062 that:

"None of the methods gives acceptable results for polynomials of degree higher than 50,"

and he notes on page 1063 that:

"If roots of high multiplicity exist, any ... method has to be used with caution."

Moreover, Karcanias and Mitrouli [38] point out that the uncertainty about the true values of the input data and roundoff errors makes the zero-finding of a polynomial a very difficult task, especially for polynomials of high degree.

1.3 Thesis contribution

Examples 1.1 and 1.2 show that problems arise when it is desired to compute multiple roots of a polynomial, and this leads to the aim of this thesis:

To establish the feasibility of structured matrix methods for a polynomial root solver that can compute multiple roots of a polynomial, particularly for 'difficult polynomials' in presence of noise. It is desirable that this root solver not require an estimate of the noise level, and that all parameters and thresholds be calculated from the data, that is, the coefficients of the given polynomial.

A polynomial root solver, based on a method developed by Gauss and described in Uspensky [62], has been therefore implemented computationally. It is noted that this root solver has not only been implemented for robustness in the presence of noise, but also been developed, with a MATLAB package implementation, in order to overcome the problems in Examples 1.1 and 1.2, and hence solve non-trivial polynomials (high degree, many multiple roots) with a great degree of accuracy in practical application.

This polynomial root solver developed in this thesis has the following property:

The multiplicities of the theoretically exact roots are preserved, even though an inexact (noisy) form of the polynomial is given.

The following mathematical methods are used in the development of the root solver that is described in this thesis:

- linear programming,
- linear and non-linear structure preserving matrix methods,
- non-linear least squares.

Since the aim of this thesis is to establish the feasibility of structured matrix methods for solving a polynomial with multiple roots, little attention has been given to computational test and complexity. Section 6.4, however, considers how the algorithm can be made more efficiently.

The success of this designed root solver is shown in the following examples through several inexact polynomials whose coefficients are perturbed by noise, such that the componentwise signal-to-noise ratio ε_c^{-1} is 10^8. Also the results are compared with the solutions returned by the **roots** function in MATLAB, and more details are shown in Chapter 9.

Exact Multiplicity	Exact Root	Computed Multiplicity	Computed Root	Root Error
1	7.0453	1	7.0453e+000	9.0672e-008
2	0.1127	2	1.1270e-001	1.5210e-009
3	2.7132	3	2.7132e+000	2.1410e-009
4	9.0179	4	9.0179e+000	3.6123e-008
5	-1.1207	5	-1.1207e+000	3.5537e-009
6	-8.7996	6	-8.7996e+000	1.0287e-008

Table 1.1: The computed roots of an inexact polynomial for Example 1.3 using the designed root solver, with $\varepsilon_c = 10^{-8}$.

Example 1.3. The 1st and 2nd columns of Table 1.1 define the exact polynomial that is perturbed by noise, such that the componentwise signal-to-noise ratio ε_c^{-1} is 10^8. The 3rd and 4th columns show the results from the root solver described in this thesis, and the 5th column shows the relative errors in the computed roots.

It is seen that this designed root solver can retain the multiplicities of the roots and the relative errors in the computed roots are approximately equal to the noise level. The **roots** function in MATLAB returns the roots shown in Figure 1.3. It is clear that the multiple roots split up into a cluster of simple roots because of roundoff errors due to floating point arithmetic and errors in polynomial coefficients. □

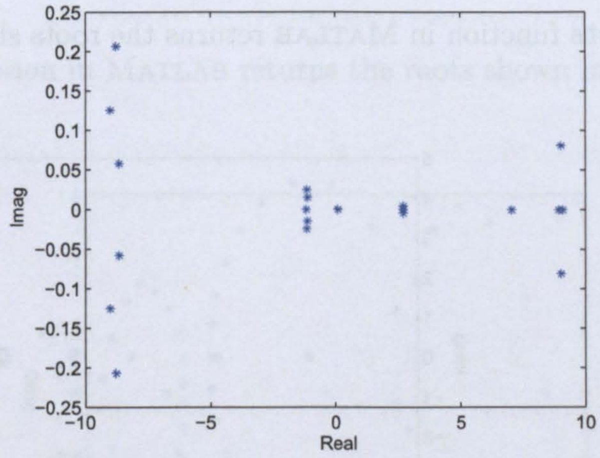


Figure 1.3: The roots of the polynomial in Example 1.3, computed by MATLAB.

The experiment is repeated in Examples 1.4−1.8, with different given polynomials, and these results obtained from Examples 1.4 − 1.8 are similar to Example 1.3. Since the explanation for Tables 1.2 − 1.6 and the analysis for Figures 1.4 − 1.8 are similar to Table 1.1 and Figure 1.3, respectively, for simplicity, Examples 1.4 − 1.8 show only the results that are obtained by the designed root solver and the **roots** function.

Example 1.4. The designed root solver is used to compute the roots of a perturbed polynomial as shown in Table 1.2.

Exact Multiplicity	Exact Root	Computed Multiplicity	Computed Root	Root Error
4	-0.67547	4	-6.7547e-001	1.8925e-009
6	5.7335	6	5.7335e+000	8.2971e-009
7	2.1747	7	2.1747e+000	6.6402e-010
10	-9.5568	10	-9.5568e+000	3.6919e-008
11	-6.5553	11	-6.5553e+000	3.0001e-008

Table 1.2: The computed roots of an inexact polynomial for Example 1.4 using the designed root solver, with $\varepsilon_c = 10^{-8}$.

The **roots** function in MATLAB returns the roots shown in Figure 1.4. □

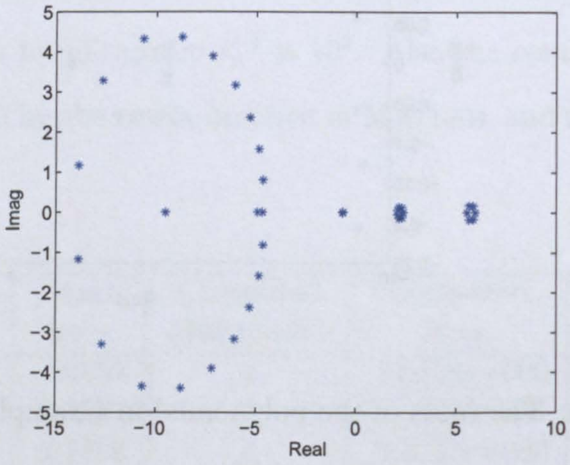


Figure 1.4: The roots of the polynomial in Example 1.4, computed by MATLAB.

Example 1.5. The designed root solver is used to compute the roots of a perturbed polynomial as shown in Table 1.3.

Exact Multiplicity	Exact Root	Computed Multiplicity	Computed Root	Root Error
2	-3.4624	2	-3.4624e+000	5.1724e-006
2	2.6891	2	2.6891e+000	8.2232e-008
2	8.4689	2	8.4689e+000	3.0297e-006
8	-2.5214	8	-2.5214e+000	1.5185e-006
9	-1.6262	9	-1.6262e+000	3.4812e-007
11	6.1616	11	6.1616e+000	4.4626e-007

Table 1.3: The computed roots of an inexact polynomial for Example 1.5 using the designed root solver, with $\varepsilon_c = 10^{-8}$.

The **roots** function in MATLAB returns the roots shown in Figure 1.5. □

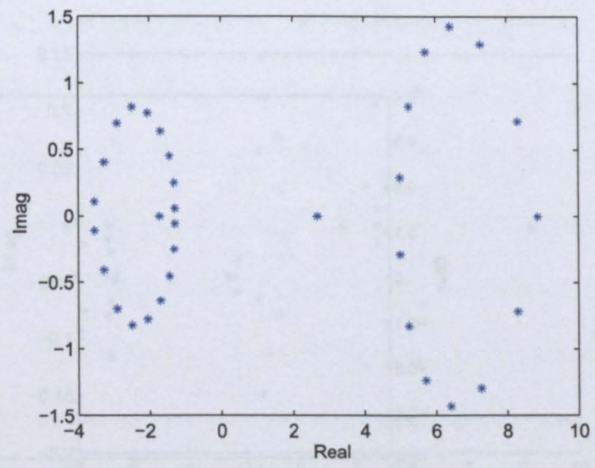


Figure 1.5: The roots of the polynomial in Example 1.5, computed by MATLAB.

Example 1.6. The designed root solver is used to compute the roots of a perturbed polynomial, even though some roots are closely spaced, as shown in Table 1.4.

Exact Multiplicity	Exact Root	Computed Multiplicity	Computed Root	Root Error
2	-3.0670	2	-3.0670e+000	2.8952e-007
2	0.42244	2	4.2244e-001	1.3346e-007
2	2.5090	2	2.5090e+000	7.3839e-007
3	-3.3076	3	-3.3076e+000	2.5817e-007
4	5.4862	4	5.4862e+000	2.1900e-007
5	0.63371	5	6.3371e-001	1.1110e-007
5	1.4923	5	1.4923e+000	2.1946e-007
6	-7.5947	6	-7.5947e+000	6.5487e-008

Table 1.4: The computed roots of an inexact polynomial for Example 1.6 using the designed root solver, with $\varepsilon_c = 10^{-8}$.

The **roots** function in MATLAB returns the roots shown in Figure 1.6. □

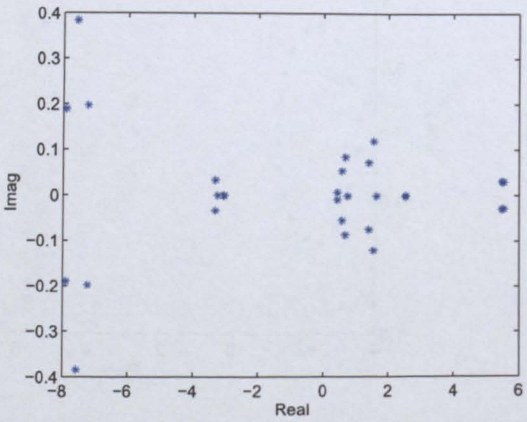


Figure 1.6: The roots of the polynomial in Example 1.6, computed by MATLAB.

Example 1.7. The designed root solver is used to compute the roots of a perturbed polynomial as shown in Table 1.5.

Exact Multiplicity	Exact Root	Computed Multiplicity	Computed Root	Root Error
2	8.3467	2	8.3467e+000	6.6322e-007
3	1.5548	3	1.5548e+000	2.0004e-008
3	2.7865	3	2.7865e+000	2.3510e-007
5	-6.7685	5	-6.7685e+000	6.7958e-008
5	4.3127	5	4.3127e+000	3.4052e-007
6	-1.3340	6	-1.3340e+000	2.7139e-008
6	-0.77536	6	-7.7536e-001	1.7967e-008

Table 1.5: The computed roots of an inexact polynomial for Example 1.7 using the designed root solver, with $\varepsilon_c = 10^{-8}$.

The **roots** function in MATLAB returns the roots shown in Figure 1.7. □

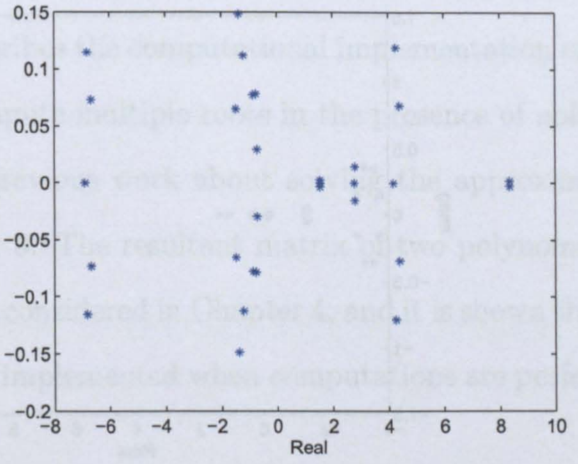


Figure 1.7: The roots of the polynomial in Example 1.7, computed by MATLAB.

Example 1.8. The designed root solver is used to compute the roots of a perturbed polynomial, even though some roots are closely spaced, as shown in Table 1.6.

Exact Multiplicity	Exact Root	Computed Multiplicity	Computed Root	Root Error
2	1.0708	2	1.0708e+000	1.9686e-008
2	1.4168	2	1.4168e+000	1.9956e-008
5	-1.4000	5	-1.4000e+000	2.9568e-009
5	0.30917	5	3.0917e-001	1.3845e-009
7	-0.16387	7	-1.6387e-001	1.3390e-009
8	-3.3864	8	-3.3864e+000	3.6839e-009
9	9.9370	9	9.9370e+000	2.0310e-009

Table 1.6: The computed roots of an inexact polynomial for Example 1.8 using the designed root solver, with $\varepsilon_c = 10^{-8}$.

The **roots** function in MATLAB returns the roots shown in Figure 1.8. □

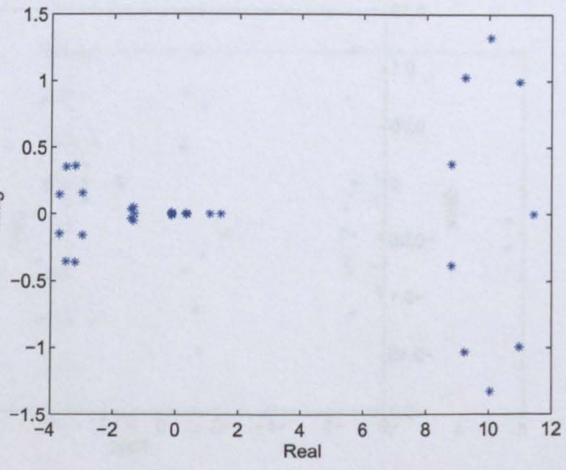


Figure 1.8: The roots of the polynomial in Example 1.8, computed by MATLAB.

Examples 1.3 – 1.8 have therefore demonstrated the success of the designed root solver because the multiplicities of the roots are preserved in the presence of noise, and the output relative errors in the computed roots are approximately equal to the input relative errors.

The layout of the thesis is now detailed.

1.4 Thesis layout

The problem arises when multiple roots of a polynomial are determined in the presence of errors, including roundoff errors, due to the ill-conditioned nature of the problem. The concept of ill-conditioning is therefore introduced in Chapter 2, along with a geometric interpretation of ill-conditioning. The results in Examples 1.3 – 1.8 are obtained by implementing a polynomial root solver that requires several greatest common divisor (GCD) computations and polynomial division operations [62]. The rest of the thesis describes the computational implementation of this root solver, such that it is able to compute multiple roots in the presence of noise.

An overview of previous work about solving the approximate GCD problem is presented in Chapter 3. The resultant matrix of two polynomials, which is required for this root solver, is considered in Chapter 4, and it is shown that some preprocessing operations should be implemented when computations are performed on the resultant matrix.

Chapter 5 describes three methods for the calculation of the degree of an approximate GCD of an inexact polynomial pair without prior knowledge of the noise level in the data. It is extended in Chapter 6 by three other methods, in which case the last method is appropriate for the calculation of the degree of an approximate GCD of an inexact polynomial and its derivative.

Chapter 7 presents the method of structured non-linear total least norm (SNTLN) for the calculation of the coefficients of an approximate GCD. A linear structure preserving matrix method for solving the polynomial division problem is then discussed in Chapter 8 along with the calculation of the multiple roots of a polynomial by the method of non-linear least squares.

The success of the designed root solver to find all zeros of an inexact polynomial is shown in Chapter 9, and a summary of the results and methods are detailed in Chapter 10 followed by possible future extensions to the work.

Chapter 2

Ill-conditioned problems

In order to appreciate why a polynomial in the presence of errors, including roundoff errors, can fail to find a multiple root, it is necessary to explain the concepts of *forward error, backward error* and *condition number* with respect to a measure of how much the errors can affect a change of a computed solution. This chapter contains explanations of these concepts, along with conditioning of the roots of a polynomial, especially for multiple roots, and a geometric interpretation of conditioning of a polynomial.

2.1 Forward and backward error and condition number

Consider a function $y = f(x)$ to be evaluated by a numerical algorithm. An approximation $\tilde{y}$ is the result of the algorithm and different from the exact solution y in most cases. So how can the quality of $\tilde{y}$ to be determined? The simplest error measure is the *forward error*, which is defined as the difference between the result and the solution; in this case, $\Delta y = \tilde{y} - y$. However, this is not always possible because the exact answer may not be known. The *backward error* is therefore used and equal to Δx such that $f(x + \Delta x) = \tilde{y}$; in other words, the backward error explains that the computed solution in error is the theoretically exact solution of a neighboring problem. In general, it is more natural to consider the relative errors $|\Delta y|/|y|$ and $|\Delta x|/|x|$ instead of the absolute errors Δy and Δx respectively. The relationship between these two errors is shown in Figure 2.1, which is reproduced from [28].

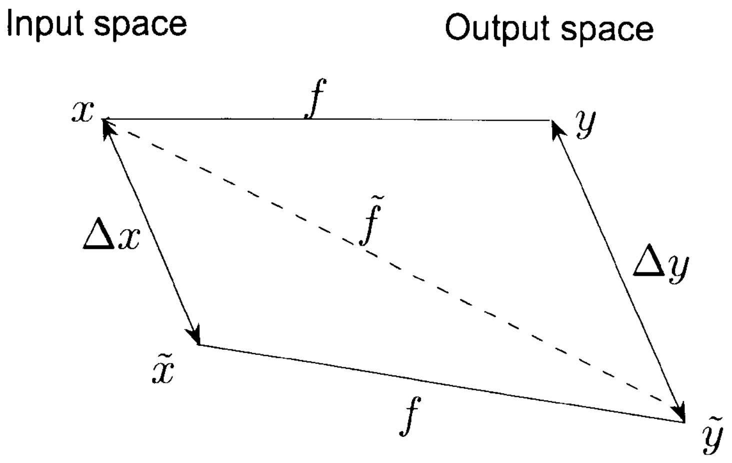


Figure 2.1: The forward error Δy and the backward error Δx, and their relation to the exact solution map f and the computed solution $\tilde{f}$.

It is seen that the forward error is measured in the output space (solution space), and the backward error is measured in the input space (data space).

The forward and backward errors of a function are directly related by the *condition number*, which is defined as a measure of the sensitivity of a function to evaluation with respect to a class of perturbations applied to the data (input parameters). If tiny perturbations in the input space, corresponding to a small backward error, will lead to a comparatively large change in the output space, i.e. a large forward error, then the problem is said to be *ill-conditioned*. Also, it can be said in another way that a problem with a high condition number is said to be *ill-conditioned* and hence highly sensitive to perturbations, while a problem with a low condition number is said to be *well-conditioned* and robust with respect to the specified class of perturbations.

Since the concepts of *forward error, backward error* and *condition number* have been explained, the conditioning of a root of a polynomial is considered in the next section.

2.2 Ill-conditioned polynomial

It is natural to expect that the problem of finding the roots of a polynomial is well-conditioned, that is, a small change in the coefficients of the polynomial will result in a small change in the roots. Unfortunately, that is not the case here. Typically, the problem is highly ill-conditioned when the polynomial has high degree and multiple roots. It has been shown in Example 1.2 that the multiple root is ill-conditioned and splits into a cluster when a random perturbation is applied to the constant coefficient of the polynomial.

If a theoretically exact polynomial is given by

$$\hat{f}(x) = \sum_{i=0}^{m} \hat{a}_i x^{m-i}, \qquad \hat{a}_0 \neq 0, \tag{2.1}$$

it is easy to extend the fundamental theorem of algebra to prove the existence of the factorization

$$\hat{f}(x) = \hat{a}_0 \prod_{j=1}^{n} (x - x_j)^{m_j}, \qquad \sum_{j=1}^{n} m_j = m, \tag{2.2}$$

for any polynomial $\hat{f}(x)$, so that $x_1, x_2, \ldots, x_n$ are the distinct roots of $\hat{f}(x)$ with multiplicities $m_1, m_2, \ldots, m_n$ respectively. A simple error model is assigned to the coefficients of the polynomial $\hat{f}(x)$, that is, a componentwise error model is applied. Each coefficient $\hat{a}_i$ is perturbed to $\hat{a}_i + \Delta a_i$ such that

$$\hat{a}_i + \Delta a_i = \hat{a}_i(1 + r_i \varepsilon_c), \qquad i = 0, \ldots, m, \tag{2.3}$$

where r_i is a uniformly distributed random variable in the range $[-1, 1]$ and ε_c^{-1} is the upper bound of componentwise signal-to-noise ratio. It follows that the componentwise error model is define by

$$|\Delta a_i| \leq \varepsilon_c |\hat{a}_i|, \qquad i = 0, \ldots, m, \tag{2.4}$$

that is, Δa_i is a uniformly distributed random variable in the range $[-\varepsilon_c|\hat{a}_i|, \varepsilon_c|\hat{a}_i|]$. This componentwise error model is used exclusively in this thesis. Also, the componentwise condition number of a root of a polynomial is considered in the following theorem [66].

Theorem 2.2.1. *Let the coefficients $\hat{a}_i$ of $\hat{f}(x)$ in (2.1) be perturbed to $\hat{a}_i + \Delta a_i$ where $|\Delta a_i| \leq \varepsilon_c |\hat{a}_i|, i = 0, \ldots, m$. Let the real root x_0 of $\hat{f}(x)$ have multiplicity r, and let one of these r roots be perturbed to $x_0 + \Delta x_0$ due to the perturbations in the coefficients. Then the componentwise condition number of x_0 is*

$$\kappa_c(x_0) = \max_{|\Delta a_i| \leq \varepsilon_c |\hat{a}_i|} \frac{\Delta x_0}{|x_0|} \frac{1}{\varepsilon_c} = \frac{1}{\varepsilon_c^{1-\frac{1}{r}}} \frac{1}{|x_0|} \left(\frac{r!}{|\hat{f}^r(x_0)|} \sum_{i=0}^{m} |\hat{a}_i x_0^{m-i}| \right)^{\frac{1}{r}}. \tag{2.5}$$

The proof is described in [66], pages 18 − 20, by Winkler. He also pointed out on page 22 that the componentwise backward error and condition number of a root x_0 of multiplicity r of $\hat{f}(x)$ are related in a simple formula to the forward error of x_0 as

$$\frac{|\Delta x_0|}{|x_0|} = \kappa_c(x_0)\left(\frac{\eta_c(\tilde{x}_0)}{\varepsilon_c}\right)^{\frac{1}{r}}\varepsilon_c, \tag{2.6}$$

where $\eta_c(\tilde{x}_0)$, the componentwise backward error of the approximate root $\tilde{x}_0$ of the root x_0 of $\hat{f}(x)$, is given by

$$\eta_c(\tilde{x}_0) = \frac{|\hat{f}(\tilde{x}_0)|}{\sum_{i=0}^{m}|\hat{a}_i\tilde{x}_0^{m-i}|}.$$

It follows that if $r = 1$, that is, x_0 is a simple root, its forward error is equal to the product of its condition number and the backward error of its approximation $\tilde{x}_0$. If r is sufficiently large, then (2.6) reduces to

$$\frac{|\Delta x_0|}{|x_0|} \approx \kappa_c(x_0)\varepsilon_c, \tag{2.7}$$

which is the condition for which (2.5) attains equality, that is, $\kappa_c(x_0)$ attains its maximum value as r increases.

Example 2.1. Consider the polynomial $f(x) = x^2$, whose coefficients are perturbed such that

(a) $f_1(x) = (1+\varepsilon)x^2$, (b) $f_2(x) = x^2 + \varepsilon x$, (c) $f_3(x) = x^2 + \varepsilon$.

If a polynomial is given by $f(x) = a_0x^2 + a_1x + a_2$, then it yields $a_0 = 1, a_1 = 0$ and $a_2 = 0$ when $f(x) = x^2$. Suppose a componentwise error model is applied to the coefficients of the polynomial $f(x)$. Each coefficient a_i is perturbed to $a_i + \Delta a_i$ such that

$$\Delta a_0 = r_0\varepsilon a_0 = r_0\varepsilon, \qquad \Delta a_1 = r_1\varepsilon a_1 = 0, \qquad \Delta a_2 = r_2\varepsilon a_2 = 0,$$

according to (2.3), where $r_i, i = 0, \ldots, 2$ are uniformly distributed random variables in the range $[-1, 1]$ and ε^{-1} is the upper bound of componentwise signal-to-noise ratio.

(a) The perturbed polynomial that is $f_1(x) = (1 + r_0\varepsilon)x^2, r_0 = 1$, has a double root at $x = 0$. This root is extremely stable because a change in the coefficient does not cause any change in the root.

(b) $f_2(x) = x^2 + \varepsilon x$ has roots at $x_1 = 0$ and $x_2 = -\varepsilon$, but (2.5) can not be used to calculate their componentwise condition numbers because $\Delta a_1 = \varepsilon \neq 0$, however, (2.5) requires that $\Delta a_1 = 0$.

(c) $f_3(x) = x^2 + \varepsilon$ has roots at $x = \pm(-\varepsilon)^{\frac{1}{2}}, \varepsilon < 0$, but (2.5) can not be used to calculate their componentwise condition numbers because $\Delta a_2 = \varepsilon \neq 0$. This is exactly the same as (b) above.

□

It is well known that any multiple root will generally, on the introduction of random perturbations applied to the coefficients of a polynomial, split into a cluster, as demonstrated in Examples 2.2 and 2.3.

Example 2.2. Consider four polynomials $(x-1)^3, (x-1)^6, (x-1)^{12}$ and $(x-1)^{20}$, whose coefficients have been randomly perturbed by noise and roots computed 500 times, using the value $\varepsilon_c = 10^{-8}$. The root distributions of the perturbed polynomials are shown in Figure 2.2.

It is well known that a multiple root can split into a dense cluster of closely spaced roots due to finite precision arithmetic and inexact input data. It is however possible to determine the location and multiplicity of a dense cluster by symbolic computations with floating-point arithmetic [31].

If the radius of the cluster is small, and the polynomial contains an isolated multiple root, then Figures 2.2(i) and (ii) seem to suggest that the original root is the approximation of the cluster of roots by a multiple root at the arithmetic mean of the cluster. Although this approach is a simple solution with an obvious justification, it becomes difficult to determine the location and multiplicity of the cluster as the multiplicity of the root increases, which is shown in Figures 2.2(iii) and (iv). □

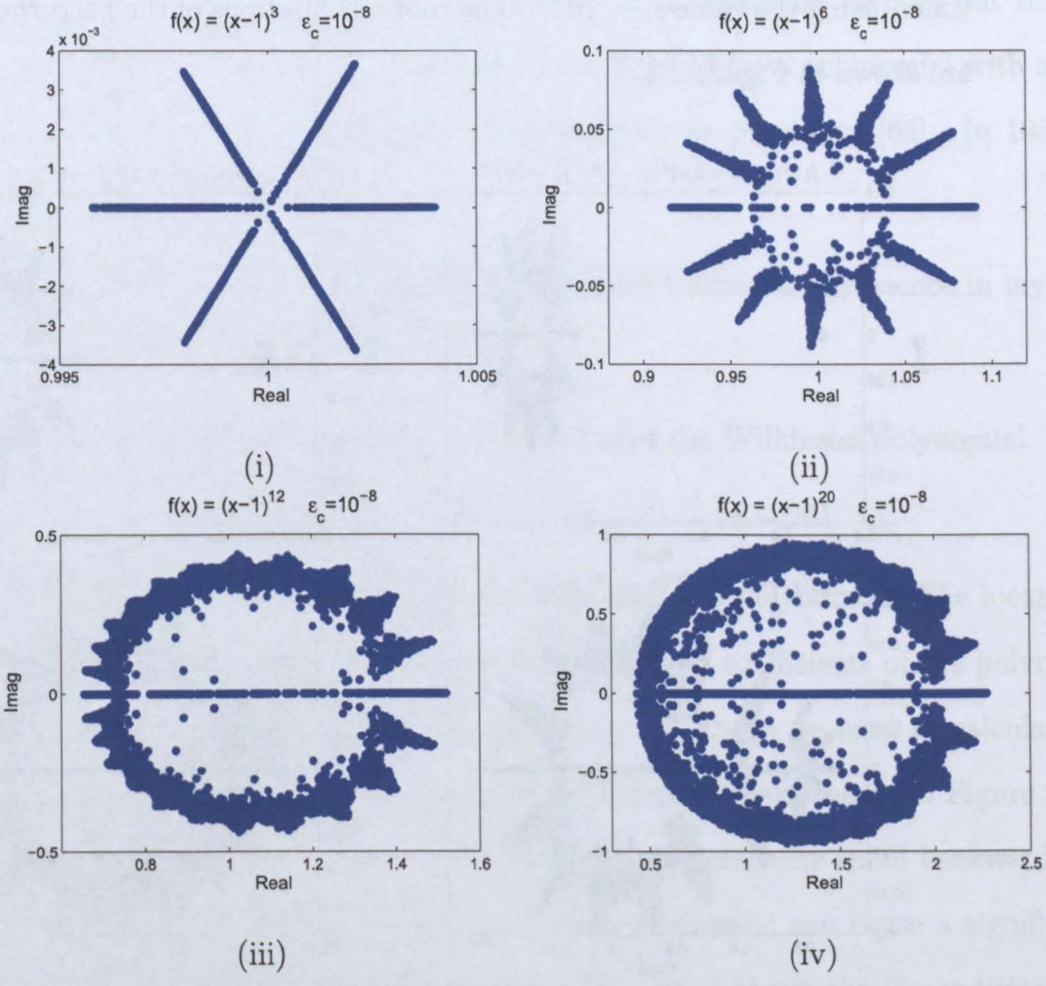


Figure 2.2: The root distribution of $f(x)$ after the coefficients have been perturbed and roots calculated 500 times by the `roots` function in MATLAB.

Example 2.3. Consider four polynomials

$$f_1(x) = (x-0.3)^3(x-1)^6, \qquad f_2(x) = (x-0.5)^3(x-1)^6,$$

$$f_3(x) = (x-0.7)^3(x-1)^6, \qquad f_4(x) = (x-0.9)^3(x-1)^6,$$

whose coefficients have been randomly perturbed by noise and roots computed 500 times, using the value $\varepsilon_c = 10^{-8}$. The root distributions of the perturbed polynomials are shown in Figure 2.3.

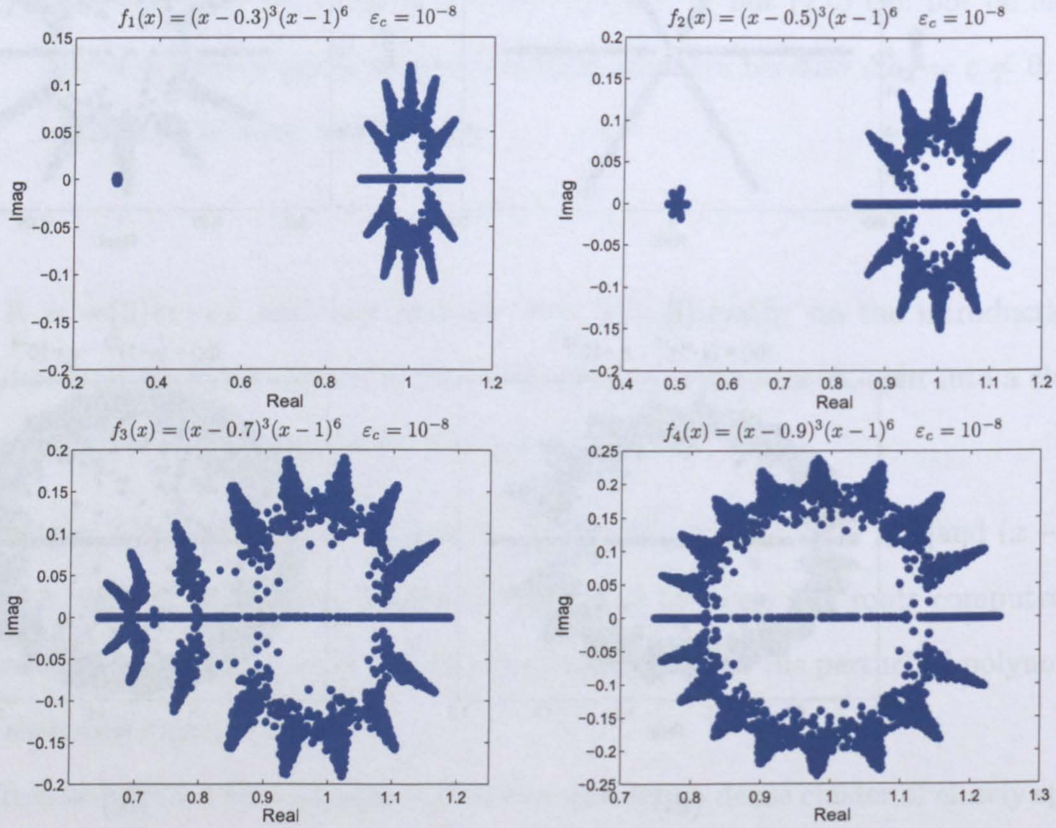


Figure 2.3: The root distribution of four polynomials after the coefficients have been perturbed and roots calculated 500 times by the `roots` function in MATLAB.

The experiment is repeated in Figure 2.3, with two multiple roots whose separation is reduced. It is seen that the values can be estimated by simple clustering when the roots are well separated. The clusters begin to merge, however, as two roots merge until they cannot be distinguished. More examples in which clustering fails to provide the correct multiple roots can be found in [47]. □

It therefore seems that ill-conditioning also occurs when a polynomial has multiple roots and/or closely spaced roots. However, James Wilkinson pointed out the fact that the problem may also be extremely ill-conditioned for a polynomial with simple and well-spaced roots regardless of its multiplicity or proximity [65]. In 1984, he described this discovery:

> "Speaking for myself I regard it as the most traumatic experience in my career as a numerical analyst."

Example 2.4. Consider a specific example, called the Wilkinson polynomial

$$f(x) = \prod_{i=1}^{20}(x-i) = (x-1)(x-2)\cdots(x-20), \tag{2.8}$$

which illustrates a difficulty with finding the roots of a polynomial: The location of the roots can be very sensitive to perturbations in the coefficients of the polynomial [65]. A Newton-Raphson solver [62], pages 174 – 179, can be used to calculate the roots, along with their forward and backward errors that are shown in Figure 2.4.

It can be clearly seen from the Figure 2.4 that a relatively small backward error in the input space owing to the finite precision arithmetic can cause a significantly larger forward error in the solution space. The detail about the ill-conditioning of this polynomial has been discussed in [64, 65]. □

The perturbations considered in Examples 2.2 and 2.3 are random (unstructured), and as noted above, they are associated with the break up of a multiple root. However, structured perturbations can be applied to preserve the multiplicity of the root, such

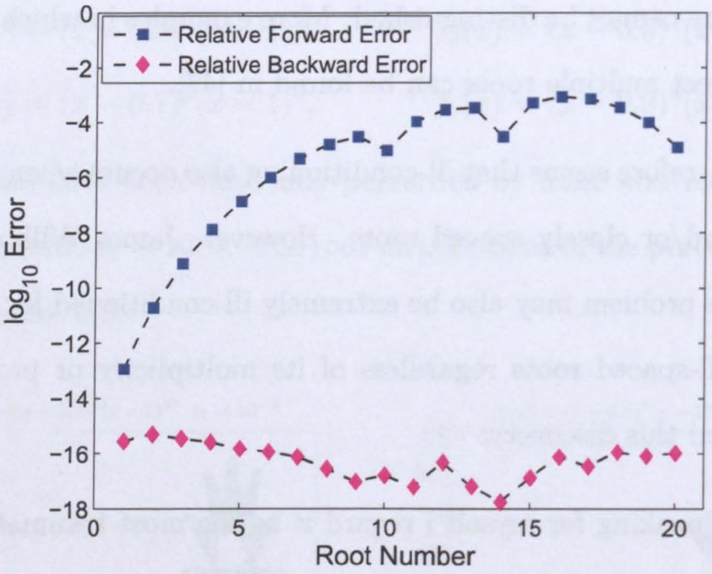


Figure 2.4: Analysis of the computed roots of (2.8).

that the multiple root does not break up, that is, the multiple root is well-conditioned with respect to these perturbations. The details about these structured perturbations are explained in the next section.

2.3 The geometry of ill-conditioned polynomial

Generally, a polynomial of degree m has its own multiplicity structure, that is, a polynomial of degree 5, for instance,

- the polynomial $(x-a)^5$ has a multiplicity structure $\{5\}$,
- the polynomial $(x-a)(x-b)^4, a \neq b$, has a multiplicity structure $\{1,4\}$ or $\{4,1\}$,

- the polynomial $(x-a)^2(x-b)^3, a \neq b$, has a multiplicity structure $\{2,3\}$ or $\{3,2\}$,
- the polynomial $(x-a)(x-b)(x-c)^3, a \neq b \neq c$, has a multiplicity structure $\{1,1,3\}$ or $\{1,3,1\}$ or $\{3,1,1\}$,
- the polynomial $(x-a)(x-b)^2(x-c)^2, a \neq b \neq c$, has a multiplicity structure $\{1,2,2\}$ or $\{2,1,2\}$ or $\{2,2,1\}$,
- the polynomial $(x-a)(x-b)(x-c)(x-d)^2, a \neq b \neq c \neq d$, has a multiplicity structure $\{1,1,1,2\}$ or $\{1,1,2,1\}$ or $\{1,2,1,1\}$ or $\{2,1,1,1\}$,
- the polynomial $(x-a)(x-b)(x-c)(x-d)(x-e), a \neq b \neq c \neq d \neq e$, has a multiplicity structure $\{1,1,1,1,1\}$.

Kahan [37] states that a polynomial of degree m with a certain multiplicity structure lies on a *pejorative manifold*. It is also stated that the pejorative manifold of a polynomial plays an important role in determining if it is ill-conditioned when it has one or more multiple roots.

In general, a multiple root is well-conditioned when the multiplicity of the root is preserved due to the structured perturbations such that the polynomial stays on its pejorative manifold. It is, however, ill-conditioned with respect to perturbations that move the polynomial off the pejorative manifold, in which case the multiple root splits into a cluster of simple roots.

and $x_1 = x_2 = x_3$.

- If $\hat{f}(x)$ has a double root and a simple root, then $x_1 = x_2 \neq x_3$, and thus the system $G(\mathbf{x}) = \hat{\mathbf{a}}$ is given by

$$G(\mathbf{x}) = \begin{cases} -2x_2 - x_3 &= \hat{a}_1 \\ x_2^2 + 2x_2x_3 &= \hat{a}_2 \qquad x_2 \neq x_3, \quad x_2, x_3 \in \mathbb{R}. \\ -x_2^2x_3 &= \hat{a}_3, \end{cases}$$

The pejorative manifold of a cubic polynomial that has a double root is, therefore, a surface in $\mathbb{R}^3$, which is shown in Figure 2.5. Different points on the surface correspond to different values of the double root x_2 and simple root x_3.

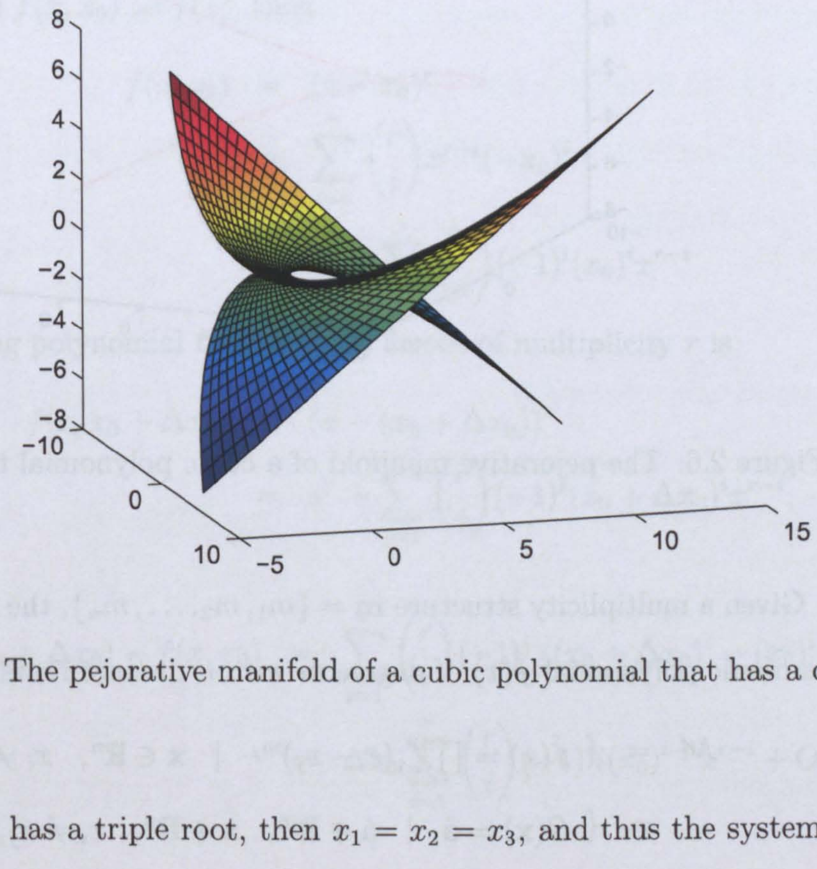


Figure 2.5: The pejorative manifold of a cubic polynomial that has a double root.

- If $\hat{f}(x)$ has a triple root, then $x_1 = x_2 = x_3$, and thus the system $G(\mathbf{x}) = \hat{\mathbf{a}}$ is given by

$$G(\mathbf{x}) = \begin{cases} -3x_1 &= \hat{a}_1 \\ 3x_1^2 &= \hat{a}_2 \qquad x_1 \in \mathbb{R}. \\ -x_1^3 &= \hat{a}_3, \end{cases}$$

The pejorative manifold of a cubic polynomial that has a triple root is, therefore, a curve in $\mathbb{R}^3$, which is shown in Figure 2.6. Different points along the curve correspond to different values of the triple root x_1. □

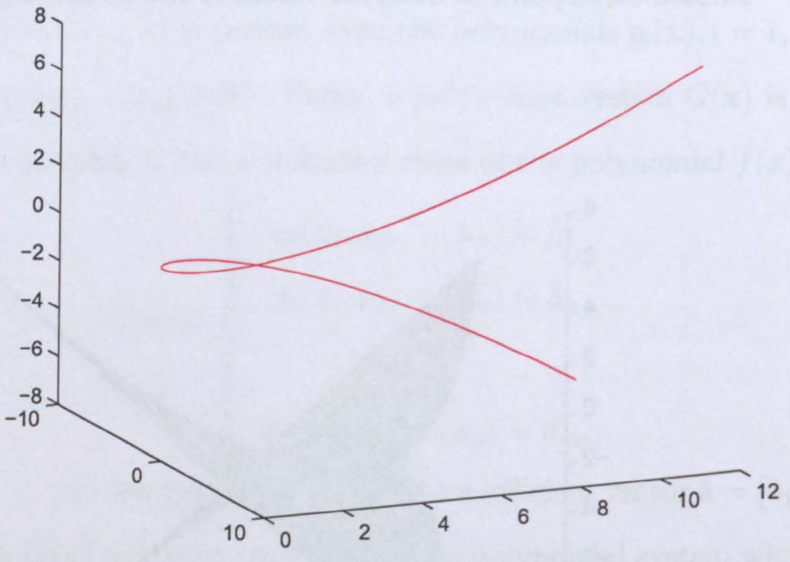


Figure 2.6: The pejorative manifold of a cubic polynomial that has a triple root.

Given a multiplicity structure $\mathbf{m} = \{m_1, m_2, \ldots, m_n\}$, the pejorative manifold $\mathcal{M}$ of a monic polynomial $\hat{f}(x)$ of degree m with n distinct roots for $\mathbf{m}$ is

$$\begin{aligned} \mathcal{M} &= \{ \ \hat{f}(x) = \textstyle\prod_{j=1}^{n}(x - x_j)^{m_j} \ \mid \ \mathbf{x} \in \mathbb{R}^n, \ x_i \neq x_j, \ i \neq j \ \} \\ &= \{ \ G(\mathbf{x}) = \hat{\mathbf{a}} \ \mid \ \hat{\mathbf{a}} \in \mathbb{R}^m, \ \mathbf{x} \in \mathbb{R}^n, \ x_i \neq x_j, \ i \neq j \ \}. \end{aligned} \tag{2.9}$$

For all polynomials whose roots have the same multiplicity structure $\mathbf{m}$, the system $G(\mathbf{x}) = \hat{\mathbf{a}}$ defines the pejorative manifold $\mathcal{M}$ as a surface of dimension n in the space $\mathbb{R}^m$.

It was stated above that a multiple root is well-conditioned when the multiplicity of the root is preserved, in which case the polynomial stays on its pejorative manifold. This result is established in the next theorem [66].

Theorem 2.3.1. *The condition number of the real root x_0 of multiplicity r of the polynomial $f(x) = (x - x_0)^r$, such that the perturbed polynomial also has a root of multiplicity r, is*

$$\rho(x_0) := \frac{|\Delta x_0|/|x_0|}{\|\Delta f\|/\|f\|} = \frac{1}{r|x_0|}\frac{\|(x-x_0)^r\|}{\|(x-x_0)^{r-1}\|} = \frac{1}{r|x_0|}\left(\frac{\sum_{i=0}^{r}\binom{r}{i}^2(x_0)^{2i}}{\sum_{i=0}^{r-1}\binom{r-1}{i}^2(x_0)^{2i}}\right)^{\frac{1}{2}}, \quad (2.10)$$

Proof. If $f(x, x_0) := f(x)$, then

$$\begin{aligned} f(x, x_0) &= (x - x_0)^r \\ &= \sum_{i=0}^{r}\binom{r}{i}x^{r-i}(-x_0)^i \\ &= x^r + \sum_{i=1}^{r}\binom{r}{i}(-1)^i(x_0)^i x^{r-i} \end{aligned}$$

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

$$\begin{aligned} f(x, x_0 + \Delta x_0) &= (x - (x_0 + \Delta x_0))^r \\ &= x^r + \sum_{i=1}^{r}\binom{r}{i}(-1)^i(x_0 + \Delta x_0)^i x^{r-i}, \end{aligned}$$

and hence

$$\begin{aligned} f(x, x_0 + \Delta x_0) - f(x, x_0) &= \sum_{i=1}^{r}\binom{r}{i}(-1)^i\left((x_0 + \Delta x_0)^i - (x_0)^i\right)x^{r-i} \\ &= \Delta x_0 \sum_{i=1}^{r}\binom{r}{i}(-1)^i i(x_0)^{i-1}x^{r-i} + \mathrm{O}(\Delta x_0^2). \end{aligned}$$

Since

$$\begin{aligned}(x-x_0)^{r-1} &= \sum_{i=0}^{r-1}\binom{r-1}{i}x^{r-1-i}(-x_0)^i \\ &= -\frac{1}{r}\sum_{i=1}^{r}\binom{r}{i}(-1)^i i(x_0)^{i-1}x^{r-i},\end{aligned}$$

it follows that to first order,

$$\Delta f := f(x,x_0+\Delta x_0) - f(x,x_0) = -r\Delta x_0(x-x_0)^{r-1},$$

and thus the condition number of x_0 that preserves its multiplicity is

$$\begin{aligned}\frac{|\Delta x_0|/|x_0|}{\|\Delta f\|/\|f\|} &= \frac{1}{r|x_0|}\frac{\|(x-x_0)^r\|}{\|(x-x_0)^{r-1}\|} \\ &= \frac{1}{r|x_0|}\frac{\|\sum_{i=0}^{r}\binom{r}{i}(-x_0)^i x^{r-i}\|}{\|\sum_{i=0}^{r-1}\binom{r-1}{i}(-x_0)^i x^{r-1-i}\|} \\ &= \frac{1}{r|x_0|}\left(\frac{\sum_{i=0}^{r}\binom{r}{i}^2(x_0)^{2i}}{\sum_{i=0}^{r-1}\binom{r-1}{i}^2(x_0)^{2i}}\right)^{\frac{1}{2}}.\end{aligned}$$

□

Example 2.6. The condition number $\rho(1)$ of the root $x_0 = 1$ of the polynomial $(x-1)^r$ is, from (2.10),

$$\rho(1) = \frac{1}{r}\left(\frac{\sum_{i=0}^{r}\binom{r}{i}^2}{\sum_{i=0}^{r-1}\binom{r-1}{i}^2}\right)^{\frac{1}{2}}.$$

Since in combinatorics, Vandermode's identity for binomial coefficients [3], pages 59 – 60, states that

$$\sum_{i=0}^{r}\binom{m}{i}\binom{n}{r-i} = \binom{m+n}{r}$$

and thus if $m = n = r$, then

$$\sum_{i=0}^{r}\binom{r}{i}^2 = \binom{2r}{r},$$

and it follows that

$$\rho(1) = \frac{1}{r}\sqrt{\frac{\binom{2r}{r}}{\binom{2(r-1)}{r-1}}} = \frac{1}{r}\sqrt{\frac{2(2r-1)}{r}} \approx \frac{2}{r},$$

if r is large. The condition number must be compared with the componentwise condition number, from (2.7)

$$\kappa_c(1) \approx \frac{|\Delta x_0|}{\varepsilon_c},$$

which is proportional to the signal-to-noise ratio. By contrast, $\rho(1)$ is independent of the perturbation of the polynomial and it decreases as the multiplicity r of the root $x_0 = 1$ increases. □

It is therefore stated that a multiple root is well-conditioned when the multiplicity of the root is preserved, in which case the polynomial stays on its pejorative manifold. In other words, if a polynomial with multiple roots lies on a pejorative manifold, then small perturbations on the manifold result in small changes in the values of the roots, that is, the multiplicity structure of the polynomial is preserved.

2.4 Summary

In this chapter the concepts of forward error and backward error have been introduced, including their relationship with the condition number. Moreover, it has been demonstrated that a multiple root is ill-conditioned, with evidence of increasing instability as its multiplicity increases and the break up as a cluster of simple roots, when random perturbations are assigned to the coefficients of the polynomial. In addition, the Wilkinson polynomial has been presented to prove that the occurrence of ill-conditioning does not only depend on the multiplicity and proximity of a root.

Also, a multiple root is well-conditioned when a structured perturbation that preserves the multiplicities of the roots is applied to the coefficients of the polynomial, that is, the perturbed polynomial has a root of the same multiplicity as the original polynomial. The pejorative manifold of a polynomial has been defined in order to motivate a geometric interpretation of ill-conditioning.

Chapter 3

A simple polynomial root solver

The conditioning of the roots of a polynomial has been discussed in Chapter 2 with particular emphasis on the effect of the root's multiplicity. A simple root is, in general, better conditioned than a multiple root and it is therefore instructive to develop a polynomial root solver that reduces the computation of the roots themselves to the solution of a sequence of polynomial equations with simple roots only. This method, which was known as early as 1863 by Gauss, is described in [62], pages 65 – 68, by Uspensky.

This method differs from the methods that are mentioned in Chapter 1 because the multiplicities of the roots are computed initially through a sequence of the greatest common divisor (GCD) computations, after which the values of the roots are calculated though polynomial division operations. Once the multiplicities of the roots are obtained, the calculation of the values of the roots is a well-conditioned problem because the multiple roots are kept on their pejorative manifold. The computation of multiple roots of a polynomial can also be applied to the computation of multiple eigenvalues [36]. Hence, a robust GCD-finder is crucial to the study of root-finding when the polynomials involve multiple roots [14, 51, 70].

A simple polynomial root solver is therefore described in this chapter. The operations that are required for the root solver are considered and it is shown that their implementation in a floating point environment is not trivial because they are ill-posed. Moreover, the data in many practical examples is inexact, and thus a practical root solver must be robust with respect to minor perturbations in the coefficients of the polynomial. The concept of *ill-posed problem* is explained in the next section.

3.1 Well-posed and ill-posed problems

The mathematical term *well-posed problem* stems from a definition given by Hadamard. In [25], he claims that a mathematical model of a physical problem has to be well-posed in the sense that it has the following three properties:

1. There exists a solution of the problem (existence).
2. There is at most one solution of the problem (uniqueness).
3. The solution depends continuously on the data (stability).

Mathematically, according to the remarks of Kirsch [40], the existence of a solution can be enforced by enlarging the solution space. If a problem has more than one solution, then information about the model is missing, and thus additional properties can be built into the model. The requirement of stability is the most important one. He notes on page 10 that:

> "If a problem lacks the property of stability, then its solution is practically impossible to compute because any measurement or numerical computation is polluted by unavoidable errors: thus the data of a problem are

always perturbed by noise! If the solution of a problem does not depend continuously on the data, then in general the computed solution has nothing to do with the true solution."

Problems that are not well-posed are termed *ill-posed* in the sense of Hadamard. Hence a problem is ill-posed if no solution exists, the problem may have more than one solution or the solution depends discontinuously upon the initial data. The GCD computation and polynomial division are often ill-posed, which would be explained in detail in Section 3.3 and Chapter 8, respectively.

An ill-conditioned problem differs from an ill-posed problem because properties 1 and 2 mentioned above are satisfied by an ill-conditioned problem. The third property is not satisfied because an ill-conditioned problem is very unstable for which a small error in the initial data can result in much larger errors in the solutions. Even if a problem is well-posed, it may still be ill-conditioned, that is, the solution may still be sensitive to the input data. The Wilkinson polynomial, which has been shown in Example 2.4, is an example because the roots are very sensitive to changes in the coefficients of the polynomial, but they are continuous functions of the coefficients.

3.2 Factorisation via GCD computations

The polynomial root solver is considered in this section and it will be apparent that it differs significantly from the root solvers mentioned in Chapter 1.

Consider the polynomial

$$d_0(x) = (x - x_1)^{m_1}(x - x_2)^{m_2} \cdots (x - x_n)^{m_n} \varrho_0(x),$$

where $m_i \geq 2, i = 1, \ldots, n$, and $\varrho_0(x)$ contains only simple roots. Since a root x_i of multiplicity m_i of $d_0(x)$ is a root x_i of multiplicity $m_i - 1$ of the derivative polynomial $d_0^{(1)}(x)$, it follows that

$$d_0^{(1)}(x) = (x - x_1)^{m_1 - 1}(x - x_2)^{m_2 - 1} \cdots (x - x_n)^{m_n - 1} \varrho_1(x),$$

where $\varrho_0(x), \varrho_1(x)$ are coprime polynomials and the roots of $\varrho_1(x)$ are simple. Therefore the GCD of $d_0(x)$ and $d_0^{(1)}(x)$ is

$$\mathrm{GCD}\left(d_0(x), d_0^{(1)}(x)\right) = (x - x_1)^{m_1 - 1}(x - x_2)^{m_2 - 1} \cdots (x - x_n)^{m_n - 1}.$$

In general, let $\chi_1(x)$ be the product of all linear factors corresponding to simple roots of $d_0(x)$, $\chi_2(x)$ be the product of all quadratic factors corresponding to double roots of $d_0(x)$, ..., $\chi_{m_*}(x)$ be the product of all factors of degree m_* corresponding to the roots of multiplicity m_* of $d_0(x)$, where m_* is the maximum multiplicity of the roots of $d_0(x)$. If $d_0(x)$ has no root of multiplicity i, $\chi_i(x)$ can be set equal to a constant. Then,

$$\chi_1(x)\chi_2^2(x) \cdots \chi_{m_*}^{m_*}(x)$$

differs only by a constant factor from $d_0(x)$, and thus

$$d_1(x) = \mathrm{GCD}\left(d_0(x), d_0^{(1)}(x)\right) = \chi_2(x)\chi_3^2(x) \cdots \chi_{m_*}^{m_* - 1}(x).$$

Similarly,

$$\begin{aligned} d_2(x) = \mathrm{GCD}\left(d_1(x), d_1^{(1)}(x)\right) &= \chi_3(x)\chi_4^2(x) \cdots \chi_{m_*}^{m_* - 2}(x), \\ d_3(x) = \mathrm{GCD}\left(d_2(x), d_2^{(1)}(x)\right) &= \chi_4(x)\chi_5^2(x) \cdots \chi_{m_*}^{m_* - 3}(x), \\ &\vdots \end{aligned}$$

and the sequence terminates at $d_{m_*}(x)$ which is a constant. A sequence of polynomials

$\tau_i(x), i = 1, \ldots, m_*$, can be defined such that

$$\begin{aligned}
\tau_1(x) &= \tfrac{d_0(x)}{d_1(x)} &= \chi_1(x)\chi_2(x)\cdots\chi_{m_*}(x), \\
\tau_2(x) &= \tfrac{d_1(x)}{d_2(x)} &= \chi_2(x)\chi_3(x)\cdots\chi_{m_*}(x), \\
\tau_3(x) &= \tfrac{d_2(x)}{d_3(x)} &= \chi_3(x)\chi_4(x)\cdots\chi_{m_*}(x), \\
&\vdots \\
\tau_{m_*}(x) &= \tfrac{d_{m_*-1}(x)}{d_{m_*}(x)} &= \chi_{m_*}(x),
\end{aligned}$$

from which all functions $\chi_1(x), \chi_2(x), \ldots, \chi_{m_*}(x)$ are

$$\chi_1(x) = \frac{\tau_1(x)}{\tau_2(x)}, \quad \chi_2(x) = \frac{\tau_3(x)}{\tau_4(x)}, \quad \cdots \quad , \chi_{m_*-1}(x) = \frac{\tau_{m_*-1}(x)}{\tau_{m_*}(x)},$$

until

$$\chi_{m_*}(x) = \tau_{m_*}(x).$$

This leads to the polynomial equations

$$\chi_1(x) = 0, \quad \chi_2(x) = 0, \quad \cdots \quad , \chi_{m_*}(x) = 0,$$

all of which contain only simple roots. They yield the simple, double, triple roots, etc., of $d_0(x)$, If x_0 is a root of $\chi_i(x)$, then it is a root of multiplicity i of $d_0(x)$. If some $\chi_i(x)$ are constants, then there is no root of multiplicity i. Algorithm 3.1 contains pseudo-code for the implementation of this method described by Uspensky for the calculation of the roots of a polynomial.

Algorithm 3.1: The calculation of the roots of a polynomial

```
Input A polynomial $d_0(x)$.
Output The roots of $d_0(x)$.
Begin
```

```
1. Set $j = 0$.
2. **While** degree $d_j > 0$ **do**
    (a) Set $j = j + 1$.
    (b) Calculate the GCD of $d_{j-1}$ and its derivative $d_{j-1}^{(1)}$,
            $d_j = \text{GCD}\left(d_{j-1}, d_{j-1}^{(1)}\right)$.
   **End While**
3. Calculate $\tau_i = \frac{d_{i-1}}{d_i}, i = 1, \ldots, j$.
4. Calculate $\chi_i = \frac{\tau_i}{\tau_{i+1}}, i = 1, \ldots, j-1$.
5. Set $\chi_j = \tau_j$.
6. Calculate the roots of $\chi_i, i = 1, \ldots, j$.        % They are of multiplicity $i$.
**End**
```

Example 3.1. Consider the polynomial

$$\begin{aligned} d_0(x) &= x^{10} - 7x^9 + 9x^8 + 29x^7 - 53x^6 \\ &\quad -57x^5 + 91x^4 + 71x^3 - 48x^2 - 36x, \end{aligned}$$

whose first derivative is

$$\begin{aligned} d_0^{(1)}(x) &= 10x^9 - 63x^8 + 72x^7 + 203x^6 - 318x^5 \\ &\quad -285x^4 + 364x^3 + 213x^2 - 96x - 36. \end{aligned}$$

It follows that

$$\begin{aligned} d_1(x) &= \text{GCD}\left(d_0(x), d_0^{(1)}(x)\right) \\ &= x^5 - 2x^4 - 6x^3 + 4x^2 + 13x + 6 \\ d_1^{(1)}(x) &= 5x^4 - 8x^3 - 18x^2 + 8x + 13 \end{aligned}$$

and then

$$\begin{aligned} d_2(x) &= \text{GCD}\left(d_1(x), d_1^{(1)}(x)\right) = x^2 + 2x + 1 \\ d_2^{(1)}(x) &= 2x + 2 \end{aligned}$$

and hence

$$\begin{aligned} d_3(x) &= \text{GCD}\left(d_2(x), d_2^{(1)}(x)\right) = x + 1 \\ d_4(x) &= \text{GCD}\left(d_3(x), d_3^{(1)}(x)\right) = 1. \end{aligned}$$

The polynomials $\tau_1(x), \tau_2(x), \tau_3(x)$ and $\tau_4(x)$ are

$$\begin{aligned} \tau_1 = \frac{d_0(x)}{d_1(x)} &= x^5 - 5x^4 + 5x^3 + 5x^2 - 6x \\ \tau_2 = \frac{d_1(x)}{d_2(x)} &= x^3 - 4x^2 + x + 6 \\ \tau_3 = \frac{d_2(x)}{d_3(x)} &= x + 1 \\ \tau_4 = \frac{d_3(x)}{d_4(x)} &= x + 1, \end{aligned}$$

and thus the polynomials χ_1, χ_2, χ_3 and χ_4 are

$$\begin{aligned} \chi_1 = \frac{\tau_1(x)}{\tau_2(x)} &= x^2 - x \\ \chi_2 = \frac{\tau_2(x)}{\tau_3(x)} &= x^2 - 5x + 6 \\ \chi_3 = \frac{\tau_3(x)}{\tau_4(x)} &= 1 \\ \chi_4 = \tau_4(x) &= x + 1. \end{aligned}$$

This leads to the polynomial equations

$$\chi_1 = 0 \quad \Rightarrow \quad x_1 = 0, x_2 = 1,$$

$$\chi_2 = 0 \quad \Rightarrow \quad x_3 = 2, x_4 = 3,$$

$$\chi_4 = 0 \quad \Rightarrow \quad x_5 = -1,$$

and thus the polynomial has two simple roots at $x_1 = 0$ and $x_2 = 1$, two double roots at $x_3 = 2$ and $x_4 = 3$, no triple roots, and one root of multiplicity 4 at $x_5 = -1$,

$$d_0(x) = x(x-1)(x-2)^2(x-3)^2(x+1)^4.$$

□

Example 3.1 introduces the process for the computation of the roots of a polynomial. Although this process is easy to follow, some essential steps are implemented in a floating point environment, which raises some difficult issues:

- The computation of the GCD of a polynomial pair is an ill-posed problem because a tiny perturbation can transform the polynomial pair to be coprime. Even for a polynomial pair of exact forms, a non-trivial GCD can be reduced to be a constant because of roundoff errors due to floating point arithmetic.

- The determination of the degree of the GCD reduces to the determination of the rank of a resultant matrix, but the rank of a matrix is not defined in a floating point environment. Since the degree of the GCD is equal to the rank loss of its resultant matrix, a tiny perturbation in the coefficients of the polynomials is sufficient to convert a rank deficient matrix to a matrix of full rank, which suggests that the GCD is a constant.

- Polynomial division reduces to the deconvolution of their coefficients, but it is not simple to obtain a computationally stable solution because this computation is an ill-posed problem.

The given data in many applications is affected by noise that may only be known approximately and not exactly, and thus the polynomials are only specified with a tolerance. It is therefore desirable that a robust polynomial root solver is developed to overcome the difficulties mentioned above, such that the root solver does not require an estimate of the noise level and other data. A substantial part of this thesis is therefore devoted to the solution of the problems discussed above.

It is known that a very important part of this root solver is the determination of the GCD of two polynomials. The computational difficulties associated with this are highlighted in the next section.

3.3 Previous work on GCD computations

Many problems in science and engineering, such as computing theory [1], blind image deconvolution [44, 54], signal processing [69], system identification [60] and control theory [5], require an estimate of the GCD of a polynomial pair in the presence of noise, that is, the computation of the GCD of two polynomials is an essential problem in algebraic and numerical computing. For example, in image processing, the desired image can be regarded as the polynomial GCD between two of its distorted versions of the same scene in the z domain [54].

The usual approach to finding the GCD is to use Euclid's algorithm [10], but this algorithm does not perform well when noise is imposed on the coefficients of one or both polynomials. The calculation of the GCD is an ill-posed problem and therefore not suitable for applications that include inexact data because a tiny random perturbation in the coefficients of a polynomial pair is small enough to reduce a non-trivial GCD to a constant, in which case the inexact polynomial pair are relatively prime (coprime).

Example 3.2. Consider the polynomials,

$$\begin{aligned}\hat{f}(x) &= (x-a)(x-b)(x-c)\\ \hat{g}(x) &= (x-a)(x-b)(x-d),\end{aligned}$$

where $a \neq b \neq c \neq d$, whose GCD is,

$$\text{GCD}(\hat{f}(x), \hat{g}(x)) = (x-a)(x-b).$$

If $\hat{f}(x)$ is perturbed such that,

$$\hat{f}(x) \rightarrow f(x) = (x-(a+\delta a))(x-(b+\delta b))(x-c),$$

where $a+\delta a \neq b \neq d, b+\delta b \neq a \neq d$ and $\delta a, \delta b \neq 0$, then $\deg(\text{GCD}(f(x), \hat{g}(x))) = 1$, that is, $f(x)$ and $\hat{g}(x)$ are coprime. □

This is a major problem in practical applications where it is common for the coefficients (input parameters) to be disturbed by noise [16]. This may be as a result of floating point arithmetic or the involvement of laboratory measurements, which allow only a limited number of significant figures to be obtained.

If data errors are present, the given inexact polynomial pair are with high probability coprime, and must be perturbed slightly in order to induce a non-trivial GCD. This computed GCD is therefore called an approximate GCD with respect to the given inexact and coprime polynomial pair and moreover, it is not unique because different perturbations on the coefficients of the polynomial pair yield different approximate GCDs.

Example 3.3. Consider the polynomials,

$$\begin{aligned}\hat{f}(x) &= (x-a)(x-b)(x-c)\\ \hat{g}(x) &= (x-a)(x-b)(x-d),\end{aligned}$$

where $a \neq b \neq c \neq d$. If $\hat{f}(x)$ and $\hat{g}(x)$ are perturbed such that

$$\begin{aligned}\hat{f}(x) &\rightarrow f(x) = (x-(a+\epsilon_1))(x-(b+\epsilon_2))(x-c)\\ \hat{g}(x) &\rightarrow g(x) = (x-(a+\omega_1))(x-(b+\omega_2))(x-d),\end{aligned}$$

where $|\epsilon_1|, |\epsilon_2|, |\omega_1|, |\omega_2| \leq$ *tolerance*, then an approximate GCD of $f(x)$ and $g(x)$ is equal to

- $x-(a+\epsilon_1)$, if $\epsilon_1 = \omega_1, \epsilon_2 \neq \omega_2$.
- $x-(b+\epsilon_2)$, if $\epsilon_1 \neq \omega_1, \epsilon_2 = \omega_2$.
- $(x-(a+\epsilon_1))(x-(b+\epsilon_2))$, if $\epsilon_1 = \omega_1, \epsilon_2 = \omega_2$.

Different approximate GCDs of $f(x)$ and $g(x)$ are therefore obtained for different noise samples, all of which are less than a threshold. □

Since the first paper about analyzing the approximate GCD problem [58] appeared in 1985, several algorithms for computing an approximate GCD have been developed, and different techniques have been used. A non-iterative maximum likelihood based method is proposed by Stoica and Söderström [60], with an assumption that the noise on the coefficients of the polynomials have a Gaussian random distribution. An optimisation method is introduced by Karmarkar and Lakshman [39] in order to calculate the smallest perturbations that should be applied to the coefficients of a polynomial pair and therefore transform a constant GCD to be a non-trivial GCD. Modifications of the Euclidean algorithm are considered in [7, 31, 51], with a prior accuracy level ϵ, in which case the crucial points are the avoidance of the ill-conditioned remainders and the choice of the termination criterion. Pan [53] uses root grouping and the Padé approximation to compute an approximate GCD and argues that perturbing the zeros of a polynomial pair is more efficient than perturbing the coefficients.

In recent years, researchers have investigated matrix-based methods, and in particular, the relationship between an approximate GCD and a resultant matrix. The singular value decomposition (SVD) of the Sylvester resultant matrix $S(f, g)$ of two polynomials $f = f(x)$ and $g = g(x)$, which will be called the Sylvester matrix for simplicity, is used in [12, 16] in order to calculate an approximate GCD. Similarly, the QR decomposition of the Sylvester matrix is described in [13, 69], but both these compositions do not preserve the structure of its matrix. Since the smallest non-zero singular value of the Sylvester matrix is a measure of its distance to singularity, this is the distance to an arbitrary rank deficient matrix [24], and not the distance to the nearest rank deficient Sylvester matrix. Furthermore, Bini and Boito [8] use the QR decomposition of the Bézout resultant matrix $B(f, g)$ to compute an approximate GCD and suggest that the QR decomposition of the Sylvester matrix by Corless et al. [13] fails to detect the correct GCD degree if a polynomial has multiple roots or a small leading coefficient.

The method of structured total least norm (STLN) [56] is used to construct a structured low rank approximation of the Sylvester matrix $S(f, g)$. It is shown in [2, 67, 73] that this approach yields an improvement in the approximate GCD computation because the rank deficiency of the low rank approximation of $S(f, g)$ is clearly defined.

3.4 Summary

In this chapter it has been shown that the factorisation of a polynomial via GCD computations can be used to reduce the problem of computing its multiple roots to that of solving a sequence of polynomial equations that contain only simple roots. Hence a simple polynomial root solver has been introduced to calculate the multiplicity of the roots initially, after which the values of the roots are determined. There exist, however, difficult computational issues that must be addressed because the GCD computations and polynomial divisions are ill-posed operations, and thus their implementation with inexact data in a floating point environment requires care. These issues are addressed in subsequent chapters.

Chapter 4

The resultant matrix

It is stated in Section 3.3 that it is common to use a resultant matrix for the calculation of an approximate GCD of two polynomials $f(x)$ and $g(x)$. This also applies to the polynomial root solver that is introduced in Section 3.2, especially for the Sylvester resultant matrix $S(f, g)$ and Bézout resultant matrix $B(f, g)$, and this chapter introduces some of their properties, including theoretical and computational aspects of resultant matrices.

A polynomial pair are relatively prime (coprime) if and only if their resultant matrix is full rank, and if they are not coprime, the degree and coefficients of their GCD can be calculated from their resultant matrix. In particular, according to Barnett [5], the degree of the GCD of the polynomial pair is equal to the rank loss of their resultant matrix, which is determined initially, after which the coefficients of the GCD can be obtained by reducing the matrix to upper triangular form through a QR or LU decomposition [24]. This situation becomes much more complicated when computations are performed in a floating point environment with perturbed coefficients of these polynomials, that is, inexact data and roundoff error can transform a theoretically singular matrix to a non-singular matrix. It is also known that the determination of the rank of a noisy matrix is a challenging problem that arises in many computational fields of science such as numerical analysis, signal processing, control theory, polynomial algebra.

In this chapter, a discussion of subresultant matrices $S_k(f,g)$ is carried out, and it is shown how they can be used to determine the degree of the GCD of $f(x)$ and $g(x)$ in the absence of noise. Since data errors are sufficient to reduce a rank deficient matrix to a full rank matrix, some preprocessing operations should be implemented when computations are performed on the matrices $S_k(f,g)$ in a floating point environment. These operations are therefore discussed in this chapter, and computational experiments show that the omission of these operations leads to a significant degradation in the computed results, particularly in the presence of noise.

4.1 The Sylvester resultant matrix

Let $\hat{f} = \hat{f}(x)$ and $\hat{g} = \hat{g}(x)$ be theoretically exact polynomials of degrees of m and n respectively,

$$\hat{f}(x) = \sum_{i=0}^{m} \hat{a}_i x^{m-i} \qquad \text{and} \qquad \hat{g}(x) = \sum_{i=0}^{n} \hat{b}_i x^{n-i}, \tag{4.1}$$

where $\hat{a}_0, \hat{b}_0 \neq 0$.

The Sylvester resultant matrix $S(\hat{f},\hat{g}) \in \mathbb{R}^{(m+n)\times(m+n)}$ of $\hat{f}(x)$ and $\hat{g}(x)$ is given by

$$S(\hat{f},\hat{g}) = \left[\begin{array}{cccc|cccc}
\hat{a}_0 & & & & \hat{b}_0 & & & \\
\hat{a}_1 & \hat{a}_0 & & & \hat{b}_1 & \hat{b}_0 & & \\
\vdots & \hat{a}_1 & \ddots & & \vdots & \hat{b}_1 & \ddots & \\
\hat{a}_{m-1} & \vdots & \ddots & \hat{a}_0 & \hat{b}_{n-1} & \vdots & \ddots & \hat{b}_0 \\
\hat{a}_m & \hat{a}_{m-1} & \ddots & \hat{a}_1 & \hat{b}_n & \hat{b}_{n-1} & \ddots & \hat{b}_1 \\
 & \hat{a}_m & \ddots & \vdots & & \hat{b}_n & \ddots & \vdots \\
 & & \ddots & \hat{a}_{m-1} & & & \ddots & \hat{b}_{n-1} \\
 & & & \hat{a}_m & & & & \hat{b}_n
\end{array}\right]. \tag{4.2}$$

$$\underbrace{\hspace{4cm}}_{n \text{ columns}} \quad \underbrace{\hspace{4cm}}_{m \text{ columns}}$$

where the coefficients $\hat{a}_i$ of $\hat{f}(x)$ occupy the first n columns, the coefficients $\hat{b}_i$ of $\hat{g}(x)$ occupy the last m columns, and each of the two submatrices is a Toeplitz matrix. It is clear that the matrix $S(\hat{f},\hat{g})$ is strictly linear and partitioned because

$$S(\alpha f + \lambda p, \beta g + \mu q) = S(\alpha f, \beta g) + S(\lambda p, \mu q),$$

where $\alpha, \beta, \lambda, \mu$ are constants, and $f = f(x), g = g(x), p = p(x), q = q(x)$ are polynomials. The derivation of $S(\hat{f},\hat{g})$ relates to its subresultant matrices that arise when the product of two polynomials is written as a matrix-vector product.

In particular, if $\hat{f}(x)$ and $\hat{g}(x)$ have a common divisor polynomial $c_k(x)$ of degree k, there exist quotient polynomials $u_k(x)$ and $v_k(x)$, such that

$$\begin{aligned}
\hat{f}(x) &= c_k(x)u_k(x), \qquad & \deg u_k < \deg \hat{f} = m, \\
\hat{g}(x) &= c_k(x)v_k(x), \qquad & \deg v_k < \deg \hat{g} = n,
\end{aligned} \tag{4.3}$$

for $k = 1, \ldots, \hat{d}$, where $\hat{d}$ is the degree of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$.

$$c_k(x) = \sum_{i=0}^{k} c_{k,i}x^{k-i}, \quad u_k(x) = \sum_{i=0}^{m-k} u_{k,i}x^{m-k-i}, \quad v_k(x) = \sum_{i=0}^{n-k} v_{k,i}x^{n-k-i}. \tag{4.4}$$

It follows from (4.3) that

$$\hat{f}(x)v_k(x) = \hat{g}(x)u_k(x), \qquad k = 1, \ldots, \hat{d}.$$

Since the product of two polynomials is equal to the convolution of their coefficients, these polynomial products on the left and right sides can be written as the product of a Toeplitz matrix, $S_{n-k+1}(\hat{f}) \in \mathbb{R}^{(m+n-k+1)\times(n-k+1)}$ and $S_{m-k+1}(\hat{g}) \in \mathbb{R}^{(m+n-k+1)\times(m-k+1)}$ respectively, and a vector [72],

$$S_{n-k+1}(\hat{f})\mathbf{v}_k = S_{m-k+1}(\hat{g})\mathbf{u}_k, \tag{4.5}$$

where

$$S_{n-k+1}(\hat{f}) = \begin{bmatrix} \hat{a}_0 & & \\ \hat{a}_1 & \ddots & \\ \vdots & \ddots & \hat{a}_0 \\ \hat{a}_{m-1} & \ddots & \hat{a}_1 \\ \hat{a}_m & \ddots & \vdots \\ & \ddots & \hat{a}_{m-1} \\ & & \hat{a}_m \end{bmatrix}, \quad S_{m-k+1}(\hat{g}) = \begin{bmatrix} \hat{b}_0 & & \\ \hat{b}_1 & \ddots & \\ \vdots & \ddots & \hat{b}_0 \\ \hat{b}_{n-1} & \ddots & \hat{b}_1 \\ \hat{b}_n & \ddots & \vdots \\ & \ddots & \hat{b}_{n-1} \\ & & \hat{b}_n \end{bmatrix},$$

and

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

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

The expression (4.5) can be written as

$$\begin{bmatrix} S_{n-k+1}(\hat{f}) & S_{m-k+1}(\hat{g}) \end{bmatrix} \begin{bmatrix} \mathbf{v}_k \\ -\mathbf{u}_k \end{bmatrix} = S_k \begin{bmatrix} \mathbf{v}_k \\ -\mathbf{u}_k \end{bmatrix} = 0, \qquad k = 1, \ldots, \hat{d}. \tag{4.6}$$

The matrix $S_k = S_k(\hat{f}, \hat{g}) \in \mathbb{R}^{(m+n-k+1)\times(m+n-2k+2)}$ is the kth subresultant matrix, which is formed by deleting the last $(k-1)$ rows of $S(\hat{f}, \hat{g})$, where $S(\hat{f}, \hat{g})$ is defined in (4.2), the last $k-1$ columns of the coefficients of $\hat{f}(x)$, and the last $k-1$ columns of the coefficients of $\hat{g}(x)$. It is clear that the index k ranges from 1 to $\min(m, n)$, and $S_1(\hat{f}, \hat{g}) = S(\hat{f}, \hat{g})$, that is, the condition $k = 1$ yields the Sylvester resultant matrix.

The next section considers the uses of the subresultant matrices $S_k(\hat{f}, \hat{g})$ for the determination of the degree of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$.

4.1.1 Subresultant matrices

The following theorem shows that the subresultant matrices $S_k(\hat{f}, \hat{g})$ can be used to determine the degree of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$.

Theorem 4.1.1. *A necessary and sufficient condition for the polynomials $\hat{f}(x)$ and $\hat{g}(x)$ to have a common divisor of degree $k \geq 1$ is that the rank of the matrix $S_k(\hat{f}, \hat{g})$ is less than or equal to $m + n - 2k + 1$.*

Proof. Since the degree of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$ is $\hat{d}$, this polynomial pair possess common factors of degree $1, 2, \ldots, \hat{d}$, but not a factor of degree $\hat{d} + 1$. The matrix $S_k(\hat{f}, \hat{g})$ is therefore rank deficient and the vectors $\mathbf{u}_k, \mathbf{v}_k$ in (4.6) are non-zero for $k \leq \hat{d}$. For $k > \hat{d}$, however, $S_k(\hat{f}, \hat{g})$ is full rank and the only solution in (4.6) is $\mathbf{u}_k = 0$ and $\mathbf{v}_k = 0$.

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

□

Furthermore, the assumption that $\hat{f}(x)$ and $\hat{g}(x)$ possess a common divisor of degree $k \leq \hat{d}$ implies $d_{k,0} \neq 0$, and thus $u_{k,0}, v_{k,0} \neq 0$. It therefore follows that if

$$S_k = \begin{bmatrix} h_k & H_k \end{bmatrix},$$

where $h_k = h_k(\hat{f}) \in \mathbb{R}^{(m+n-k+1)}$ is the first column of $S_k(\hat{f}, \hat{g})$ and $H_k = H_k(\hat{f}, \hat{g}) \in \mathbb{R}^{(m+n-k+1)\times(m+n-2k+1)}$ is the matrix formed from its other columns, then (4.6) can be written as

$$\begin{bmatrix} h_k & H_k \end{bmatrix} \begin{bmatrix} \mathbf{v}_k \\ -\mathbf{u}_k \end{bmatrix} = \begin{bmatrix} h_k & H_k \end{bmatrix} \begin{bmatrix} -1 \\ x_k \end{bmatrix} = 0$$

where

$$x_k = \begin{bmatrix} v_{k,1} & \cdots & v_{k,n-k} & -u_{k,0} & \cdots & -u_{k,m-k} \end{bmatrix}^T \in \mathbb{R}^{m+n-2k+1},$$

and $v_{k,0}$ can be set equal to -1, that is, a linear algebraic equation can be obtained by moving h_k to the right hand side,

$$\begin{aligned} H_k x_k &= h_k, &\quad \text{for} \quad & k = 1, \ldots, \hat{d}, \\ H_k x_k &\neq h_k, &\quad \text{for} \quad & k = \hat{d}+1, \ldots, \min(m,n). \end{aligned} \tag{4.7}$$

For each value of $k < \hat{d}$, (4.7) possesses an infinite number of solutions, because H_k is rank deficient, but only a finite number of this infinite number of solutions x_k yield polynomials $\mathbf{u}_k$ and $\mathbf{v}_k$ such that

$$c_k(x) = \frac{\hat{f}(x)}{u_k(x)} = \frac{\hat{g}(x)}{v_k(x)}, \tag{4.8}$$

is a polynomial and not a rational function. When $k = \hat{d}$, there is a different situation because (4.7) has unique solution. In this case, since the GCD is unique and condition $v_{k,0} = -1$ has been imposed, the polynomial $c_k(x)$ is equal to the GCD of $\hat{f}(x)$ and $\hat{g}(x)$.

The polynomials $\hat{f}(x)$ and $\hat{g}(x)$ have a finite number of common divisors, defined up to a scalar multiplier, and therefore a finite number of coprime polynomials. An infinite number of vectors, however, defined to within an arbitrary scalar multiplier, lie in the null space of $S(\hat{f},\hat{g})$ [1], and it is therefore instructive to consider the characterisation of the vectors that lie in the null space of $S(\hat{f},\hat{g})$, but do not define coprime polynomials. Example 4.1 shows the solution to this problem and it is shown there is a clear difference in the cases $k < \hat{d}$ and $k = \hat{d}$.

Example 4.1. Consider the polynomials

$$\hat{f}(x) = (x-1)(x-2)(x-3) = x^3 - 6x^2 + 11x - 6,$$

and

$$\hat{g}(x) = (x-1)^2(x-2) = x^3 - 4x^2 + 5x - 2,$$

whose GCD is of degree 2.

The Sylvester matrix $S(\hat{f},\hat{g}) = S_1(\hat{f},\hat{g})$ of $\hat{f}(x)$ and $\hat{g}(x)$,

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ -6 & 1 & 0 & -4 & 1 & 0 \\ 11 & -6 & 1 & 5 & -4 & 1 \\ -6 & 11 & -6 & -2 & 5 & -4 \\ 0 & -6 & 11 & 0 & -2 & 5 \\ 0 & 0 & -6 & 0 & 0 & -2 \end{bmatrix}, \tag{4.9}$$

has rank 4 because the rank loss of $S(\hat{f},\hat{g})$ is equal to the degree of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$ [5]. The family of vectors that lie in the null space of $S(\hat{f},\hat{g})$ is

[1]The dimension of the null space of a matrix A is called the nullity of A. The rank and nullity of a matrix A with n columns are related by the equation: rank(A)+nullity(A)= n.

$$\begin{bmatrix} \mathbf{v}_1 \\ -\mathbf{u}_1 \end{bmatrix} = \begin{bmatrix} v_{1,0} \\ v_{1,1} \\ v_{1,2} \\ -u_{1,0} \\ -u_{1,1} \\ -u_{1,2} \end{bmatrix} = \begin{bmatrix} v_{1,0} \\ v_{1,1} \\ -v_{1,0} - v_{1,1} \\ -v_{1,0} \\ 2v_{1,0} - v_{1,1} \\ 3v_{1,0} + 3v_{1,1} \end{bmatrix},$$

where $v_{1,0}$ and $v_{1,1}$ are non-zero arbitrary parameters, such that

$$\begin{aligned} u_1(x) &= v_{1,0}x^2 - (2v_{1,0} - v_{1,1})x - 3v_{1,0} - 3v_{1,1} = (v_{1,0}x + v_{1,0} + v_{1,1})(x-3) \\ v_1(x) &= v_{1,0}x^2 + v_{1,1}x - v_{1,0} - v_{1,1} = (v_{1,0}x + v_{1,0} + v_{1,1})(x-1), \end{aligned}$$

and thus the common divisors of $\hat{f}(x)$ and $\hat{g}(x)$ are

$$c(x) := \frac{\hat{f}(x)}{u_1(x)} = \frac{\hat{g}(x)}{v_1(x)} = \frac{(x-1)(x-2)}{v_{1,0}x + v_{1,0} + v_{1,1}}. \tag{4.10}$$

The GCD of $\hat{f}(x)$ and $\hat{g}(x)$ is obtained for $v_{1,0} = 0, v_{1,1} \neq 0$, and $c(x)$, which is in general a rational function, is proportional to the linear common divisors $(x-1)$ and $(x-2)$ of $\hat{f}(x)$ and $\hat{g}(x)$ for

$$v_{1,1} = -2v_{1,0} \qquad \text{and} \qquad v_{1,1} = -3v_{1,0}, \tag{4.11}$$

respectively. Other values of $v_{1,0}$ and $v_{1,1}$ yield rational functions $c(x)$, and they are therefore not of interest. It follows that the null space of $S(\hat{f}, \hat{g})$ includes a vector, defined up to an arbitrary scalar multiplier, that defines:

- the GCD of $\hat{f}(x)$ and $\hat{g}(x)$,
- a finite number of vectors, each of which is defined up to an arbitrary scalar multiplier, that represent the coefficients of the common linear divisors of $\hat{f}(x)$ and $\hat{g}(x)$,

- an infinite number of vectors $\mathbf{v}_1$ and $\mathbf{u}_1$ that define polynomials, which lead to rational functions $c(x)$.

The situation is slightly different when (4.7) is considered because the constraint $v_{1,0} = -1$ implies only a subspace of the null space of the Sylvester resultant matrix (4.9) is considered. In particular, it follows immediately from (4.10) that this constraint cannot recover the GCD of $\hat{f}(x)$ and $\hat{g}(x)$, but it follows from (4.11) that it can recover their common linear divisors. It is shown, however, that the GCD of $\hat{f}(x)$ and $\hat{g}(x)$ can be recovered from the subresultant matrix $S_2(f, g)$ if $v_{2,0} = -1$.

If $k = 1$ and, then $S(\hat{f}, \hat{g}) = S_1(\hat{f}, \hat{g})$ and it follows that (4.7) becomes

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & -4 & 1 & 0 \\ -6 & 1 & 5 & -4 & 1 \\ 11 & -6 & -2 & 5 & -4 \\ 6 & 11 & 0 & -2 & 5 \\ 0 & -6 & 0 & 0 & -2 \end{bmatrix} \begin{bmatrix} v_{1,1} \\ v_{1,2} \\ -u_{1,0} \\ -u_{1,1} \\ -u_{1,2} \end{bmatrix} = \begin{bmatrix} 1 \\ -6 \\ 11 \\ -6 \\ 0 \\ 0 \end{bmatrix}. \tag{4.12}$$

where the coefficient matrix H_1 also has rank 4. This equation therefore has an infinite number of solutions,

$$u_{1,0} = -1, \quad u_{1,1} = v_{1,1} + 2, \quad u_{1,2} = 3 - 3v_{1,1}, \quad v_{1,2} = 1 - v_{1,1},$$

where $v_{1,1}$ is arbitrary, and

$$\begin{aligned} u_1(x) &= -x^2 + (2 + v_{1,1})x + 3 - 3v_{1,1} = (-x + v_{1,1} - 1)(x - 3) \\ v_1(x) &= -x^2 + v_{1,1}x + 1 - v_{1,1} = (-x + v_{1,1} - 1)(x - 1). \end{aligned}$$

It follows from (4.8) that the common divisor $c_1(x)$ is

$$c_1(x) = \frac{\hat{f}(x)}{u_1(x)} = \frac{\hat{g}(x)}{v_1(x)} = \frac{(x-1)(x-2)}{-x + v_{1,1} - 1}.$$

It is seen that $c_1(x)$ is, in general, a rational function, and only two values of the arbitrary parameter $v_{1,1}$ yield a polynomial. In particular, the value $v_{1,1} = 2$ yields the common divisor $c_1(x) = -(x-2)$, and the value $v_{1,1} = 3$ yields the common linear divisor $c_1(x) = -(x-1)$. These values of $v_{1,1}$ are, as required, the same values specified in (4.11).

Consider now the situation $k = 2$, in which case (4.7) becomes

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ -6 & 5 & -4 \\ 11 & -2 & 5 \\ 6 & 0 & -2 \end{bmatrix} \begin{bmatrix} v_{2,1} \\ -u_{2,0} \\ -u_{2,1} \end{bmatrix} = \begin{bmatrix} 1 \\ -6 \\ 11 \\ -6 \\ 0 \end{bmatrix}, \tag{4.13}$$

which has the unique solution

$$v_{2,1} = 1, \quad u_{2,0} = -1, \quad u_{2,1} = 3.$$

Since $v_{2,0} = -1$, it follows that

$$u_2(x) = -(x-3) \qquad \text{and} \qquad v_2(x) = -(x-1)$$

which are coprime, and (4.8) shows that the common divisor is

$$c_2(x) = -(x-1)(x-2).$$

Consider now the situation $k = 3$, in which case (4.7) becomes

$$\begin{bmatrix} 1 \\ -4 \\ 5 \\ -2 \end{bmatrix} (-u_{3,0}) = \begin{bmatrix} 1 \\ -6 \\ 11 \\ -6 \end{bmatrix},$$

Since this equation does not possess a solution, the polynomials $\hat{f}(x)$ and $\hat{g}(x)$ do not have a common divisor of degree $k = 3$, and thus the degree of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$ is two. $\square$

The next section considers the definition of the Bézout resultant matrix.

4.2 The Bézout resultant matrix

The Sylvester resultant matrix was introduced in Section 4.1, and some of its properties were described. This section considers the Bézout resultant matrix, which is another resultant matrix that will be used by the principle of maximum likelihood for the calculation of the degree of an approximate GCD of two polynomials in Section 5.1.

It is shown in [5], pages 44 – 45, that the element $\beta_{i,j}$ of the Bézout resultant matrix $B(\hat{f},\hat{g}) \in \mathbb{R}^{c\times c}, c = \max(m,n)$ is

$$\begin{aligned}\beta_{i,j} &= |\hat{a}_{n-i-j+1},\hat{b}_n| + |\hat{a}_{n-i-j+2},\hat{b}_{n-1}| + \cdots + \\ &\quad |\hat{a}_{n-i-j+k+1},\hat{b}_{n-k}| \\ &= \sum_{k=0}^{\min(i-1,j-1)} |\hat{a}_{n-i-j+k+1},\hat{b}_{n-k}|, \qquad i,j = 1,\ldots,c, \end{aligned} \tag{4.14}$$

where $|\hat{a}_i,\hat{b}_j| = \hat{a}_i\hat{b}_j - \hat{a}_j\hat{b}_i$. In particular, every element of $B(\hat{f},\hat{g})$ is a bilinear function of the coefficients $\hat{a}_i$ of $\hat{f}(x)$ and $\hat{b}_i$ of $\hat{g}(x)$. The matrix $B(\hat{f},\hat{g})$ is of order $c\times c$, and if $m > n$ then $\hat{g}(x)$ is padded with $m - n$ zeros, and similarly if $n > m$.

Compared with the matrix $S(\hat{f},\hat{g})$ that is strictly linear and partitioned, the matrix $B(\hat{f},\hat{g})$ is bilinear, and thus it arises

$$B(\alpha\hat{f},\beta\hat{g}) = \alpha\beta B(\hat{f},\hat{g}), \quad \alpha,\beta \in \mathbb{R}\setminus 0. \tag{4.15}$$

Moreover, $B(\hat{f},\hat{g})$ plays an important role in many fields of symbolic and numerical computing, including signal processing and control theory [4, 18, 27].

The Sylvester and Bézout resultant matrices are introduced above, and some of their properties are therefore considered in the next section.

4.3 The rank of a resultant matrix

The following property makes $B(\hat{f}, \hat{g})$ and $S(\hat{f}, \hat{g})$ attractive for performing computations on rank estimation [5]:

- The rank loss of $B(\hat{f}, \hat{g})$ and $S(\hat{f}, \hat{g})$ is equal to the degree of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$.

This property for the exact polynomial pair $\hat{f}(x)$ and $\hat{g}(x)$ is extended to the inexact polynomial pair $f(x)$ and $g(x)$ by assuming either that the numerical rank of $B(f, g)$ and $S(f, g)$ is defined, or that the noise level is known, such that a threshold can be placed on the small singular values of these noisy matrices.

The determination of the rank of a noisy matrix is a challenging problem that arises in many computational fields of science. Although the SVD of a resultant matrix is frequently used to determine the numerical rank of a matrix [12] [16] [42] [72], it suffers from disadvantages. In particular, the presence of roundoff error due to finite precision arithmetic may suggest that a matrix is non-singular even if this matrix is theoretically singular. The following example shows that the numerical rank of $B(\hat{f}, \hat{g})$ and $S(\hat{f}, \hat{g})$ may not be defined, even if only roundoff errors are present and the exact polynomial pair $\hat{f}(x)$ and $\hat{g}(x)$ are used.

Example 4.2. Consider the exact polynomial pair

$$\begin{aligned}\hat{f}(x) &= (x-1)^2(x-2)^3(x-5)\\ \hat{g}(x) &= (x-1)(x-2)^2(x-6)^9\end{aligned}$$

whose GCD is of degree 3.

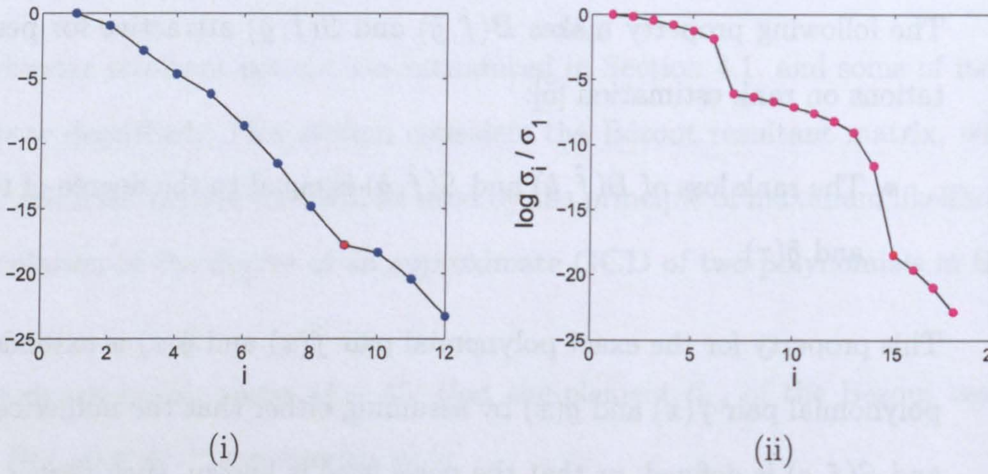


Figure 4.1: The normalised singular values of (i) the Bézout resultant matrix $B(\hat{f},\hat{g})$, and (ii) the Sylvester resultant matrix $S(\hat{f},\hat{g})$, in the absence of noise.

Figure 4.1 shows the normalised singular values σ_i/σ_1 of $B(\hat{f},\hat{g})$, and the normalised singular values of $S(\hat{f},\hat{g})$, in the absence of noise. Figure 4.1(i) shows that the rank of $B(\hat{f},\hat{g})$ is not defined, and Figure 4.1(ii) suggests that the degree of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$, computed from $S(\hat{f},\hat{g})$, is 4 rather than the correct answer of 3 because

$$\max_{i=1,\ldots,17}\left\{\frac{\sigma_i}{\sigma_{i+1}}\right\}=\frac{\sigma_{14}}{\sigma_{15}}.$$

These computations are performed in the absence of data errors, that is, only roundoff errors are considered, and thus the results for inexact polynomial pair must necessarily be inferior. $\square$

This example and the preceding discussion show that the SVD of $B(\hat{f},\hat{g})$ and $S(\hat{f},\hat{g})$ cannot be used to estimate the degree of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$, and this disadvantage of the SVD is more apparent when data errors are present because the data errors are usually much larger than roundoff errors. Furthermore, Triantafyllou and Mitrouli [61] point out that

> "the roundoff errors during the numerical implementation of the algorithms ... may lead to serious problem for the computation of the rank of the Sylvester matrix."

Alternative methods have been proposed, such as the rank revealing QR decomposition [9], the rank revealing LU decomposition [48] and a new rank revealing algorithm [41, 43].

One problem with the vast majority of methods is that a threshold, as a function of the noise level, is required to be manually set in order to determine the index of the smallest singular value that defines the numerical rank. This is a problem because the noise level may not be known, or it may only be known approximately. The following example shows that the numerical rank of $S(f,g)$ of the inexact polynomial pair $f(x)$ and $g(x)$ can be defined with a *priori* knowledge of the noise level that is imposed.

Example 4.3. Consider the polynomial pair $\hat{f}(x)$ and $\hat{g}(x)$, from Example 4.2, and introduce a componentwise error model, which is defined in (2.3) and (2.4), to the coefficients of $\hat{f}(x)$ and $\hat{g}(x)$ with a signal to noise ratio $\varepsilon_c^{-1} = 10^9$, that is, the exact polynomial pair change to the noisy form of $f = f(x)$ and $g = g(x)$.

The Sylvester resultant matrix, $S(f,g) \in \mathbb{R}^{18\times 18}$, is now constructed from the perturbed polynomial pair, and the **rank** function in MATLAB is called. If the default tolerance is used, then MATLAB returns a rank of 14. If the tolerance is manually set to 10^{-10}, then the calculated rank is 16. The correct rank of 15 can be obtained when the tolerance is set equal to ε_c as 10^{-9}. It is clear that setting this threshold may be problematic if the signal-to-noise ratio is not known exactly. $\square$

4.4 Preprocessing operations

It has been shown that the subresultant matrices $S_k(f, \hat{g})$ can be applied to determine the degree of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$. This is, however, not the case when computations are performed in a floating point environment, especially for the inexact polynomials $f(x)$ and $g(x)$. The matrix $S_k(f, g)$ is reduced to a matrix of full rank because $f(x)$ and $g(x)$ are coprime. Three preprocessing operations are therefore considered in this section for the improvement of computational results on rank estimation, and two of them originate from the partitioned nature of the Sylvester matrix, and one of them originates from the difficulty of performing reliable computations on polynomials whose coefficients vary widely in magnitude.

In [22] Ghaderpanah and Klasa remark that:

> "A wide variation in the magnitude of coefficients of polynomials may be a source of computational problem in root-finding algorithm, as the floating point arithmetic operations on such coefficients may render floating point overflow or underflow,"

with particular emphasis in the application of root-finding methods:

> "particularly those involving calculation of the greatest common divisors of two polynomials."

It is therefore important that the inexact polynomial pair need to be processed before an approximate GCD is computed, and it is necessary to distinguish between exact and inexact polynomials. Thus, $f(x)$ and $g(x)$ denote the inexact forms of the theoretically exact polynomials $\hat{f}(x)$ and $\hat{g}(x)$, respectively, which are defined as

$$f(x) = \sum_{i=0}^{m} a_i x^{m-i} \qquad \text{and} \qquad g(x) = \sum_{i=0}^{n} b_i x^{n-i}, \tag{4.16}$$

where $a_0, b_0 \neq 0$.

The matrix $S_k(f,g)$ has a partitioned structure because the coefficients of $f(x)$ occupy its first $n-k+1$ columns, and the coefficients of $g(x)$ occupy its last $m-k+1$ columns. It may therefore yield an unbalanced matrix $S_k(f,g)$ if the coefficients of $f(x)$ are much smaller or larger than the coefficients of $g(x)$. For example, if $|a_i| \gg |b_j|, i = 0,\ldots,m, j = 0,\ldots,n$, then the rank of $S(f,g)$ is approximately equal to n even if $f(x)$ and $g(x)$ are coprime, and similarly, if $|a_i| \ll |b_j|$, then the rank of $S(f,g)$ is approximate equal to m. These are the two extreme conditions, but they illustrate the problems that can occur if precautions are not taken. It is therefore necessary to preprocess the polynomials instead of the matrix in order to preserve the structure of the matrix.

The first preprocessing operation therefore involves normalising the coefficients of $f(x)$ and $g(x)$ by the geometric mean of their coefficients, such that $S_k(f,g)$ is better balanced. The 2-norm of the coefficients of a polynomial is frequently applied for normalisation because it yields a matrix that is better conditioned [8] [13]. It has been established computationally, however, that it is advantageous to normalise the coefficients of a polynomial by the geometric mean of its coefficients because it provides a 'better average' when the coefficients of a polynomial vary widely in magnitude. Thus $\bar{f}(x)$ and $\bar{g}(x)$ are scaled from $f(x)$ and $g(x)$ by the geometric means of their coefficients, and are therefore given by

$$\bar{f}(x) = \sum_{i=0}^{m} \bar{a}_i x^{m-i}, \qquad \bar{a}_i = \frac{a_i}{\left(\prod_{i=0}^{m} |a_i|\right)^{\frac{1}{m+1}}} \tag{4.17}$$

and

$$\bar{g}(x) = \sum_{i=0}^{n} \bar{b}_i x^{n-i}, \qquad \bar{b}_i = \frac{b_i}{\left(\prod_{i=0}^{n} |b_i|\right)^{\frac{1}{n+1}}} \tag{4.18}$$

where a_i and b_i are the non-normalised coefficients of $f(x)$ and $g(x)$ respectively, and it is assumed they are non-zero. If, however, one or more of these coefficients are zero, then the geometric mean is computed with respect to the non-zero coefficients only, and not all the coefficients.

Example 4.4. Consider the exact polynomial pair

$$\begin{aligned}\hat{f}(x) &= (x+0.5161)^5(x+7.1052)^5(x+0.1132)^3\\ \hat{g}(x) &= (x+0.5161)^5(x+7.1052)^5(x+8.8614)^7(x-2.0476)^7\end{aligned}$$

whose GCD is of degree 10. A componentwise error is applied to the coefficients of $\hat{f}(x)$ and $\hat{g}(x)$ with signal-to-noise ratio $\varepsilon_c^{-1} = 10^8$, that is, the exact polynomial pair change to the noisy forms $f(x)$ and $g(x)$.

The importance of normalisation by the geometric mean is shown in Figure 4.2, and it is seen that the matrix $S(\bar{f}, \bar{g})$ can give a better estimate of the degree of an approximate GCD of $f(x)$ and $g(x)$, compared to the matrix $S(f, g)$, because

$$\begin{array}{llllll}\text{rank } S(f,g) = 13 & \Rightarrow & \deg \text{GCD}(f,g) = 24 & \Rightarrow & \text{incorrect,}\\ \text{rank } S(\bar{f},\bar{g}) = 27 & \Rightarrow & \deg \text{GCD}(\bar{f},\bar{g}) = 10 & \Rightarrow & \text{correct.}\end{array}$$

□

Consider now the second preprocessing operation.

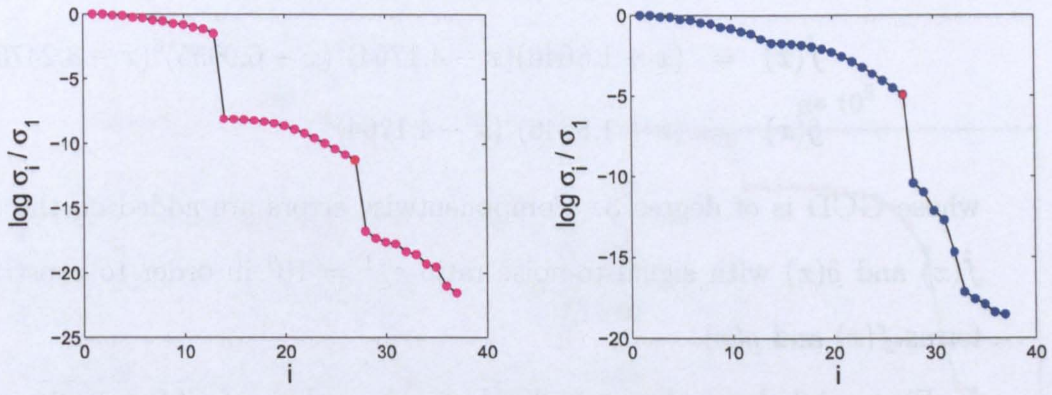


Figure 4.2: The normalised singular values of (i)$S(f,g)$, (ii) $S(\bar{f},\bar{g})$, with $\varepsilon_c = 10^{-8}$.

Since

$$\deg \text{ GCD } (f,g) = \deg \text{ GCD } (f,\alpha g),$$

where α is an arbitrary non-zero constant, it follows that

$$\text{rank loss } S(f,g) = \text{rank loss } S(f,\alpha g),$$

and thus the polynomial $g(x)$ can be generalised to $\alpha g(x)$, where α can be used to achieve optimal results using a specified criterion. This criterion and the method used to calculate the optimal value will be addressed after the third preprocessing operation has been discussed. It is therefore better to use the Sylvester matrix $S(f,\alpha g)$ rather than $S(f,g)$. It would seem intuitive that this scale factor α has no obvious effect on the determination of the rank of $S_k(f,\alpha g)$. The following example demonstrates, however, that this is not the case.

Example 4.5. Consider the exact polynomial pair

$$\begin{aligned}\hat{f}(x) &= (x+1.8646)(x-4.1764)^2(x-6.9955)^6(x-8.2475)^6\\ \hat{g}(x) &= (x+1.8646)^4(x-4.1764)^2\end{aligned}$$

whose GCD is of degree 3. Componentwise errors are added on the coefficients of $\hat{f}(x)$ and $\hat{g}(x)$ with signal-to-noise ratio $\varepsilon_c^{-1} = 10^9$ in order to construct the noisy forms $f(x)$ and $g(x)$.

Figure 4.3 shows the normalised singular values of $S(f, \alpha g)$ for six values of α. Since $\deg f(x) = 15$ and $\deg g(x) = 6$, the coefficients of $f(x)$ occupy the first 6 columns of $S(f, \alpha g)$, and the coefficients of $\alpha g(x)$ occupy the last 15 columns. It is shown that rank $S(f, \alpha g) \approx 6 = \deg g(x)$ when $\alpha = 1$, and that rank $S(f, \alpha g) \approx 15 = \deg f(x)$ when $\alpha = 10^{16}$. Also, it can be seen clearly that the matrix $S(f, \alpha g)$ appears more and more rank deficient as $\alpha \rightarrow 10^{10}$, in particular, rank loss $S(f, \alpha g) \rightarrow 3 = \deg \operatorname{GCD}(f, g)$. It is however reasonable because

$$\frac{\max|\text{coefficients of } f(x)|}{\max|\text{coefficients of } g(x)|} \approx 10^{10},$$

and thus $\alpha \approx 10^{10}$ 'balances' the Sylvester resultant matrix. □

Consider now the third preprocessing operation.

It is known that computations performed on polynomials whose coefficients have a wide variation in magnitude are unreliable [14] [22], and it is therefore necessary to minimise this variation. This is achieved by the substitution

$$x = \theta y, \tag{4.19}$$

in (4.17) and (4.18), where θ is a parameter whose optimal value is to be determined and y is the new independent variable. The polynomials $\bar{f}(x)$ and $\bar{g}(x)$ are therefore

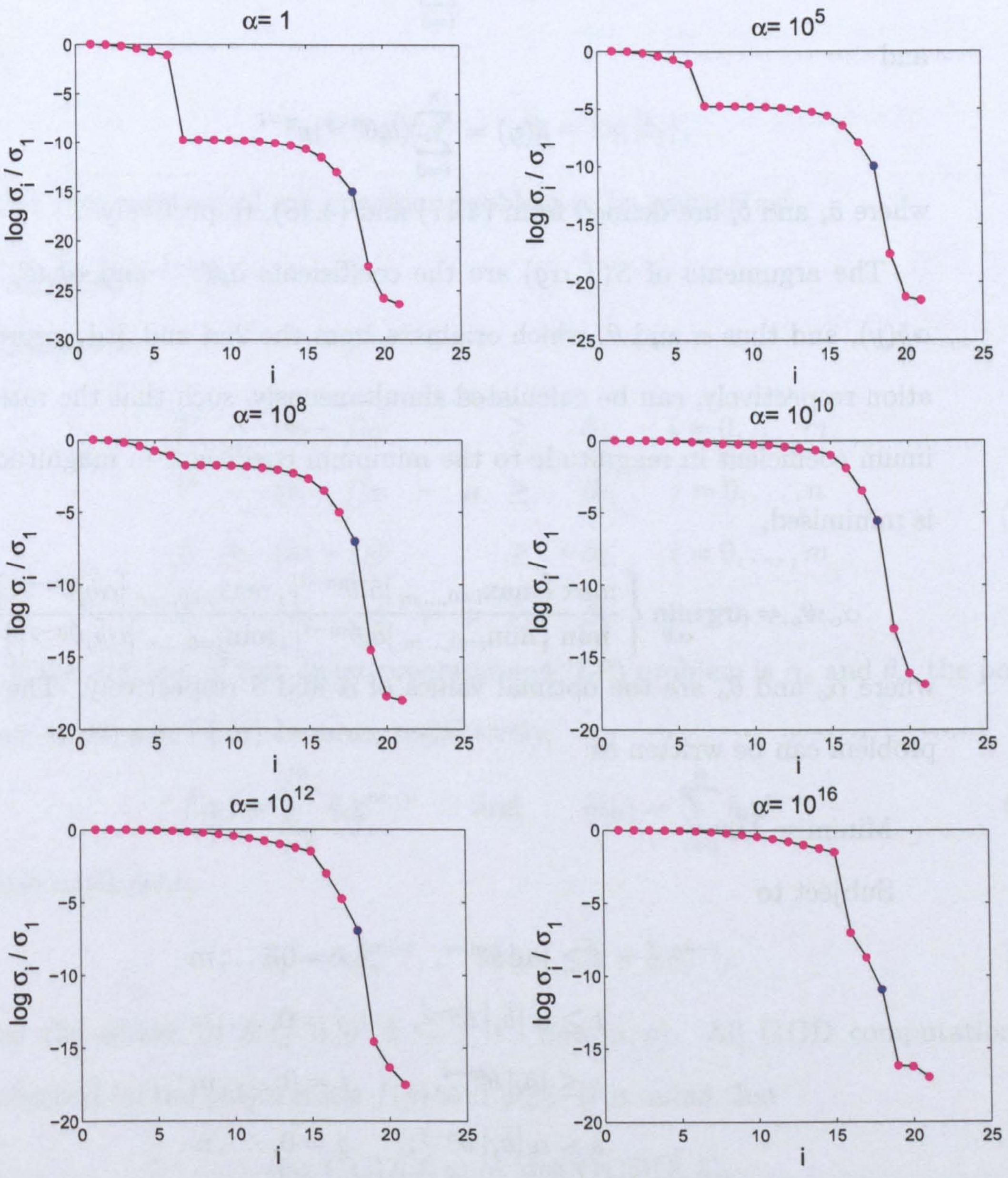


Figure 4.3: The effect of α on the normalised singular values of $S(f, \alpha g)$, with $\varepsilon_c = 10^{-9}$.

transformed to the polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$ respectively,

$$\tilde{f}(y) = \sum_{i=0}^{m} (\bar{a}_i \theta^{m-i}) y^{m-i}, \tag{4.20}$$

and

$$\tilde{g}(y) = \sum_{i=0}^{n} (\bar{b}_i \theta^{n-i}) y^{n-i}, \tag{4.21}$$

where $\bar{a}_i$ and $\bar{b}_i$ are defined from (4.17) and (4.18), respectively.

The arguments of $S(\tilde{f}, \alpha\tilde{g})$ are the coefficients $\bar{a}_i\theta^{m-i}$ and $\alpha\bar{b}_i\theta^{n-i}$ of $\tilde{f}(y)$ and $\alpha\tilde{g}(y)$, and thus α and θ, which originate from the 2nd and 3rd preprocessing operation respectively, can be calculated simultaneously, such that the ratio of the maximum coefficient in magnitude to the minimum coefficient in magnitude of $S(\tilde{f}, \alpha\tilde{g})$ is minimised,

$$\alpha_o, \theta_o = \arg\min_{\alpha,\theta} \left\{ \frac{\max\left\{\max_{i=0,\ldots,m} \left|\bar{a}_i\theta^{m-i}\right|, \max_{j=0,\ldots,n} \left|\alpha\bar{b}_j\theta^{n-j}\right|\right\}}{\min\left\{\min_{i=0,\ldots,m} \left|\bar{a}_i\theta^{m-i}\right|, \min_{j=0,\ldots,n} \left|\alpha\bar{b}_j\theta^{n-j}\right|\right\}} \right\}, \tag{4.22}$$

where α_o and θ_o are the optimal values of α and θ respectively. The minimisation problem can be written as:

Minimise t/s

Subject to

$$\begin{array}{ll}
t \geq |\bar{a}_i|\,\theta^{m-i}, & i = 0,\ldots,m \\
t \geq \alpha\left|\bar{b}_j\right|\theta^{n-j}, & j = 0,\ldots,n \\
s \leq |\bar{a}_i|\,\theta^{m-i}, & i = 0,\ldots,m \\
s \leq \alpha\left|\bar{b}_j\right|\theta^{n-j}, & j = 0,\ldots,n \\
s > 0 & \\
\theta > 0 & \\
\alpha > 0 & .
\end{array}$$

The transformations

$$T = \log t, \qquad S = \log s, \qquad \phi = \log \theta, \qquad \mu = \log \alpha$$

and

$$\bar{\alpha}_i = \log |\bar{a}_i|, \qquad \bar{\beta}_j = \log |\bar{b}_j|,$$

enable this constrained minimisation problem to be written as:

Minimise $T - S$

Subject to

$$\begin{array}{rcrcrcrl} T & - & (m-i)\phi & & & \geq & \bar{\alpha}_i, & i = 0, \ldots, m \\ T & - & (n-j)\phi & - & \mu & \geq & \bar{\beta}_j, & j = 0, \ldots, n \\ -S & + & (m-i)\phi & & & \geq & -\bar{\alpha}_i, & i = 0, \ldots, m \\ -S & + & (n-j)\phi & + & \mu & \geq & -\bar{\beta}_j, & j = 0, \ldots, n. \end{array} \tag{4.23}$$

If the solution of this linear programming (LP) problem is α_o and θ_o, the polynomials (4.20) and (4.21) become, respectively,

$$\tilde{f}(y) = \sum_{i=0}^{m} \tilde{a}_i y^{m-i} \qquad \text{and} \qquad \tilde{g}(y) = \sum_{i=0}^{n} \tilde{b}_i y^{n-i}, \tag{4.24}$$

whose coefficients

$$\tilde{a}_i = \bar{a}_i \theta_o^{m-i} \qquad \text{and} \qquad \tilde{b}_i = \bar{b}_i \theta_o^{n-i}, \tag{4.25}$$

form the entries of $S_k(\tilde{f}, \alpha_o \tilde{g}), k = 1, \ldots, \min(m,n)$. All GCD computations are performed on the polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$. It is noted that

$$\deg \text{ GCD}(f,g) = \deg \text{ GCD}(\tilde{f}, \tilde{g}),$$

and thus

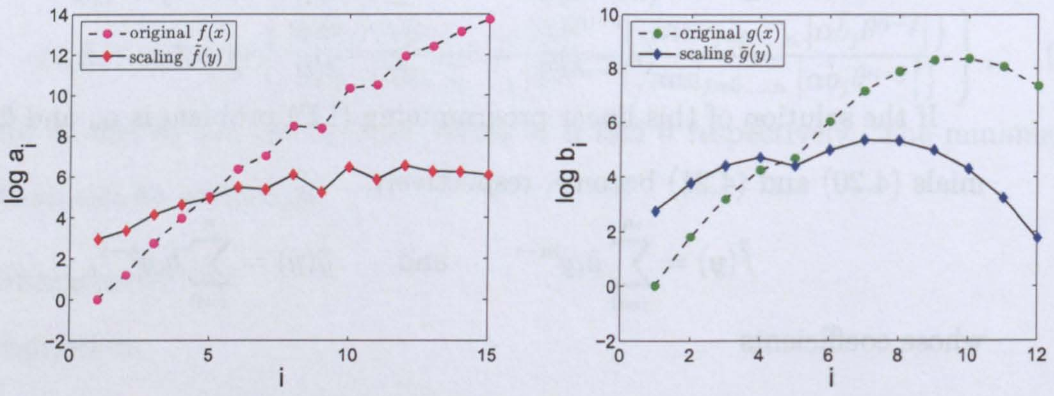
$$\text{rank } S(f,g) = \text{rank } S(\tilde{f}, \tilde{g}).$$

The importance of polynomial scaling by α_o and θ_o is shown in Example 4.6, and it is also shown that failure to implement this substitution (4.19) may cause incorrect results to be obtained.

Example 4.6. Consider the exact polynomial pair

$$\begin{aligned}\hat{f}(x) &= (x-9.3722)^8(x+9.9450)^6\\ \hat{g}(x) &= (x-9.3722)(x+9.9450)^7(x+0.6239)^3\end{aligned}$$

whose GCD is of degree 7. The coefficients of $\hat{f}(x)$ and $\hat{g}(x)$ are perturbed by componentwise errors with signal-to-noise ratio $\varepsilon_c^{-1} = 10^7$ in order to construct the noisy forms $f(x)$ and $g(x)$.



(i) (ii)

Figure 4.4: The coefficients of (i) $f(x)$ and $\bar{f}(y)$, and (ii) $g(x)$ and $\bar{g}(y)$, with $\varepsilon_c = 10^{-7}$.

First, $f(x)$ and $g(x)$ are normalised by their geometric means using (4.17) and (4.18), respectively, in order to obtain $\bar{f}(x)$ and $\bar{g}(x)$. Then the optimal values α_o and θ_o of α and θ, respectively, are calculated from the LP problem, such that the reduction in the magnitude of the coefficients of $\bar{f}(x)$ and $\bar{g}(x)$ arises from the substitution (4.19). Hence $\tilde{f}(y)$ and $\tilde{g}(y)$ are computed from (4.24), and their coefficients are shown in Figure 4.4, compared with the coefficients of the original polynomial pair $f(x)$ and $g(x)$.

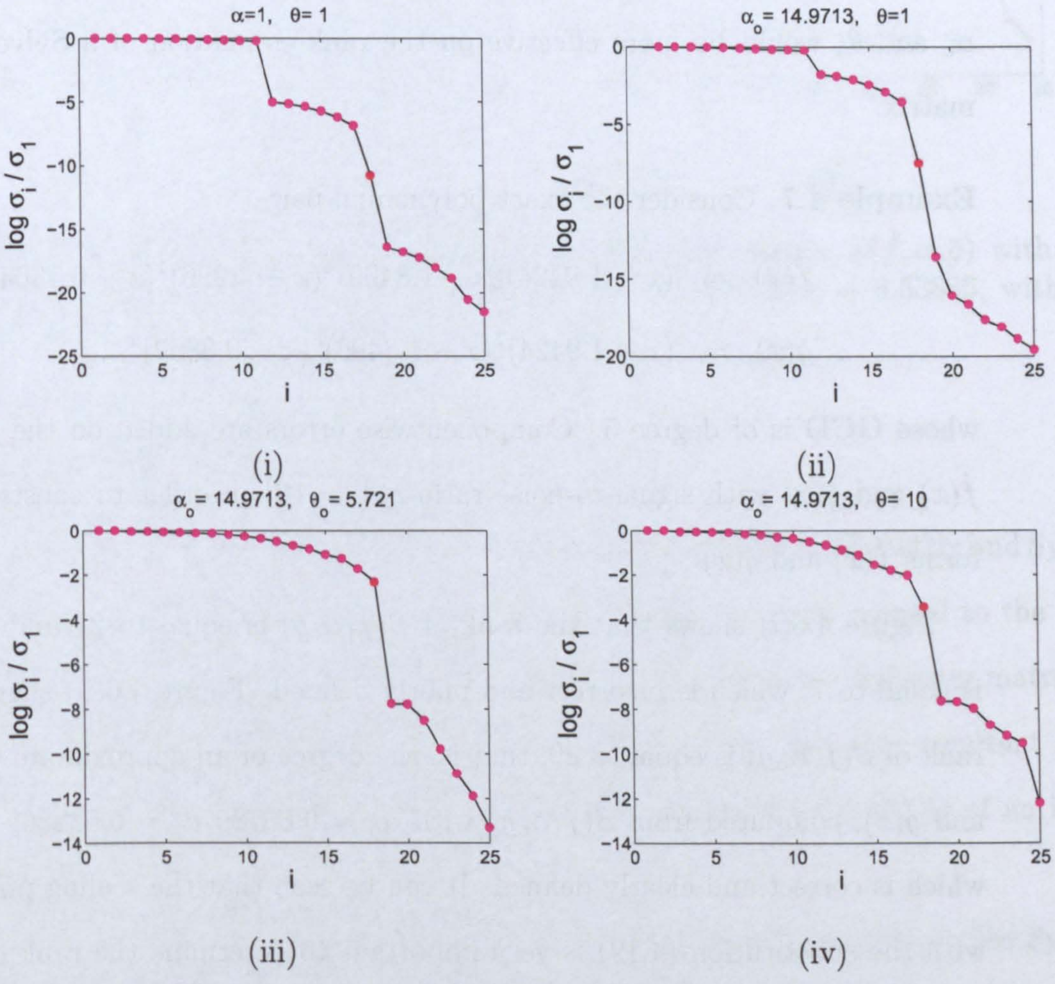


Figure 4.5: The normalised singular values of (i) the matrix $S(f,g)$, (ii) the matrix $S(\bar{f},\alpha_o\bar{g})$ with $\alpha_o = 14.971$, (iii) the matrix $S(\tilde{f},\alpha_o\tilde{g})$ with $\alpha_o = 14.9713, \theta_o = 5.721$ and (iv) the matrix $S(\tilde{f},\alpha_o\tilde{g})$ with $\alpha_o = 14.9713, \theta = 10$, with $\varepsilon_c = 10^{-7}$.

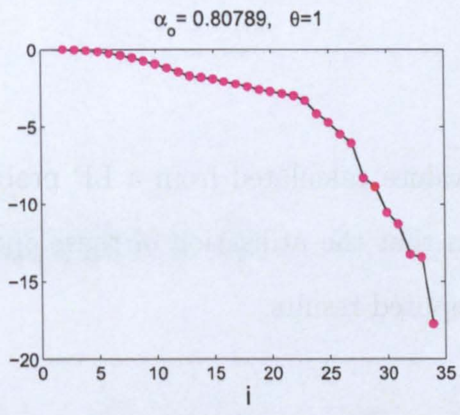
It can be seen from Figure 4.4 that the variation in the magnitude of the coefficients of $f(x)$ and $g(x)$ becomes small after scaling of this polynomial pair. Figure 4.5 shows the normalised singular values of the different matrices produced as θ is varied. It is clear that Figure 4.5(iii) returns the best and correct answer rather than the other three figures, and it seems that polynomial scaling with optimal values of α_o and θ_o would be more effective on the rank estimation of a Sylvester resultant matrix. □

Example 4.7. Consider the exact polynomial pair

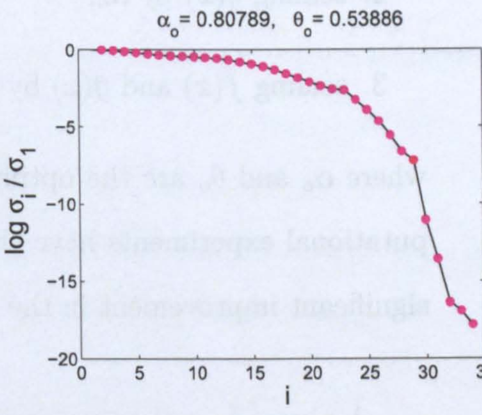
$$\begin{aligned} \hat{f}(x) &= (x+1.9424)(x-1.8499)^4(x-4.996)^4(x-0.1004)^9 \\ \hat{g}(x) &= (x+1.9424)^7(x-1.8499)^5(x-0.3862)^4 \end{aligned}$$

whose GCD is of degree 5. Componentwise errors are added on the coefficients of $\hat{f}(x)$ and $\hat{g}(x)$ with signal-to-noise ratio $\varepsilon_c^{-1} = 10^7$ in order to construct the noisy forms $f(x)$ and $g(x)$.

Figure 4.6(i) shows that the rank of $S(\bar{f}, \alpha_o \bar{g})$ is equal to 27 and the rank loss is equal to 7, which is incorrect and poorly defined. Figure 4.6(ii) suggests that the rank of $S(\tilde{f}, \alpha_o \tilde{g})$ is equal to 29, that is, the degree of an approximate GCD of $f(x)$ and $g(x)$, computed from $S(\tilde{f}, \alpha_o \tilde{g})$ with $\alpha_o = 0.80789, \theta_o = 0.53886$, is equal to 5, which is correct and clearly defined. It can be seen that the scaling polynomial pair with the substitution (4.19) is very important to determine the rank of a Sylvester resultant matrix. □



(i)



(ii)

Figure 4.6: The normalised singular values of (i) the matrix $S(\bar{f}, \alpha_o \bar{g})$ with $\alpha_o = 0.80789$, and (ii) the matrix $S(\tilde{f}, \alpha_o \tilde{g})$ with $\alpha_o = 0.80789, \theta_o = 0.53886$, with $\varepsilon_c = 10^{-7}$.

4.5 Summary

This chapter has reviewed some properties of the Bézout resultant matrix and Sylvester resultant matrix, in which the rank loss of a resultant matrix is equal to the degree of the GCD of the polynomial. The subresultant matrices of a Sylvester matrix that are obtained by deleting some rows and columns of the Sylvester resultant matrix are important when it is required to calculate an approximate GCD of an inexact polynomial pair.

Three preprocessing operations have been considered to perform on the Sylvester resultant matrix and its subresultant matrices in order to improve the rank estimation problem. If $f(x)$ and $g(x)$ are the given inexact polynomials, then

1. normalising $f(x)$ and $g(x)$ by the geometric means in order to obtain $\bar{f}(x)$ and $\bar{g}(x)$,

2. scaling $\bar{g}(x)$ by α_o,

3. scaling $\bar{f}(x)$ and $\bar{g}(x)$ by θ_o.

where α_o and θ_o are the optimal values calculated from a LP problem. Also, computational experiments have shown that the utilisation of these operations causes a significant improvement in the computed results.

Chapter 5

The degree of an approximate GCD, Part I

It has been shown in Chapter 3 that the crucial part of the polynomial root solver is the calculation of an approximate GCD of a noisy polynomial pair. The most difficult part of the calculation of an approximate GCD is the calculation of its degree because this is a non-trivial problem that reduces to the estimation of the rank loss of a resultant matrix, the entries of which are functions of the coefficients of this polynomial pair. It has been stated in Chapter 4 that the determination of the rank of a resultant matrix in the presence of noise is a challenging problem, in which the computation is usually performed by placing a threshold on the small singular value of the matrix. It suffers, however, from disadvantages because the numerical rank of the matrix may not be defined, or the noisy level may not be known, or it may only be known approximately. Moreover, Examples 4.2 and 4.3 show the limitations of a standard rank estimate method, and they provide the motivation for this chapter and the next chapter.

It is therefore necessary to develop some new methods that do not require the knowledge of the noise level, and are performed directly on the coefficients of a noisy polynomial pair. This means that these methods are entirely data driven, such that thresholds, parameters and constraints are not required. Three methods for the calculation of the degree of an approximate GCD are described theoretically and compared computationally in this chapter.

- **Method 1: The principle of maximum likelihood (ML).** Probability distributions are assigned to the non-zero and zero singular values of the Bézout resultant matrix $B(f, g)$, which enables a likelihood expression $L(r)$ of the singular values, as a function of the assumed rank r, to be developed, where the first r singular values are assigned a distribution that is appropriate for the non-zero singular values, and the other singular values are assigned a distribution that is appropriate for the zero singular values. The value of r_* that maximises $L(r)$ enables the degree of an approximate GCD of $f(x)$ and $g(x)$ to be calculated.

- **Method 2: The angle between subspaces.** The Sylvester resultant matrix $S(f, g)$ has a partitioned structure, and this enables two subspaces to be defined. The angle between these subspaces changes significantly from the kth subresultant matrix $S_k(f, g)$ to the $(k+1)$th subresultant matrix $S_{k+1}(f, g)$, where the integer k is the degree of an approximate GCD of $f(x)$ and $g(x)$. The compelling advantage of this method is that additional assumptions are not required.

- **Method 3: The residual of approximate linear algebraic equation.** The error between two estimates of an approximate common divisor of $f(x)$

and $g(x)$, as a function of its degree k, has a minimum at the degree of an approximate GCD of $f(x)$ and $g(x)$. This method also does not require any additional assumptions.

5.1 Method 1: The principle of maximum likelihood

The principle of maximum likelihood (ML) can be used to estimate the rank of the Bézout resultant matrix $B(f, g)$, where the elements of $B(f, g)$ are defined in (4.14), and $f(x)$ and $g(x)$ are defined in (4.16). The ML estimate of the rank of the matrix $B(f, g)$ is the value of the rank that maximises the likelihood expression, which can be derived from Zarowski [68].

Consider a matrix $B(f, g) \in \mathbb{R}^{c \times c}$ of rank $r \leq c$ whose theoretically exact singular values are

$$\hat{\sigma}_i = \begin{cases} \hat{\sigma}_1 \geq \hat{\sigma}_2 \geq \cdots \geq \hat{\sigma}_r > 0 \\ \hat{\sigma}_{r+1} = \hat{\sigma}_{r+2} = \cdots = \hat{\sigma}_c = 0. \end{cases}$$

In many practical problems, the singular values are known approximately and not exactly, in which case only estimates σ_i of the exact singular values $\hat{\sigma}_i$ are available,

$$\sigma_i = \begin{cases} \hat{\sigma}_i + e_i & i = 1, \ldots, r \\ e_i & i = r+1, \ldots, c. \end{cases} \tag{5.1}$$

It is assumed that the errors e_i are statistically independent random variables with Gaussian and exponential probability distributions,

$$p(e_i) = \begin{cases} \frac{1}{\sqrt{2\pi}s} \exp\left(-\frac{e_i^2}{2s^2}\right) & i = 1, \ldots, r \\ \beta \exp(-\beta e_i) & i = r+1, \ldots, c, \end{cases} \tag{5.2}$$

where $s, \beta > 0$. The model is used because it enables considerable analytical progress to be made, and in particular, it provides a trade-off between a physically accurate model and a mathematically simple model.

It follows from (5.2) that assuming all the random variables are independent, the joint probability density function of the random variables e_i is

$$p(\sigma) = \frac{\beta^{c-r}}{(2\pi s^2)^{\frac{r}{2}}} \exp\left(-\frac{1}{2s^2}\sum_{i=1}^{r} e_i^2 - \beta \sum_{i=r+1}^{c} e_i\right),$$

and the substitution of e_i, from (5.1), into this expression yields the probability density function for the estimates σ_i of the exact singular values $\hat{\sigma}_i$,

$$p(\sigma) = \frac{\beta^{c-r}}{(2\pi s^2)^{\frac{r}{2}}} \exp\left(-\frac{1}{2s^2}\sum_{i=1}^{r}(\sigma_i - \hat{\sigma}_i)^2 - \beta \sum_{i=r+1}^{c} \sigma_i\right). \tag{5.3}$$

The ML estimate $\hat{\beta}$ of β is obtained by setting the partial derivative of $\log p(\sigma)$ with respect to β equal to zero, which yields

$$\hat{\beta} = \frac{c-r}{\sum_{i=r+1}^{c} \sigma_i}.$$

It follows from (5.3) that the ML estimate $\hat{s}^2$ of s^2 satisfies

$$\sum_{i=1}^{r} (\sigma_i - \hat{\sigma}_i)^2 = r\hat{s}^2.$$

From (5.3) an expression for the logarithm of the likelihood function, which allows the rank r of $B(f, g)$ to be obtained, is

$$L(r) \;=\; (c-r)\ln\hat{\beta} - \frac{r}{2}\ln\left(2\pi\hat{s}^2\right) - \left(\frac{1}{2\hat{s}^2}\sum_{i=1}^{r}(\sigma_i - \hat{\sigma}_i)^2 + \hat{\beta}\sum_{i=r+1}^{c}\sigma_i\right). \tag{5.4}$$

The substitution of the ML estimates $\hat{\beta}$ and $\hat{s}^2$ into (5.4) yields

$$L(r) = (c-r)\ln\left(\frac{c-r}{\sum_{i=r+1}^{c}\sigma_i}\right) - \frac{r}{2}\ln\left(\frac{2\pi}{r}\sum_{i=1}^{r}(\sigma_i - \hat{\sigma}_i)^2\right) - \left(c - \frac{r}{2}\right). \tag{5.5}$$

In [68], it is assumed that the theoretically exact non-zero singular values $\hat{\sigma}_i$ can be modeled with a finite series of Gram polynomials [29] of degree l. It therefore yields

$$\begin{aligned} L(r) &= (c-r)\ln\left(\frac{c-r}{\sum_{i=r+1}^{c}\sigma_i}\right) - \frac{r}{2}\ln\left(\frac{2\pi}{r}\left(\rho_r - \sigma^{(r)^T}V^{(r,l)}\sigma^{(r)}\right)\right) \\ &\quad - \left(c-\frac{r}{2}\right). \end{aligned} \tag{5.6}$$

where $\rho_r = \sum_{i=1}^{r}\sigma_i^2$, $\sigma^{(r)} \in \mathbb{R}^r$ is the vector of the r largest inexact singular values, and $V^{(r,l)} \in \mathbb{R}^{r\times r}$ is a matrix whose entries are functions of the Gram polynomials, and more details are shown in [2]. The smoothing of noisy data requires that $l \ll r$, and numerous computational experiments, which were also observed by Allan [2], showed that the second term on the right hand side of (5.6) can be simplified because

$$\rho_r \gg \sigma^{(r)^T}V^{(r,l)}\sigma^{(r)},$$

for all values of l. The likelihood expression (5.6) therefore simplifies to

$$L(r) = (c-r)\ln\left(\frac{c-r}{\sum_{i=r+1}^{c}\sigma_i}\right) - \frac{r}{2}\ln\left(\frac{2\pi}{r}\rho_r\right) - \left(c-\frac{r}{2}\right), \tag{5.7}$$

and this expression is evaluated for all values of $r = 1,\dots,c$. The value of r_* that maximises $L(r)$ is equal to the rank of $B(f,g)$.

The derivation of the ML formulation makes some assumptions that are not discussed by Zarowski [68]:

- A Gaussian distribution for the errors of the non-zero singular values and the independence for these errors are assumed, but their justification is not stated.

- The exponential distribution is one-sided and therefore suitable for the representation of the errors of the zero singular values. Another possible distribution is the one-side Gaussian distribution,

$$p(e_i) = \frac{2}{\sqrt{\pi}s}\exp\left(-\frac{e_i^2}{2s^2}\right) \qquad i = r+1,\dots,c$$

where $e_i \geq 0$, but a discussion of these and other probability distributions is not given.

- Gram polynomials are suitable for performing a least squares approximate over a discrete point set in $\sum_{i=1}^{r}(\sigma_i - \hat{\sigma}_i)^2$, but it is not guaranteed that the interpolated singular values are non-negative.
- Low degree polynomial models are computationally convenient. They are not, however, optimal for representing the decay of the singular values of a matrix, which is typically exponential.

It is noted that the matrix $B(f,g)$ can be used for the principle of ML due to its property of (4.15), and the matrix $S(f,g)$ is not suitable because

$$S(f,\alpha g) \neq \alpha S(f,g), \quad \alpha \neq 1.$$

and the singular values $\sigma_j(S(f,\alpha g))$ of $S(f,\alpha g)$ satisfy

$$\sigma_j(S(f,\alpha g)) \neq \alpha\sigma_j(S(f,g)), \quad j = 1,\dots,m+n, \alpha \neq 1.$$

Example 5.1. Consider the polynomials

$$\begin{aligned}\hat{f}(x) &= (x-6.1917)^6(x-3.9534)^5(x+1.8435)^{15}(x+0.8783)^{24}\\ \hat{g}(x) &= (x-6.1917)^6(x+7.8799)^6(x+2.5278)^7\end{aligned}$$

whose GCD is of degree 6, and thus the rank of $B(\hat{f},\hat{g})$ is equal to 44.

Noise with componentwise signal-to-noise value $\varepsilon_c^{-1} = 10^7$ was applied to the polynomials $\hat{f}(x)$ and $\hat{g}(x)$, and the singular values of the perturbed Bézout matrix were computed. The results were repeated 1000 times, and the histograms of four singular values are shown in Figure 5.1. It seems that σ_{44} has an exponential distribution, rather than a Gaussian distribution that is assumed by the principle of ML. It is clear that σ_{49} and σ_{50} have an exponential distribution, but they have different values of β

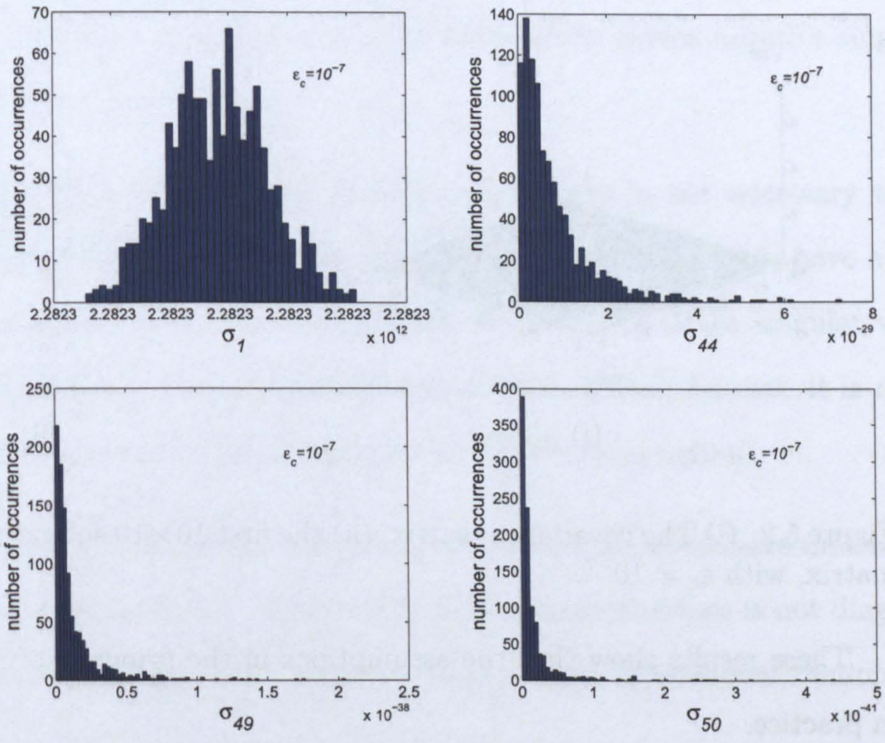


Figure 5.1: Histograms of four singular values of a perturbed Bézout matrix.

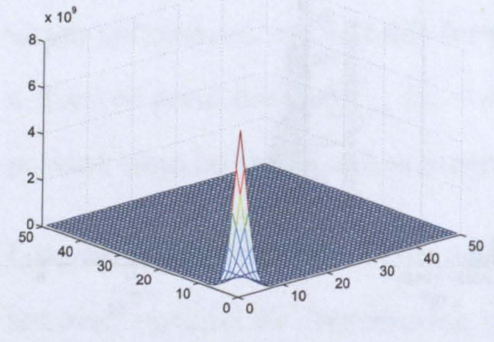
in (5.2) because $\beta \approx 220$ when $\sigma_{49} = 0$ and $\beta \approx 390$ when $\sigma_{50} = 0$.

Figure 5.2 shows the covariance matrix of the singular values that are calculated from 1000 sets of the singular values of $B(f, g)$. It can be seen that:

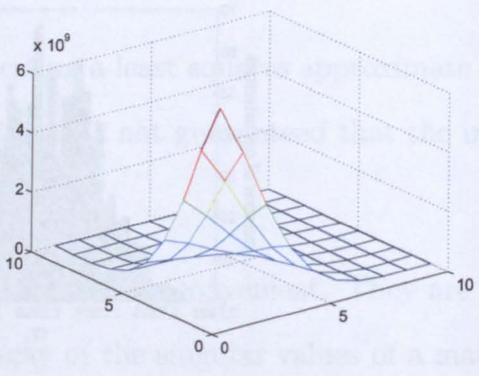
(a) The covariance matrix is ill-posed because the singular values are not independent variables.

(b) The non-increasing order and rapid decay of the singular values implies that the covariance matrix has a small bandwidth.

(c) Only the elements in a small leading submatrix of the covariance matrix are significant.



(i)



(ii)

Figure 5.2: (i) The covariance matrix, (ii) the first 10×10 submatrix of the covariance matrix, with $\varepsilon_c = 10^{-7}$.

These results show that the assumptions in the principle of ML are not realised in practice. □

Several improvements for the principle of maximum likelihood are considered as follows:

- A cubic polynomial spline [6], pages 159−162, can be used to represent the decay of the non-zero singular values instead of one polynomial, but computational experiments show that the results are very sensitive to the location of knots and their number.

- Since the computed singular values cannot be negative, constraints are necessary for the least squares problem

$$\min_{\hat{\sigma} \in \mathbb{R}^r} \sum_{i=1}^{r} (\sigma_i - \hat{\sigma_i})^2 ,$$

  and this leads to the bound least square (BLS) problem. Theoretically, the active-set method [50], pages 500 − 507, can be used to solve the constrained optimization problem. It is, however, difficult to implement in this case because the constraint $\hat{\sigma}_r > 0$ is always inactive, which causes negative singular values to occur.

- Using $\log \hat{\sigma}_i$ instead of $\hat{\sigma}_i$ is better because it is not necessary to force the non-negativity constraint on the singular values, and $\log \hat{\sigma}_i$ have a log-normal distribution. The calculation of the ML estimate of the singular values using $\log \hat{\sigma}_i$ instead of $\hat{\sigma}_i$ is, however, much more difficult because it is necessary to solve a set of non-linear questions in the ML formulation.

- Example 5.1 and other examples showed that the covariance matrix of a structured matrix that is subject to structured perturbations is not diagonal, which means the errors of the singular values are not independent random variables.

- Example 5.1 and other examples showed that the probability distribution of the small singular values $\sigma_i, i \leq r$, may or may not be Gaussian, depending on the form of matrix. Similarly, experiments showed that the probability distribution of the zero singular values $\sigma_i, r < i \leq c$, may or may not be exponential.

5.2 Method 2: The angle between subspaces

This section uses (4.7) and the partitioned structure of the subresultant matrices $S_k(\hat{f}, \hat{g})$ to calculate the degree $\hat{d}$ of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$. In particular, an expression is derived for the smallest angle between the spaces spanned by the columns of $S_{n-k+1}(\hat{f})$ and $S_{m-k+1}(\hat{g})$, which are defined in (4.6). The smallest angle is therefore defined by certain principal vectors in each of these spaces [24, 63].

Let $\mathcal{F}_k$ and $\mathcal{G}_k$ be the subspaces spanned by the the columns of $S_{n-k+1}(\hat{f})$ and $S_{m-k+1}(\hat{g})$ respectively, whose dimensions satisfy

$$\dim \mathcal{F}_k = n - k + 1 := p, \quad \dim \mathcal{G}_k = m - k + 1 := q, \quad m + n - k + 1 := l. \tag{5.8}$$

If $u_k \in \mathcal{F}_k$ and $v_k \in \mathcal{G}_k$ are non-zero vectors, then the unique angle ϑ_k between u_k and v_k is

$$\cos \vartheta_k = \frac{u_k^T v_k}{\|u_k\| \|v_k\|},$$

where $\|\cdot\| = \|\cdot\|_2$.

Obviously the angle ϑ_k changes as different vectors u_k and v_k are chosen. The first principal angle between $\mathcal{F}_k$ and $\mathcal{G}_k$ is defined to be the smallest angle that can be formed between $u_k \in \mathcal{F}_k$ and $v_k \in \mathcal{G}_k$. Since this angle is minimized when the cosine is maximised, the first principal angle satisfies

$$\cos \vartheta_{k,1} = \max\{u_k^T v_k | u_k \in \mathcal{F}_k, \|u_k\| = 1, v_k \in \mathcal{G}_k, \|v_k\| = 1\}. \tag{5.9}$$

Theorem 5.2.1. *The first principle angle $\vartheta_{k,1}$ between $\mathcal{F}_k$ and $\mathcal{G}_k$ is zero if and only if the exact polynomials $\hat{f}(x)$ and $\hat{g}(x)$ have a common divisor of degree $k \geq 1$.*

Proof. Assume $\hat{f}(x)$ and $\hat{g}(x)$ have a common divisor of degree $k \geq 1$, in which case it follows from (4.5) that there exist a non-zero vector $\mathbf{t}_k$, such that

$$\mathbf{t}_k = S_{n-k+1}(\hat{f})\mathbf{v}_k = S_{m-k+1}(\hat{g})\mathbf{u}_k \neq 0, \tag{5.10}$$

Since $\mathbf{t}_k \neq 0$ for $k = 1, \ldots, \hat{d}$, where $\hat{d}$ is the degree of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$, it follows that $\mathbf{t}_k$ is a linear combination of the columns of $S_{n-k+1}(\hat{f})$ and a linear combination of the columns of $S_{m-k+1}(\hat{g})$, and thus $\mathbf{t}_k$ lies in $\mathcal{F}_k$ and $\mathcal{G}_k$. According to the definition of the first principal angle, the smallest angle $\vartheta_{k,1}$ between $\mathcal{F}_k$ and $\mathcal{G}_k$ that can be formed between a vector $\mathbf{t}_k \in \mathcal{F}_k$ and a vector $\mathbf{t}_k \in \mathcal{G}_k$ is equal to zero for $k = 1, \ldots, \hat{d}$.

Conversely, if $\vartheta_{1,k} = 0$ for $k = 1, \dots, \hat{d}$,, then there exists a vector $\mathbf{s}_k \neq 0$ that lies in $\mathcal{F}_k$ and $\mathcal{G}_k$, and thus there exist vectors $\mathbf{u}_k$ and $\mathbf{v}_k$ such that

$$\mathbf{s}_k = S_{n-k+1}(\hat{f})\mathbf{v}_k = S_{m-k+1}(\hat{g})\mathbf{u}_k \neq 0,$$

from which (4.5) follows. □

If $\hat{f}(x)$ and $\hat{g}(x)$ are coprime, then the only solution of (4.6) is $\mathbf{u}_k = 0$ and $\mathbf{v}_k = 0$, that is, there does not exist a non-zero vector $\mathbf{t}_k$ that satisfies (5.10). Hence the first principal angle $\vartheta_{k,1}$ between $\mathcal{F}_k$ and $\mathcal{G}_k$ is greater than zero.

Also equation (5.10) does not possess a non-zero vector $\mathbf{t}_k$ for all values of k when inexact polynomials $f(x)$ and $g(x)$ are specified because they are coprime. It follows that, in addition to the preprocessing operations that are discussed in Section 4.4, Theorem 5.2.1 must be modified slightly so that it is suitable for inexact polynomials.

If $\tilde{f}(y)$ and $\tilde{g}(y)$ are the processed polynomials from $f(x)$ and $g(x)$, then the application of Theorem 5.2.1 to $\tilde{f}(y)$ and $\tilde{g}(y)$ requires that $\vartheta_{k,1}$ is monitored as a function of k, and the value of k at which $\vartheta_{k,1}$ changes from a small value to a large value is equal to the degree d of an approximate GCD of this polynomial pair. The value of d is therefore determined by k for which the change between $\vartheta_{k,1}$ and $\vartheta_{k+1,1}$ is a maximum, as

$$d = \{k : \max(\vartheta_{k+1,1} - \vartheta_{k,1}); \quad k = 1, \dots, \min(m, n) - 1\}. \tag{5.11}$$

Since the first principle angle $\vartheta_{k,1}$ is used to determine d in (5.11), the next section shows how $\vartheta_{k,1}$ can be calculated.

5.2.1 Calculating the first principal angles

The calculation of the first principle angle between $\mathcal{F}_k$ and $\mathcal{G}_k$ is considered in this section.

The QR decomposition of $S_{n-k+1}(\tilde{f})$ and $S_{m-k+1}(\tilde{g})$ yields matrices $P_k \in \mathbb{R}^{l\times p}$ and $Q_k \in \mathbb{R}^{l\times q}$ whose columns define orthonormal bases for $\mathcal{F}_k$ and $\mathcal{G}_k$ respectively,

$$\begin{aligned} S_{n-k+1}(\tilde{f}) &= P_k R_{k,p} \qquad P_k \in \mathbb{R}^{l\times p} \qquad R_{k,p} \in \mathbb{R}^{p\times p}, \\ S_{m-k+1}(\tilde{g}) &= Q_k R_{k,q} \qquad Q_k \in \mathbb{R}^{l\times q} \qquad R_{k,q} \in \mathbb{R}^{q\times q}, \end{aligned} \tag{5.12}$$

and the columns of P_k and Q_k are orthogonal,

$$P_k^T P_k = I_{k,p} \qquad \text{and} \qquad Q_k^T Q_k = I_{k,q}. \tag{5.13}$$

There therefore exist vectors $y_k \in \mathbb{R}^p$ and $z_k \in \mathbb{R}^q$ such that

$$u_k = P_k y_k \qquad \text{and} \qquad v_k = Q_k z_k$$

and the conditions (5.13) and

$$\|u_k\| = \|v_k\| = 1 \qquad \text{or} \qquad u_k^T u_k = v_k^T v_k = 1,$$

implies

$$\|y_k\| = \|z_k\| = 1 \qquad \text{or} \qquad y_k^T y_k = z_k^T z_k = 1.$$

It therefore follows from (5.9) that

$$\max_{u_k\in\mathcal{F}_k} \max_{v_k\in\mathcal{G}_k} u_k^T v_k = \max_{y_k\in\mathbb{R}^p} \max_{z_k\in\mathbb{R}^q} y_k^T (P_k^T Q_k) z_k \tag{5.14}$$

subject to

$$\|u_k\| = \|v_k\| = \|y_k\| = \|z_k\| = 1.$$

If the singular value decomposition of $P_k^T Q_k$ is $Y_k \Sigma_k Z_k^T$, where $Y_k \in \mathbb{R}^{p\times p}$ and $Z_k \in \mathbb{R}^{q\times q}$ are orthogonal matrices, $\Sigma_k \in \mathbb{R}^{p\times q}$, and the singular values $\varsigma_{k,i}, i =$

$1,\ldots,\min(p,q)$, are arranged in non-increasing order, then (5.14) yields

$$\max_{u_k\in\mathcal{F}_k}\max_{v_k\in\mathcal{G}_k} u_k^T v_k = \max_{y_k\in\mathbb{R}^p}\max_{z_k\in\mathbb{R}^q} y_k^T(Y_k\Sigma_k Z_k^T)z_k = \varsigma_{k,1},$$

which implies that the cosine of the first principal angle is equal to the largest singular value of $P_k^T Q_k$, based on (5.9),

$$\cos\vartheta_{k,1} = \varsigma_{k,1}.$$

This maximum is attained when y_k and z_k are equal to the first column of Y_k and Z_k, respectively.

Algorithm 5.1 summarises the use of the SVD to calculate the first principal angle between the subspaces $\mathcal{F}_k$ and $\mathcal{G}_k$.

Algorithm 5.1: The calculation of the first principal angle

Input: Two inexact polynomials $f(x)$ and $g(x)$, and an integer k.

Output: The first principal angle $\vartheta_{k,1}$.

Begin

1. Preprocess $f(x)$ and $g(x)$ to yield the polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$, as shown in Section 4.4, and form the matrices $S_{n-k+1}(\tilde{f})$ and $S_{m-k+1}(\tilde{g})$.

2. Apply the QR decomposition (5.12) to $S_{n-k+1}(\tilde{f})$ and $S_{m-k+1}(\tilde{g})$ in order to calculate the matrices P_k and Q_k.

3. Compute $P_k^T Q_k$.

4. Calculate the SVD of $P_k^T Q_k$, which is equal to $Y_k\Sigma_k Z_k^T$. Let $\varsigma_{k,1}$ denote the largest singular value of $P_k^T Q_k$.

5. Calculate the first principal angle $\vartheta_{k,1} = \cos^{-1} \varsigma_{k,1}$.

End

The first principal angle between the subspaces $\mathcal{F}_k$ and $\mathcal{G}_k$ is given by

$$\vartheta_{k,1} = \cos^{-1} \varsigma_{k,1}, \tag{5.15}$$

and computational problems arise when $\vartheta_{k,1} \approx 0$ because it follows from this equation that to first order,

$$\delta\vartheta_{k,1} = -\frac{\delta\varsigma_{k,1}}{\sin\vartheta_{k,1}}, \tag{5.16}$$

and thus $|\delta\vartheta_{k,1}| \gg |\delta\varsigma_{k,1}|$ if $\vartheta_{k,1} \approx 0$. Since the computation (5.15) cannot yield an accurate value for an angle near zero, a modification to this method of calculating the first principal angle is therefore required in this circumstance, and this is considered in the next section.

5.2.2 Calculating the small first principal angle

It was shown in Section 5.2.1 that the first principal angle cannot be determined accurately when it is small, and thus a stable method for its computation is required. This case is considered in this section, and the following theorem is established in [63].

Theorem 5.2.2. *Let the columns of $W \in \mathbb{R}^{l\times p}$ be orthonormal, and let W be partitioned as*

$$W = \begin{bmatrix} W_1 \\ W_2 \end{bmatrix}, \qquad W_1 \in \mathbb{R}^{l_1\times p}, \qquad W_2 \in \mathbb{R}^{l_2\times p}, \qquad l_1 + l_2 = l.$$

Let ς_1 be the largest singular value of W_1, and let σ_1 be the smallest non-zero singular value of W_2, then

$$\varsigma_1^2 + \sigma_1^2 = 1. \tag{5.17}$$

Proof. Since the columns of W are orthonormal, it follows that

$$W^T W = W_1^T W_1 + W_2^T W_2 = I_p. \tag{5.18}$$

If (λ, t) and (μ, t) are eigenpairs of $W_1^T W_1$ and $W_2^T W_2$ respectively, then

$$(W_1^T W_1)t = \lambda t, \quad \text{and} \quad (W_2^T W_2)t = \mu t,$$

and thus

$$(W_1^T W_1 + W_2^T W_2)t = (\lambda + \mu)t.$$

It therefore follows from (5.18) that

$$I_p t = (\lambda + \mu)t,$$

that is,

$$\lambda + \mu = 1. \tag{5.19}$$

The first eigenvalue of $W_1^T W_1$ is ς_1^2, and thus it follows from (5.19) that the first eigenvalue of $W_2^T W_2$ is $1 - \varsigma_1^2$. Since the first eigenvalue of $W_2^T W_2$ is equal to σ_1^2, it follows that the sum of the first eigenvalue of $W_1^T W_1$ and $W_2^T W_2$ is equal to one and thus (5.17) is established. $\square$

Consider now the orthogonal complements $\mathcal{F}_k^\perp$ and $\mathcal{G}_k^\perp$ which will be required in Theorem 5.2.3, where

$$\mathcal{F}_k \cup \mathcal{F}_k^\perp = \mathbb{R}^l, \qquad \dim \mathcal{F}_k^\perp = l - p,$$

and

$$\mathcal{G}_k \cup \mathcal{G}_k^{\perp} = \mathbb{R}^l, \qquad \dim \mathcal{G}_k^{\perp} = l - q.$$

It will be necessary to calculate orthonormal bases for $\mathcal{F}_k^{\perp}$ and $\mathcal{G}_k^{\perp}$, and these bases will define the columns of the matrices $\overline{P}_k \in \mathbb{R}^{l\times(l-p)}$ and $\overline{Q}_k \in \mathbb{R}^{l\times(l-q)}$ respectively. The columns of P_k and Q_k, which are introduced in Section 5.2.1, define orthonormal bases for $\mathcal{F}_k$ and $\mathcal{G}_k$ respectively. It follows that the columns of F_k and G_k are given by

$$F_k = \begin{bmatrix} P_k & \overline{P}_k \end{bmatrix} \in \mathbb{R}^{l\times l}, \qquad F_k^T F_k = F_k F_k^T = I_l,$$

and

$$G_k = \begin{bmatrix} Q_k & \overline{Q}_k \end{bmatrix} \in \mathbb{R}^{l\times l}, \qquad G_k^T G_k = G_k G_k^T = I_l,$$

respectively, which define orthonormal bases for $\mathbb{R}^l$. The following theorem is established in [63].

Theorem 5.2.3. *Let $\mathcal{F}_k$ and $\mathcal{G}_k$ be subspaces of $\mathbb{R}^m$ where (5.8) is satisfied, and let $\vartheta_{k,1}$ be the first principal angle between them. Let the columns of $P_k \in \mathbb{R}^{l\times p}$ and $Q_k \in \mathbb{R}^{l\times q}$ define orthonormal bases for $\mathcal{F}_k$ and $\mathcal{G}_k$ respectively. Also, let the columns of $\overline{P}_k \in \mathbb{R}^{l\times(l-p)}$ and $\overline{Q}_k \in \mathbb{R}^{l\times(l-q)}$ define orthonormal bases for $\mathcal{F}_k^{\perp}$ and $\mathcal{G}_k^{\perp}$ respectively. Then the smallest non-zero singular values of $\overline{P}_k^T Q_k \in \mathbb{R}^{(l-p)\times q}$ and $P_k^T \overline{Q}_k \in \mathbb{R}^{p\times(l-q)}$ are*

$$\sigma_{k,1} = \sin \vartheta_{k,1}.$$

Proof. Since F_k is an orthogonal matrix and Q_k has orthonormal columns, the columns of $W_{f,k} \in \mathbb{R}^{l \times q}$,

$$W_{f,k} = F_k^T Q_k = \begin{bmatrix} P_k^T Q_k \\ \overline{P}_k^T Q_k \end{bmatrix}, \qquad P_k^T Q_k \in \mathbb{R}^{p \times q}, \qquad \overline{P}_k^T Q_k \in \mathbb{R}^{(l-p) \times q},$$

must be orthonormal. Also the largest singular value of $P_k^T Q_k$ is $\varsigma_{k,1} = \cos \vartheta_{k,1}$, so by Theorem 5.2.2, the smallest non-zero singular value of $\overline{P}_k^T Q_k$ is equal to

$$\sigma_{k,1} = \sqrt{1 - \varsigma_{k,1}^2} = \sqrt{1 - \cos^2 \vartheta_{k,1}} = \sin \vartheta_{k,1}. \tag{5.20}$$

Consider now the matrix $W_{g,k} \in \mathbb{R}^{l \times p}$,

$$W_{g,k} = G_k^T P_k = \begin{bmatrix} Q_k^T P_k \\ \overline{Q}_k^T P_k \end{bmatrix}, \qquad Q_k^T P_k \in \mathbb{R}^{q \times p}, \qquad \overline{Q}_k^T P_k \in \mathbb{R}^{(l-q) \times p}.$$

Since the largest singular value of $Q_k^T P_k$ is $\cos \vartheta_{k,1}$, it follows from Theorem 5.2.3 that the smallest non-zero singular value $\overline{Q}_k^T P_k$ is $\sin \vartheta_{k,1}$. □

The computation $\vartheta_{k,1} = \arcsin \sigma_{k,1}$ in (5.20) will give an accurate value when $\vartheta_{k,1} \approx 0$. By contrast, the computation (5.15) will give an accurate value when $\vartheta_{k,1} \approx \frac{\pi}{2}$.

The only issue that must be considered is the calculation of the matrices $\overline{P}_k$ and $\overline{Q}_k$, whose columns define orthonormal bases for spaces $\mathcal{F}_k^{\perp}$ and $\mathcal{G}_k^{\perp}$. It is recalled that P_k and Q_k are calculated from the QR decomposition of $S_{n-k+1}(\tilde{f})$ and $S_{m-k+1}(\tilde{g})$, respectively, as shown in (5.12). It is adequate to consider the calculation of $\overline{P}_k$ because the calculation of $\overline{Q}_k$ follows identically.

The columns of P_k provide an orthonormal basis for $\mathcal{F}_k$, and thus all vectors $x \in \mathbb{R}^m$ that satisfy $P_k^T x = 0$, are orthogonal to the columns of P_k, which means these vectors x lie in $\mathcal{F}_k^{\perp}$. Since an orthonormal basis for $\mathcal{F}_k^{\perp}$ is required, it is necessary to choose an orthonormal set of vectors x, and this is now considered.

If the SVD of P_k is

$$P_k = U_k \begin{bmatrix} \Sigma_k \\ 0 \end{bmatrix} V_k^T,$$

where $U_k \in \mathbb{R}^{l\times l}, V_k \in \mathbb{R}^{p\times p}, \Sigma_k \in \mathbb{R}^{p\times p}$ is a diagonal matrix of the singular values $\gamma_i, i = 1,\ldots,p$, of P_k, arranged in non-increasing order, and the zero matrix is of order $(l-p)\times p$, then

$$P_k^T x_j = V_k \begin{bmatrix} \Sigma_k^T & 0 \end{bmatrix} U_k^T x_j,$$

where $x_j, j = 1,\ldots,l$, is the jth column of U_k. It is necessary to consider two situations, which are defined by $1 \leq j \leq p$ and $p+1 \leq j \leq l$.

If $1 \leq j \leq p$, then

$$P_k^T x_j = V_k \begin{bmatrix} \Sigma_k^T & 0 \end{bmatrix} e_j = \gamma_j c_j,$$

where e_j is the jth unit basis vector and c_j is the jth column of V_k.

If $p+1 \leq j \leq l$, then

$$P_k^T x_j = V_k \begin{bmatrix} \Sigma_k^T & 0 \end{bmatrix} e_j = 0,$$

and thus the last $l-p$ columns of the left singular matrix U_k of P_k provide an orthonormal basis for $\mathcal{F}_k^{\perp}$. It follows that if

$$\overline{P}_k = \begin{bmatrix} x_{p+1} & x_{p+2} & \cdots & x_{l-1} & x_l \end{bmatrix}, \tag{5.21}$$

then

$$\overline{P}_k^T \overline{P}_k = I_{l-p}, \qquad P_k^T \overline{P}_k = 0, \qquad \overline{P}_k^T P_k = 0.$$

Similarly, the matrix $\overline{Q}_k$ is defined by the last $l-q$ columns of the left singular matrix of Q_k.

Algorithm 5.2 summarises the use of the SVD to calculate accurately the small first principal angle between the subspaces $\mathcal{F}_k$ and $\mathcal{G}_k$.

Algorithm 5.2: The calculation of the small first principal angle

Input: Two inexact polynomials $f(x)$ and $g(x)$, and an integer k.

Output: The first principal angle $\vartheta_{k,1}$.

Begin

```
1. Preprocess $f(x)$ and $g(x)$ to yield the polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$, as shown in
   Section 4.4, and form the matrices $S_{n-k+1}(\tilde{f})$ and $S_{m-k+1}(\tilde{g})$.

2. Apply the QR decomposition (5.12) to $S_{n-k+1}(\tilde{f})$ and $S_{m-k+1}(\tilde{g})$ in order to
   calculate the matrices $P_k$ and $Q_k$.

3. Calculate the matrices $\overline{P}_k$, where $\overline{P}_k$ is defined in (5.21).

4. Compute $\overline{P}_k^T Q_k$.

5. Calculate the SVD of $\overline{P}_k^T Q_k$. Let $\sigma_{k,1}$ denote the smallest non-zero singular
   value of $\overline{P}_k^T Q_k$.

6. Calculate the first principal angle $\vartheta_{k,1} = \arcsin \sigma_{k,1}$.
```

End

5.3 Method 3: The error between two estimates of an approximate common divisor

This section considers the change, with $k = 1, \dots, \min(m, n)$, of the error between two estimates of an approximate common divisor of degree k of $f(x)$ and $g(x)$ in order to calculate the degree d of an approximate GCD of $f(x)$ and $g(x)$. These estimates are calculated from the Sylvester subresultant matrices $S_k(\tilde{f}, \alpha_o \tilde{g})$ and $S_k(\tilde{g}, \tilde{f}/\alpha_o)$ where $\tilde{f} = \tilde{f}(y)$ and $\tilde{g} = \tilde{g}(y)$ are the scaled polynomials of $f(x)$ and $g(x)$, respectively, the parameter α_o is equal to the weight of $\tilde{g}(y)$ relative to the weight of $\tilde{f}(y)$, as shown in Section 4.4.

It is shown in (4.7) that the constraint $v_{k,0} = -1$ is imposed when exact data is specified, and this allows the homogeneous equation (4.6) to be transformed to the linear algebraic equation (4.7). This constraint for exact data can be replaced by the constraint $u_{k,0} = 1$ because the leading coefficient of the quotient polynomials $u_k(x)$ and $v_k(x)$ cannot be equal to zero in (4.3) due to the existence of a common divisor of degree k.

Equation (4.7) does not possess a solution when inexact polynomial pair $\tilde{f}(y)$ and $\tilde{g}(y)$ are specified, and it is therefore solved in the least squares sense, in which case the polynomials $u_k(x)$ and $v_k(x)$ are replaced by the polynomials $\tilde{u}_k(y)$ and $\tilde{v}_k(y)$, respectively. It is not clear, however, which constraint, $\tilde{u}_{k,0} = 1$ or $\tilde{v}_{k,0} = -1$, should be imposed in this circumstance, that is, it is not known *a priori* which constraint yields an approximate solution that is nearer a solution of (4.7) when inexact data is specified because the two constraints may yields different approximate GCDs.

Consider the constraint $\tilde{v}_{k,0} = -1$ first, and thus (4.7) is replaced by the approximate equation

$$\begin{bmatrix} & & & \alpha_o\tilde{b}_0 & & & \\ \tilde{a}_0 & & & \alpha_o\tilde{b}_1 & \alpha_o\tilde{b}_0 & & \\ \tilde{a}_1 & \ddots & & \vdots & \alpha_o\tilde{b}_1 & \ddots & \\ \vdots & \ddots & \tilde{a}_0 & \alpha_o\tilde{b}_{n-1} & \vdots & \ddots & \alpha_o\tilde{b}_0 \\ \tilde{a}_{m-1} & \ddots & \tilde{a}_1 & \alpha_o\tilde{b}_n & \alpha_o\tilde{b}_{n-1} & \ddots & \alpha_o\tilde{b}_1 \\ \tilde{a}_m & \ddots & \vdots & & \alpha_o\tilde{b}_n & \ddots & \vdots \\ & \ddots & \tilde{a}_{m-1} & & & \ddots & \alpha_o\tilde{b}_{n-1} \\ & & \tilde{a}_m & & & & \alpha_o\tilde{b}_n \end{bmatrix} \begin{bmatrix} \tilde{v}_1 \\ \vdots \\ \tilde{v}_{n-k} \\ -\tilde{u}_0 \\ -\tilde{u}_1 \\ \vdots \\ -\tilde{u}_{m-k} \end{bmatrix} \approx \begin{bmatrix} \tilde{a}_0 \\ \tilde{a}_1 \\ \vdots \\ \tilde{a}_{m-1} \\ \tilde{a}_m \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

that is,

$$\dot{H}_k x_k \approx \dot{h}_k, \qquad k = 1, \dots, \min(m, n), \tag{5.22}$$

where

$$\begin{bmatrix} \tilde{\mathbf{v}}_k \\ -\tilde{\mathbf{u}}_k \end{bmatrix} = \begin{bmatrix} \tilde{v}_{k,0} \\ x_k \end{bmatrix} = \begin{bmatrix} -1 \\ x_k \end{bmatrix}, \qquad k = 1, \dots, \min(m, n), \tag{5.23}$$

and $\dot{H}_k = \dot{H}(\tilde{f}, \alpha_o\tilde{g})$ and $\dot{h}_k = \dot{h}_k(\tilde{f})$ are obtained by imposing a constraint on $\tilde{v}_{k,0}$. The least squares solution of (5.22) is

$$\dot{x}_k = \dot{H}_k^\dagger \dot{h}_k, \qquad \dot{H}_k^\dagger = (\dot{H}_k^T \dot{H}_k)^{-1}\dot{H}_k^T, \qquad k = 1, \dots, \min(m, n), \tag{5.24}$$

and thus its normalised residual is

$$res_{k,1} = \frac{\|\dot{H}_k\dot{x}_k - \dot{h}_k\|}{\|\dot{h}_k\|} = \frac{\|(\dot{H}_k\dot{H}_k^\dagger - I)\dot{h}_k\|}{\|\dot{h}_k\|}, \qquad k = 1, \dots, \min(m, n). \tag{5.25}$$

Now consider the constraint $\tilde{u}_{k,0} = 1$, which is most easily imposed by developing the Sylvester subresultant matrices $S_k(\tilde{g}, \tilde{f}/\alpha_o), k = 1, \dots, \min(m, n)$, due to $S_k(\tilde{f}, \alpha_o\tilde{g}) = \alpha_o S_k(\tilde{f}/\alpha_o, \tilde{g})$, that is, the coefficients of $\tilde{g}(y)$ occupy the first $m - k + 1$

columns, and the coefficients of $\tilde{f}(y)$ occupy the last $n-k+1$ columns, of $S_k(\tilde{g}, \tilde{f}/\alpha_o)$. Equation (5.22) is therefore replaced by the approximate equation

$$\begin{bmatrix} & & & \tilde{a}_0/\alpha_o & & & \\ \tilde{b}_0 & & & \tilde{a}_1/\alpha_o & \tilde{a}_0/\alpha_o & & \\ \tilde{b}_1 & \ddots & & \vdots & \tilde{a}_1/\alpha_o & \ddots & \\ \vdots & \ddots & \tilde{b}_0 & \tilde{a}_{m-1}/\alpha_o & \vdots & \ddots & \tilde{a}_0/\alpha_o \\ \tilde{b}_{n-1} & \ddots & \tilde{b}_1 & \tilde{a}_m/\alpha_o & \tilde{a}_{m-1}/\alpha_o & \ddots & \tilde{a}_1/\alpha_o \\ \tilde{b}_n & \ddots & \vdots & & \tilde{a}_m/\alpha_o & \ddots & \vdots \\ & \ddots & \tilde{b}_{n-1} & & & \ddots & \tilde{a}_{m-1}/\alpha_o \\ & & \tilde{b}_n & & & & \tilde{a}_m/\alpha_o \end{bmatrix} \begin{bmatrix} -\tilde{u}_{k,1} \\ \vdots \\ -\tilde{u}_{k,m-k} \\ \tilde{v}_{k,0} \\ \tilde{v}_{k,1} \\ \vdots \\ \tilde{v}_{k,n-k} \end{bmatrix} \approx \begin{bmatrix} \tilde{b}_0 \\ \tilde{b}_1 \\ \vdots \\ \tilde{b}_{n-1} \\ \tilde{b}_n \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

that is,

$$\ddot{H}_k x_k \approx \ddot{h}_k, \qquad k = 1, \ldots, \min(m, n), \tag{5.26}$$

where

$$\begin{bmatrix} -\tilde{\mathbf{u}}_k \\ \tilde{\mathbf{v}}_k \end{bmatrix} = \begin{bmatrix} -\tilde{u}_{k,0} \\ x_k \end{bmatrix} = \begin{bmatrix} -1 \\ x_k \end{bmatrix}, \qquad k = 1, \ldots, \min(m, n),$$

where $\ddot{H}_k = \ddot{H}_k(\tilde{g}, \tilde{f}/\alpha_o)$ is formed from $S_k(\tilde{g}, \tilde{f}/\alpha_o)$ by deleting the first column of the coefficients of $\tilde{g}(y)$, and $\ddot{h}_k = \ddot{h}_k(\tilde{g})$. The quotient polynomials $\tilde{u}_k(y)$ and $\tilde{v}_k(y)$ can be computed from the approximation (5.26), whose least squares solution is

$$\ddot{x}_k = \ddot{H}_k^{\dagger}\ddot{h}_k, \qquad \ddot{H}_k^{\dagger} = (\ddot{H}_k^T \ddot{H}_k)^{-1}\ddot{H}_k^T, \qquad k = 1, \ldots, \min(m, n), \tag{5.27}$$

and its residual is, following (5.25),

$$res_{k,2} = \frac{\|\ddot{H}_k \ddot{x}_k - \ddot{h}_k\|}{\|\ddot{h}_k\|} = \frac{\|(\ddot{H}_k \ddot{H}_k^{\dagger} - I)\ddot{h}_k\|}{\|\ddot{h}_k\|}, \quad k = 1, \ldots, \min(m, n). \tag{5.28}$$

The criterion for deciding which of the approximate solutions (5.24) and (5.27) to use requires the residuals (5.25) and (5.28):

$$\begin{array}{cc}
res_{k,1} \leq res_{k,2} & res_{k,1} > res_{k,2} \\
\Downarrow & \Downarrow \\
H_k = \dot{H}_k = \dot{H}_k(\tilde{f}, \alpha_o \tilde{g}) & H_k = \ddot{H}_k = \ddot{H}_k(\tilde{g}, \tilde{f}/\alpha_o) \\
h_k = \dot{h}_k = \dot{h}_k(\tilde{f}) & h_k = \ddot{h}_k = \ddot{h}_k(\tilde{g}) \\
\Downarrow & \Downarrow \\
x_k = \dot{x}_k & x_k = \ddot{x}_k \\
\begin{bmatrix} \tilde{\mathbf{v}}_k \\ -\tilde{\mathbf{u}}_k \end{bmatrix} = \begin{bmatrix} -1 \\ \dot{x}_k \end{bmatrix}, & \begin{bmatrix} -\tilde{\mathbf{u}}_k \\ \tilde{\mathbf{v}}_k \end{bmatrix} = \begin{bmatrix} -1 \\ \ddot{x}_k \end{bmatrix},
\end{array} \tag{5.29}$$

for $k = 1, \ldots, \min(m, n)$. The solution that has the smaller residual is therefore chosen for the calculation of the approximations $\tilde{u}_k(y)$ and $\tilde{v}_k(y)$ to the theoretically exact quotient polynomials because this solution is nearer an exact solution of (4.3), the existence of which is a sufficient condition for the theoretically exact forms of $\hat{f}(x)$ and $\hat{g}(x)$ to have a non-constant common divisor.

Suppose that $res_{k,1} \leq res_{k,2}$ for some values of $k \in [1, \min(m,n)]$, and $res_{k,1} > res_{k,2}$ for the remaining values of k, and thus the forms of $\tilde{\mathbf{u}}_k$ and $\tilde{\mathbf{v}}_k$ in (5.29) are functions of k. The following theory is developed for a given but arbitrary value of k, and it follows from (4.4) that

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

Estimates $\dot{c}_k(y)$ and $\ddot{c}_k(y)$ of the common divisors of the theoretically exact forms of $\tilde{f}(y)$ and $\tilde{g}(y)$ are obtained from $\tilde{u}_k(y)$ and $\tilde{v}_k(y)$,

$$\dot{c}_k(y) \approx \frac{\tilde{f}(y)}{\tilde{u}_k(y)} \qquad \text{and} \qquad \ddot{c}_k(y) \approx \frac{\tilde{g}(y)}{\tilde{v}_k(y)}, \tag{5.30}$$

where $\approx$ is used because $\tilde{u}_k(y)$ and $\tilde{v}_k(y)$ are derived from the least squares solution

(5.24) or (5.27), and

$$\dot{c}_k(y) = \sum_{i=0}^{k} \dot{c}_{k,i} y^{k-i} \qquad \text{and} \qquad \ddot{c}_k(y) = \sum_{i=0}^{k} \ddot{c}_{k,i} y^{k-i}.$$

It is noted that $\dot{c}_k(y) \neq \ddot{c}_k(y)$ because $\tilde{f}(y)$ and $\tilde{g}(y)$ are inexact polynomials and therefore assumed to be coprime.

Since interest is restricted to solutions for which $\dot{c}_k(y)$ and $\ddot{c}_k(y)$ are polynomials, the two approximate equations in (5.30) are written in matrix-vector form, respectively,

$$\begin{bmatrix} \tilde{u}_{k,0} & & & \\ \tilde{u}_{k,1} & \tilde{u}_{k,0} & & \\ \vdots & \tilde{u}_{k,1} & \ddots & \\ \tilde{u}_{k,m-k-1} & \vdots & \ddots & \tilde{u}_{k,0} \\ \tilde{u}_{k,m-k} & \tilde{u}_{k,m-k-1} & \ddots & \tilde{u}_{k,1} \\ & \tilde{u}_{k,m-k} & \ddots & \vdots \\ & & \ddots & \tilde{u}_{k,m-k-1} \\ & & & \tilde{u}_{k,m-k} \end{bmatrix} \begin{bmatrix} \dot{c}_{k,0} \\ \dot{c}_{k,1} \\ \vdots \\ \dot{c}_{k,k-1} \\ \dot{c}_{k,k} \end{bmatrix} \approx \begin{bmatrix} \tilde{a}_0 \\ \tilde{a}_1 \\ \vdots \\ \tilde{a}_{m-1} \\ \tilde{a}_m \end{bmatrix}, \qquad (5.31)$$

and

$$\begin{bmatrix} \tilde{v}_{k,0} & & & \\ \tilde{v}_{k,1} & \tilde{v}_{k,0} & & \\ \vdots & \tilde{v}_{k,1} & \ddots & \\ \tilde{v}_{k,n-k-1} & \vdots & \ddots & \tilde{v}_{k,0} \\ \tilde{v}_{k,n-k} & \tilde{v}_{k,n-k-1} & \ddots & \tilde{v}_{k,1} \\ & \tilde{v}_{k,n-k} & \ddots & \vdots \\ & & \ddots & \tilde{v}_{k,n-k-1} \\ & & & \tilde{v}_{k,n-k} \end{bmatrix} \begin{bmatrix} \ddot{c}_{k,0} \\ \ddot{c}_{k,1} \\ \vdots \\ \ddot{c}_{k,k-1} \\ \ddot{c}_{k,k} \end{bmatrix} \approx \begin{bmatrix} \tilde{b}_0 \\ \tilde{b}_1 \\ \vdots \\ \tilde{b}_{n-1} \\ \tilde{b}_n \end{bmatrix}, \qquad (5.32)$$

that is,

$$\tilde{U}_k \dot{\mathbf{c}}_k \approx \tilde{\mathbf{f}} \qquad \text{and} \qquad \tilde{V}_k \ddot{\mathbf{c}}_k \approx \tilde{\mathbf{g}}, \tag{5.33}$$

where $\tilde{U}_k \in \mathbb{R}^{(m+1)\times(k+1)}, \tilde{V}_k \in \mathbb{R}^{(n+1)\times(k+1)}, \dot{\mathbf{c}}_k, \ddot{\mathbf{c}}_k \in \mathbb{R}^{k+1}, \tilde{\mathbf{f}} \in \mathbb{R}^{m+1}$ and $\tilde{\mathbf{g}} \in \mathbb{R}^{n+1}$. The approximate equations in (5.33) are solved in the least squares sense,

$$\dot{\mathbf{c}}_k = \tilde{U}_k^{\dagger}\tilde{\mathbf{f}} \qquad \text{and} \qquad \ddot{\mathbf{c}}_k = \tilde{V}_k^{\dagger}\tilde{\mathbf{g}}.$$

Example 4.1 shows that (4.7) possesses an infinite number of solutions for $k = 1, \ldots, \hat{d} - 1$, and a unique solution for $k = \hat{d}$, but it does not possess a solution for $k = \hat{d} + 1, \ldots, \min(m, n)$. It therefore implies that the degree d of an approximate GCD of $\tilde{f}(y)$ and $\tilde{g}(y)$ is the value of k, for which the error measure

$$e_k = \frac{\|\dot{\mathbf{c}}_k - \ddot{\mathbf{c}}_k\|}{\|\dot{\mathbf{c}}_k\| + \|\ddot{\mathbf{c}}_k\|}, \qquad k = 1, \ldots, \min(m, n), \tag{5.34}$$

achieves its minimum value.

5.4 Examples

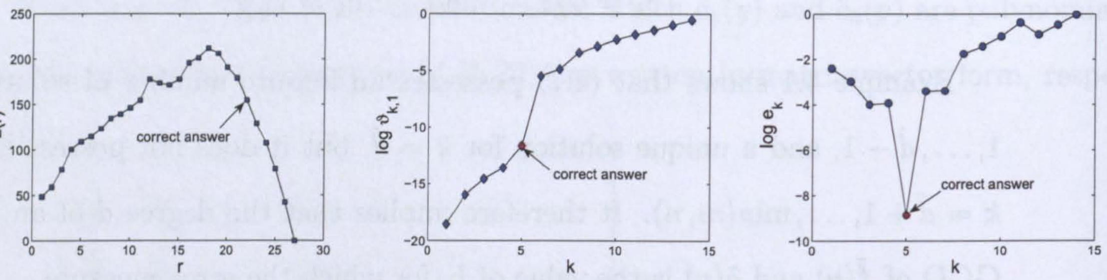
This section contains examples that compare Methods 1, 2 and 3 for the estimation of the degree of an approximate GCD of an inexact polynomial pair.

Example 5.2. Consider the exact polynomial pair

$$\begin{aligned} \hat{f}(x) &= (x - 6.7974)(x - 0.5903)^4(x - 3.3634)^3(x + 1.1265)^6 \\ \hat{g}(x) &= (x - 6.7974)^8(x - 0.5903)^9(x + 4.8572)^5(x + 6.8740)^5 \end{aligned}$$

whose GCD is of degree 5.

Each polynomial is perturbed by noise, such that the componentwise signal-to-noise ratio ε_c^{-1} is 10^8, and the inexact polynomial pair is then preprocessed by the operations described in Section 4.4. Figure 5.3 shows that Method 1 using the likelihood function (5.7) returns the incorrect value, but Method 2 and 3 using (5.11) and (5.34), respectively, return the correct value of the degree of an approximate GCD of $\tilde{f}(y)$ and $\tilde{g}(y)$. □



(i) (ii) (iii)

Figure 5.3: The variation of (i) the likelihood function $L(r)$ with the rank r, (ii) the first principal angle $\log \vartheta_{k,1}$ and (iii) the error measure $\log e_k$, with the degree k of an approximate common divisor, with $\varepsilon_c = 10^{-8}$.

Example 5.3. Consider the exact polynomial pair

$$\begin{aligned}\hat{f}(x) &= (x-7.0613)^6(x+1.1520)^8(x+3.3486)(x-1.8319)^{10}\\ \hat{g}(x) &= (x-7.0613)^3(x+1.1520)^4\end{aligned}$$

whose GCD is of degree 7.

Noise with componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^8$ was applied to $\hat{f}(x)$ and $\hat{g}(x)$, and this inexact polynomial pair is then preprocessed by the operations described in Section 4.4, thereby yielding $\tilde{f}(y)$ and $\tilde{g}(y)$.

The normalised residuals $res_{k,1}$ and $res_{k,2}$, which are defined in (5.25) and (5.28) respectively, are calculated as functions of $k = 1, \ldots, \min(m,n)$, in order to determine the importance of the criterion (5.29), using $S_k(\tilde{f}, \alpha_o\tilde{g})$ and $S_k(\tilde{g}, \tilde{f}/\alpha_o)$. Similarly, the error measure e_k, which is defined in (5.34), is calculated using $S_k(\tilde{f}, \alpha_o \tilde{g})$ and $S_k(\tilde{g}, \tilde{f}/\alpha_o)$, and the results for both experiments are shown in Figure 5.4. Figure 5.4(i) shows that an incorrect result may occur if a criterion to calculate the degree of an approximate GCD of $\tilde{f}(y)$ and $\tilde{g}(y)$ is based on $res_{k,1}$ and $res_{k,2}$, and Figure 5.4(ii) shows that a criterion based on the error measure e_k yields the correct result, independent of whether calculations are performed on $S_k(\tilde{f}, \alpha_o \tilde{g})$ or $S_k(\tilde{g}, \tilde{f}/\alpha_o)$. Also, it is clear that the minimum in Figure 5.4(ii) is very well defined, and in particular, it is defined more clearly than the minimum in Figure 5.4(i) for $S_k(\tilde{f}, \alpha_o \tilde{g})$.

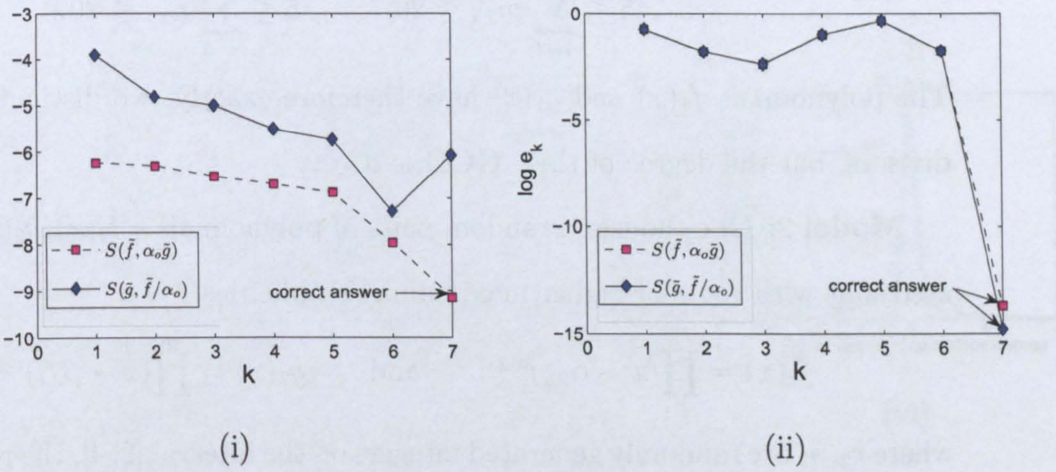


Figure 5.4: The degree of an approximate GCD calculated by (i) the residuals (5.25) and (5.28), and (ii) the error measure e_k, with $\varepsilon_c = 10^{-8}$.

The criterion (5.29) for the calculation of the degree of an approximate GCD of $\tilde{f}(y)$ and $\tilde{g}(y)$ is based on $res_{k,1}$ and $res_{k,2}$. A large number of computational experiments showed, however, that if the degree of an approximate GCD of $\tilde{f}(y)$ and $\tilde{g}(y)$ is computed based on e_k, then the error measures obtained from $S_k(\tilde{f}, \alpha_o \tilde{g})$ are very similar to the error measures obtained from $S_k(\tilde{g}, \tilde{f}/\alpha_o)$. $\square$

Example 5.4. Consider one thousand pairs of the polynomials defined from Model 1 and Model 2, respectively.

Model 1: One thousand random pairs of polynomials $\left\{\hat{f}_1(x), \hat{g}_1(x)\right\}$ were generated,

$$\hat{f}_1(x) = \prod_{i=1}^{r_1}(x-\alpha_{1,i})^{m_{1,i}} \qquad \text{and} \qquad \hat{g}_1(x) = \prod_{i=1}^{s_1}(x-\beta_{1,i})^{n_{1,i}},$$

where r_1, s_1 are randomly generated integers on the interval $[2, 4]$, the roots $\alpha_{1,1}, \ldots, \alpha_{1,r_1}$ are arbitrary, $\beta_{1,1} = \alpha_{1,1}, \beta_{1,2} = \alpha_{1,2}$, $\beta_{1,3}, \ldots, \beta_{1,s_1}$ are arbitrary, and

$$\begin{array}{cc} -10 \leq \alpha_{1,i}, \beta_{1,i} \leq 10, & 1 \leq m_{1,i}, n_{1,i} \leq 6, \\ 5 \leq \sum_{i=1}^{r_1} m_{1,i} \leq 20, & 5 \leq \sum_{i=1}^{s_1} n_{1,i} \leq 20. \end{array}$$

The polynomials $\hat{f}_1(x)$ and $\hat{g}_1(x)$ have therefore exactly two distinct common linear divisors, but the degree of their GCD is $\hat{d} \geq 2$.

Model 2: One thousand random pairs of polynomials $\left\{\hat{f}_2(x), \hat{g}_2(x)\right\}$ were generated, but with roots of higher maximum multiplicities,

$$\hat{f}_2(x) = \prod_{i=1}^{r_2}(x-\alpha_{2,i})^{m_{2,i}} \qquad \text{and} \qquad \hat{g}_2(x) = \prod_{i=1}^{s_2}(x-\beta_{2,i})^{n_{2,i}},$$

where r_2, s_2 are randomly generated integers on the interval $[2, 4]$, the roots $\alpha_{2,1}, \ldots, \alpha_{2,r_2}$ are arbitrary, $\beta_{2,1} = \alpha_{2,1}, \beta_{2,2} = \alpha_{2,2}$, $\beta_{2,3}, \ldots, \beta_{2,s_2}$ are arbitrary, and

$$\begin{array}{cc} -10 \leq \alpha_{2,i}, \beta_{2,i} \leq 10, & 1 \leq m_{2,i}, n_{2,i} \leq 11, \\ 5 \leq \sum_{i=1}^{r_2} m_{2,i} \leq 35, & 5 \leq \sum_{i=1}^{s_2} n_{2,i} \leq 35. \end{array}$$

The polynomials $\hat{f}_2(x)$ and $\hat{g}_2(x)$ have therefore exactly two distinct common linear divisors, but the degree of their GCD is $\hat{d} \geq 2$.

Noise is added to each of these 4000 polynomials, corresponding to a componentwise signal-to-noise ratio of 10^8, and these inexact polynomials are then preprocessed by the operations described in Section 4.4. The error between the degree $\hat{d}$ of the GCD of theoretically exact polynomials, and the degree d of an approximate GCD of their inexact forms, is computed from the 2000 random pairs of polynomials, and the results are shown in Figure 5.5.

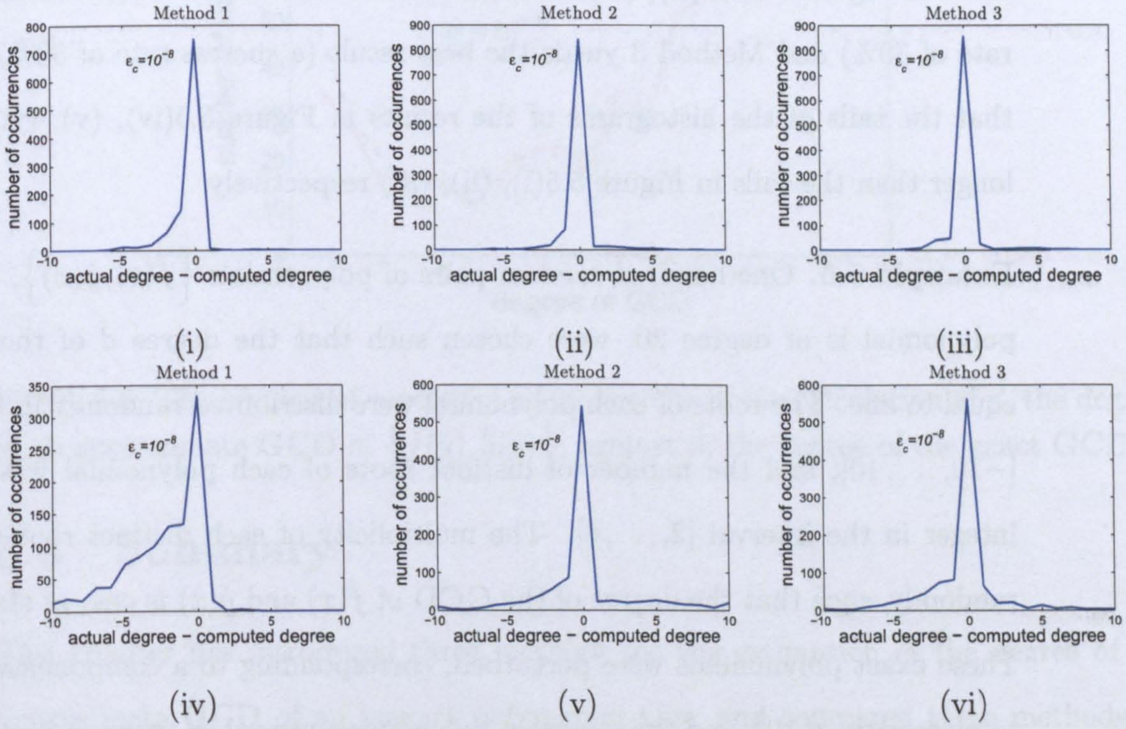


Figure 5.5: Histograms of the results for 1000 pairs of the polynomials using (a) Model 1, graphs (i), (ii) and (iii), and (b) Model 2, graphs (iv), (v) and (vi).

Figure 5.5(i), (ii), (iii) show the results for the polynomials in Model 1, and Figure 5.5(iv), (v), (vi) show the results for the polynomials in Model 2, using Methods 1, 2 and 3, in which Methods 1, 2 and 3 use (5.7), (5.11) and (5.34), respectively, to calculate the degree d of an approximate GCD of these polynomials. For the polynomials in Model 1, Method 1 correctly calculates the degree d on 70% of the 1000 experiments, and Method 3 correctly calculates the degree d on 85% the 1000 experiments.

The polynomials in Model 2 provide a more stringent test than do the polynomials in Model 1 because the multiplicities of the roots, and the total degree of the polynomials are larger. The results in Figure 5.5(iv), (v), (vi) follow the same pattern as those in Figure 5.5(i), (ii), (iii), because Method 1 yields the worst results (a success rate of 30%) and Method 3 yields the best result (a success rate of 50%). It is seen that the tails of the histograms of the results in Figure 5.5(iv), (v), (vi) are much longer than the tails in Figure 5.5(i), (ii), (iii) respectively. $\square$

Example 5.5. One hundred random pairs of polynomials $\left\{\hat{f}(x), \hat{g}(x)\right\}$, where each polynomial is of degree 20, were chosen such that the degree $\hat{d}$ of their GCD is equal to one. The roots of each polynomial were distributed randomly in the interval $[-10, \ldots, 10]$, and the number of distinct roots of each polynomial was a random integer in the interval $[2, \ldots, 6]$. The multiplicity of each distinct root was chosen randomly, such that the degree of the GCD of $\hat{f}(x)$ and $\hat{g}(x)$ is one, as stated above. These exact polynomials were perturbed, corresponding to a componentwise signal-to-noise ratio of 10^8, and these inexact polynomials were then preprocessed in order to transform them to the scaling forms as $\left\{\tilde{f}(y), \tilde{g}(y)\right\}$. The degree d of an approximate GCD of each pair of these inexact polynomials was computed by Methods 2 and 3.

The experiment is repeated for $\hat{d} = 2, 3, \ldots, 19$, and the results are shown in Figure 5.6. It is seen that both methods yield similar results and the probability of correctly computing d increases as $\hat{d}$ increases. This figure provides more detail than the histograms in Figure 5.5 because it shows that the success of Methods 2 and 3 is dependent upon the degree $\hat{d}$ of the theoretically exact GCD. $\square$

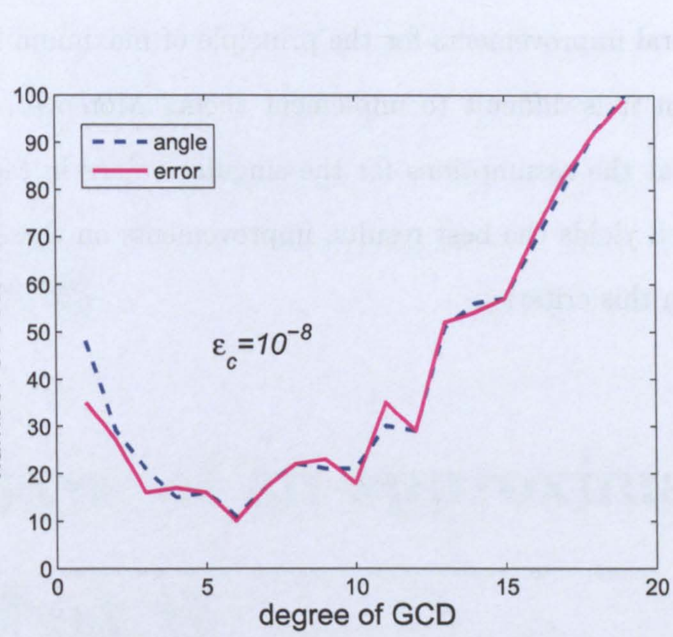


Figure 5.6: The number of successful computations of the calculation of d, the degree of an approximate GCD of $\left\{\tilde{f}(y), \tilde{g}(y)\right\}$, against $\hat{d}$, the degree of the exact GCD.

5.5 Summary

This chapter has introduced three methods for the estimation of the degree of an approximate GCD of an inexact polynomial pair, and compared these methods in several examples. The results suggest that Method 1, which uses the principle of maximum likelihood, yields the worst results, and Method 3, which is based on the error measure e_k, yields the best results. A possible explanation for this difference is that the principle of maximum likelihood is a general method, that is, it does not explicitly exploit the properties of a resultant matrix. By contrast, Method 2 exploits the partitioned structure of $S_k(f, g)$, and Method 3 exploits the polynomial nature of the computations such that the non-rational form of an approximate GCD is imposed as a constraint.

Several improvements for the principle of maximum likelihood have been considered, but it is difficult to implement them. Moreover, computational experiments show that the assumptions for the singular values in Method 1 are not true. Since Method 3 yields the best results, improvements on these results should therefore be based on this criterion.

Chapter 6

The degree of an approximate GCD, Part II

Chapter 5 has compared three methods for the estimation of the degree of an approximate GCD of an inexact polynomial pair, and Method 3, which is based on the error measure e_k, yields the best results. Also, it is seen from Example 5.3 that the degree of an approximate GCD computed from $S_k(f, g)$ may not be equal to the degree of an approximate GCD computed from $S_k(g, f)$, depending on the criterion used. It is necessary that a method for the estimation of the degree of an approximate GCD of an inexact polynomial pair is independent of the order of the polynomials ((f, g) or (g, f)), and thus Method 3 must be extended in order that this requirement be satisfied. Moreover, Example 5.4 shows that the polynomials in Model 2 are computationally more difficult than the polynomials in Model 1, and the results for the polynomials in Model 2 are inferior with respect to the results for the polynomials in Model 1 for Methods 2 and 3. Hence, Methods 2 and 3 must be developed so that they yield better results for the polynomials in Model 2.

This chapter extends the work in Methods 2 and 3, and describes another two methods for the calculation of the degree of an approximate GCD of an inexact polynomial pair $f(x)$ and $g(x)$, such that knowledge of the noise level is not required, and assumptions of the singular values of the Sylvester resultant matrix and its subresultant matrices are not made. All parameters in the methods are therefore calculated directly from the coefficients of $f(x)$ and $g(x)$, which is an advantage. One method uses the first principal angle between a line and a hyperplane, the equations of which are calculated from $S_k(f,g)$, and the other method uses the residual of a linear algebraic equation whose coefficient matrix and right hand side vector are derived from $S_k(f,g)$. Furthermore, one more method that expands these two new methods is developed to calculate the degree of an approximate GCD of an inexact polynomial $f(x)$ and its derivative $f^{(1)}(x)$.

6.1 The degree of an approximate GCD of $f(x)$ and $g(x)$

The preprocessing operations discussed in Section 4.4 transform the given inexact polynomials $f(x)$ and $g(x)$ to $\tilde{f}(y)$ and $\tilde{g}(y)$, which are defined in (4.24), and all computations are performed on these polynomials.

As mentioned in Section 4.1.1, when exact polynomials are specified, (4.7) possesses at least one solution if $k \leq \hat{d}$, where $\hat{d}$ is the degree of the theoretically exact GCD, otherwise, there do not exist a solution for (4.7), and thus $\hat{d}$ is equal to the largest value of k for which (4.7) possesses a solution. This situation is, however, significantly more complicated when inexact data is specified.

It is known from Section 5.3 that the matrices $S_k(\tilde{f}, \alpha_o\tilde{g})$ have full rank for $k = 1, \ldots, \min(m, n)$ due to the coprime polynomial pair $\tilde{f}(y)$ and $\tilde{g}(y)$, and thus (4.7) does not possess one or more exact solutions, and it therefore reduces to the approximation,

$$H_k x_k \approx h_k, \qquad k = 1, \ldots, \min(m, n). \tag{6.1}$$

Equation (6.1) requires that the first column of $S_k(\tilde{f}, \alpha_o\tilde{g}), k = 1, \ldots, \min(m, n)$, be defined as the right hand vector h_k. This approach is adequate when exact polynomials are used, but a modification to this approach is required when the inexact polynomials are specified. In particular, it is assumed that $S_k(\tilde{f}, \alpha_o\tilde{g})$ has full rank, and thus its columns are linearly independent, that is, there does not exist a column of $S_k(\tilde{f}, \alpha_o\tilde{g})$ that lies in the column space of the remaining $m + n - 2k + 1$ columns, and it is therefore necessary to perturb the matrix H_k and vector h_k such that (6.1) is an equation and not an approximation. These perturbations are calculated by the method of structured nonlinear total least norm (SNTLN) [57], which will be described in the following chapter.

Figure 5.4(i) in Example 5.3 shows that choosing different columns to move to the right hand side of (6.1) leads to different results, which is incorrect because the result of a GCD computation, or an approximate GCD computation must be independent of the order in which the polynomials are specified. This problem is therefore overcome by selecting the best column of $S_k(\tilde{f}, \alpha_o\tilde{g})$ to move to the right hand side, rather than insisting that the first column be moved. This requirement for the best column implies that the same result is obtained for the polynomial pairs $(\tilde{f}, \tilde{g})$ and $(\tilde{g}, \tilde{f})$.

The smallest error in the approximation (6.1) for each value of $k = 1, \ldots, \min(m, n)$, is achieved by choosing h_k as the column of $S_k(\tilde{f}, \alpha_o\tilde{g})$ such that the angle between this column and the space spanned by the remaining $m + n - 2k + 1$ columns is a minimum, which means the smaller the angle, the smaller the error in the approximation (6.1). An alternative method requires the residual of the approximation be considered. It is therefore necessary to extend (6.1) from the selection of the first column of $S_k(\tilde{f}, \alpha_o\tilde{g})$ to an arbitrary column, where the optimal column for each $k = 1, \ldots, \min(m, n)$ yields the smallest error. Equation (6.1) is therefore written as

$$H_{k,i}x_{k,i} \approx h_{k,i}, \qquad k = 1, \ldots, \min(m, n), i = 1, \ldots, m + n - 2k + 2, \tag{6.2}$$

where $h_{k,i}$ is the ith column of $S_k(\tilde{f}, \alpha_o\tilde{g})$, $H_{k,i}$ is the matrix from the remaining $m + n - 2k + 1$ columns of $S_k(\tilde{f}, \alpha_o\tilde{g})$,

$$H_{k,i} = \left[\begin{array}{cccccc} h_{k,1} & \cdots & h_{k,i-1} & h_{k,i+1} & \cdots & h_{k,m+n-2k+2} \end{array} \right].$$

It is noted that $h_{k,i} = h_{k,i}(\tilde{f})$ or $h_{k,i} = h_{k,i}(\alpha_o\tilde{g})$, depending on the value of i, and that $H_{k,i} = H_{k,i}(\tilde{f}, \alpha_o\tilde{g})$.

Suppose that, for a given value of k, the i^*th column of $S_k(\tilde{f}, \alpha_o\tilde{g})$ is the optimal column that is moved to the right hand side of (6.2). Since $i^* = i^*(k)$ is a function of k, that is, different values of k yield different optimal columns, the substitution of $i = i^*$ into (6.2) becomes

$$H_{k,i^*}x_{k,i^*} \approx h_{k,i^*}, \qquad k = 1, \ldots, \min(m, n), \tag{6.3}$$

such that the angle between the space spanned by h_{k,i^*} and the space spanned by the columns of H_{k,i^*} is minimised for each value k.

Let d be the degree of an approximate GCD of $\tilde{f}(y)$ and $\tilde{g}(y)$. Computational experiments showed that the angle between the space spanned by h_{k,i^*} and the space spanned by the columns of H_{k,i^*} for $k = 1, \ldots, d$, is much smaller than the angle for $k = d+1, \ldots, \min(m,n)$, because (6.3) yields an unacceptable large angle when $k > d$. The degree d is therefore given by the index k for which the change in the angle of (6.3), between two successive values of k, is a maximum.

Similarly, for a given value of k, the index of optimal column is equal to i^* for which the residual of (6.3) is minimised. Also, the residual of (6.3) is relatively small for $k = 1, \ldots, d$, but it is relatively large for $k = d+1, \ldots, \min(m,n)$.

Since the calculation of the optimal values of the indices i and k motivates the calculation of the degree d, there are two issues that must be addressed:

(a) The calculation of the index $i = i^*$ of the column of $S_k(\tilde{f}, \alpha_o \tilde{g})$ that defines the column h_{k,i^*} in (6.3) for each value of k.

(b) The calculation of the degree $k = d$ of an approximate GCD of $\tilde{f}(y)$ and $\tilde{g}(y)$.

Two methods, based on the first principal angle and the residual of (6.2),

- Method 4: The method of the first principal angle,
- Method 5: The method of residual,

can be used to solve this problem, and this is considered in Section 6.1.1 and 6.1.2 respectively.

6.1.1 Method 4: The method of the first principal angle

Consider initially a method based on the first principal angle $\phi_{k,i}$, which has been introduced in Section 5.2, that is, the smallest angle, between the space $\mathcal{L}_{k,i}$ spanned by $h_{k,i}$, and the space $\mathcal{H}_{k,i}$ spanned by the columns of $H_{k,i}$,

$$\phi_{k,i} = \angle(\mathcal{L}_{k,i}, \mathcal{H}_{k,i}), \qquad k = 1, \ldots, \min(m,n), i = 1, \ldots, m+n-2k+2, \tag{6.4}$$

where

$$\begin{aligned}\mathcal{L}_{k,i} &= \text{span}\{\ h_{k,i}\ \},\\ \mathcal{H}_{k,i} &= \text{span}\{\ h_{k,1}\ \cdots\ h_{k,i-1}\ \ h_{k,i+1}\ \cdots\ h_{k,m+n-2k+2}\ \}.\end{aligned}$$

It is stated in Section 6.1 that the indices i and k are computed by this method in two stages. Firstly, the minimum value ϕ_k of $\phi_{k,i}$ for each value of k is computed,

$$\phi_k = \min_i \{\phi_{k,i} : i = 1, \ldots, m+n-2k+2\}, \quad k = 1, \ldots, \min(m,n), \tag{6.5}$$

and the column index i^* for each of the $\min(m,n)$ minima occurs is recorded as $p_k = i^*$ for each $k = 1, \ldots, \min(m,n)$ respectively. Since it is known from Section 6.1 that the angle ϕ_k for $k = 1, \ldots, d$, is much smaller than ϕ_k for $k = d+1, \ldots, \min(m,n)$, the degree $d = d_\phi$ of an approximate GCD is equal to the index k for which the change in ϕ_k between two successive values of k is a maximum,

$$d_\phi = \{k : \max(\phi_{k+1} - \phi_k); k = 1, \ldots, \min(m,n) - 1\}. \tag{6.6}$$

Equation (6.6) defines the criterion for the calculation of d_ϕ, but an expression for $\phi_{k,i}$, which is defined in (6.4), must be obtained. Moreover, the procedure for the calculation of the first principal angle between a line and a subspace is similar to the calculation of the first principal angle between two subspaces, which is shown in Section 5.2.1 and 5.2.2. The following analysis therefore reproduces these sections for the special case of the angle between a line and hyperplane.

An orthonormal basis for $\mathcal{H}_{k,i}$ is required, and this is obtained by applying the QR decomposition to $H_{k,i}$,

$$H_{k,i} = O_{k,i}R_{k,i}, \qquad O_{k,i}^T O_{k,i} = I_{m+n-2k+1},$$

where $O_{k,i} \in \mathbb{R}^{(m+n-k+1)\times(m+n-2k+1)}$, $R_{k,i} \in \mathbb{R}^{(m+n-2k+1)\times(m+n-2k+1)}$ is an upper triangular matrix, and columns of $O_{k,i}$ define an orthonormal basis for $\mathcal{H}_{k,i}$. Every vector

$t_{k,i} \in \mathcal{H}_{k,i}$ can be therefore written as

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

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

$$s_{k,i} = \frac{h_{k,i}}{\|h_{k,i}\|} \in \mathcal{L}_{k,i}, \qquad \dim \mathcal{L}_{k,i} = 1,$$

and $t_{k,i}$, and thus

$$\cos\phi_{k,i} = \max_{\|t_{k,i}\|=1} s_{k,i}^T t_{k,i} = \max_{\|w_{k,i}\|=1} (s_{k,i}^T O_{k,i}) w_{k,i}. \tag{6.7}$$

If the SVD of $s_{k,i}^T O_{k,i}$ is equal to $\Sigma_{k,i} W_{k,i}^T$, where $W_{k,i}$ is an orthogonal matrix of order $m+n-2k+1$, and

$$\Sigma_{k,i} = \begin{bmatrix} \varsigma_{k,i,1} & 0 & \cdots & 0 \end{bmatrix} \in \mathbb{R}^{m+n-2k+1},$$

then (6.7) yields

$$\cos\phi_{k,i} = \max_{\|t_{k,i}\|=1} s_{k,i}^T t_{k,i} = \max_{\|w_{k,i}\|=1} (\Sigma_{k,i} W_{k,i}^T) w_{k,i},$$

which implies that $\cos\phi_{k,i}$ is equal to the non-zero singular value of $s_{k,i}^T O_{k,i}$,

$$\cos\phi_{k,i} = \varsigma_{k,i,1}. \tag{6.8}$$

This maximum is attained when $w_{k,i}$ is equal to the first column of $W_{k,i}$.

It was shown in Section 5.2.1 and 5.2.2 that computational problems arise when $\phi_{k,i} \approx 0$ due to (5.16), and thus (6.8) cannot be used to calculate the first principal angle when it is small. As shown in Section 5.2.2, this problem is solved by considering the orthonormal complements $\mathcal{L}_{k,i}^{\perp}$ and $\mathcal{H}_{k,i}^{\perp}$, where

$$\mathcal{L}_{k,i} \cup \mathcal{L}_{k,i}^{\perp} = \mathbb{R}^{m+n-k+1} \qquad \text{and} \qquad \mathcal{H}_{k,i} \cup \mathcal{H}_{k,i}^{\perp} = \mathbb{R}^{m+n-k+1},$$

and

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

It will be necessary to calculate orthonormal bases for $\mathcal{L}_k^{\perp}$ and $\mathcal{H}_k^{\perp}$, and these bases will define the columns of matrices $\overline{s}_{k,i} \in \mathbb{R}^{(m+n-k+1)\times(m+n-k)}$ and $\overline{O}_{k,i} \in \mathbb{R}^{(m+n-k+1)\times k}$, respectively.

If the SVD of $O_{k,i}$ is $U_{k,i}S_{k,i}V_{k,i}^T$, where $U_{k,i} \in \mathbb{R}^{(m+n-k+1)\times(m+n-k+1)}$, $S_{k,i} \in \mathbb{R}^{(m+n-k+1)\times(m+n-2k+1)}$, $V_{k,i} \in \mathbb{R}^{(m+n-2k+1)\times(m+n-2k+1)}$, then the last k columns of the left singular matrix $U_{k,i}$ of $O_{k,i}$ provide an orthonormal bases for $\overline{O}_{k,i}$ according to (5.21). Similarly, the matrix $\overline{s}_{k,i}$ is defined by the last $m+n-k$ columns of the left singular matrix of $s_{k,i}$.

It follows from Theorems 5.2.2 and 5.2.3 that the non-zero singular value of $s_{k,i}^T\overline{O}_{k,i} \in \mathbb{R}^k$ is equal to the smallest non-zero singular value of $\overline{s}_{k,i}^T O_{k,i} \in \mathbb{R}^{(m+n-k)\times(m+n-2k+}$

$$\sigma_{k,i,1} = \sqrt{1-\varsigma_{k,i,1}^2} = \sqrt{1-\cos^2\phi_{k,i}} = \sin\phi_{k,i},$$

and thus $\phi_{k,i}$ is obtained from

$$\phi_{k,i} = \sin^{-1}\sigma_{k,i,1}.$$

6.1.2 Method 5: The method of residual

Another method for the calculation of indices k and i can be performed by considering the residual $r_{k,i} = r_{k,i}(H_{k,i}, h_{k,i})$ of (6.2). Let $z_{k,i}$ be the least squares approximate solution of (6.2),

$$r_{k,i} = h_{k,i} - H_{k,i}z_{k,i}, \quad z_{k,i} = H_{k,i}^{\dagger}h_{k,i}, \quad H_{k,i}^{\dagger} = (H_{k,i}^T H_{k,i})^{-1}H_{k,i}^T,$$

for $k = 1, \ldots, \min(m,n), i = 1, \ldots, m+n-2k+2$, where

$$\|r_{k,i}\|^2 + \|H_{k,i}z_{k,i}\|^2 = \|h_{k,i}\|^2, \qquad r_{k,i}^T(H_{k,i}z_{k,i}) = 0,$$

which is shown in Figure 6.1.

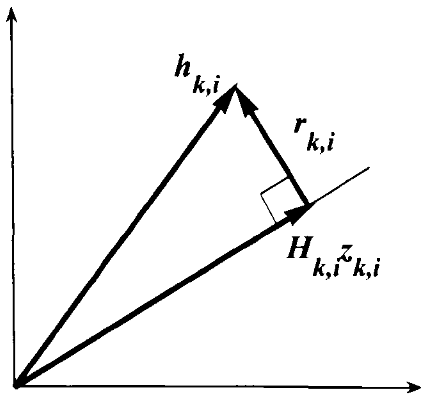


Figure 6.1: Geometry of the least square problem.

It follows that $\|r_{k,i}\|$ is equal to the perpendicular distance of the point with position vector $h_{k,i}$ to the point with position vector $H_{k,i}z_{k,i}$ on the plane $t = H_{k,i}x_{k,i}$, which defines the column space of $H_{k,i}$.

The procedure for the method based on residual is similar to the method based on the first principal angle, which is defined in Section 6.1.1, and thus the minimum value of $\|r_{k,i}\|$ for each value of $k = 1, \ldots, \min(m,n)$, is calculated,

$$r_k = \min_i \left\{ \|r_{k,i}\| : i = 1, \ldots, m+n-2k+2 \right\}, \quad k = 1, \ldots, \min(m,n), \tag{6.9}$$

and the column index i^* for each of the $\min(m,n)$ minima occurs is recorded as $q_k = i^*$ for each $k = 1, \ldots, \min(m,n)$ respectively. As above, the degree d_r of an approximate GCD is equal to the index k for which the change in r_k between two successive values of k is a maximum,

$$d_r = \left\{ k : \max(r_{k+1} - r_k); k = 1, \ldots, \min(m,n) - 1 \right\}. \tag{6.10}$$

It is important to note that Methods 4 and 5 may not yield the same optimal column for some values of k, and the degree d. This issue must be investigated computationally, and this is shown in Section 6.3. Algorithm 6.1 shows the implementation of Methods 4 and 5 for the calculation of (p_k, d_ϕ) and (q_k, d_r).

Algorithm 6.1: The calculation of the degree of an approximate GCD of two polynomials

Input Two inexact polynomials $f(x)$ and $g(x)$.

Output Two estimates, d_ϕ and d_r, of the degree of an approximate GCD of $f(x)$ and $g(x)$, and the column indices p_k and q_k associated with the smallest angle and residual respectively, for each value of k.

```
Begin

1. Preprocess $f(x)$ and $g(x)$ to yield the polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$, as shown in
   Section 4.4.

2. For $k = 1, \ldots, \min(m, n)$     % Loop for all the subresultant matrices

        For $i = 1, \ldots, m + n - 2k + 2$     % Loop for the columns

            (i) Define the column $h_{k,i}$ from $S_k(\tilde{f}, \alpha_0\tilde{g})$.

            (ii) Define the matrix $H_{k,i}$ from $S_k(\tilde{f}, \alpha_0\tilde{g})$.

            (iii) Calculate the angle $\phi_{k,i}$ and residual $r_{k,i}$.

        End $i$

        2.1 Calculate $\phi_k$ and $p_k$ from (6.5), and $r_k$ and $q_k$ from (6.9).

   End $k$
```

3. Calculate two estimates d_ϕ and d_r of the degree of an approximate GCD from (6.6) and (6.10).

End

6.2 The degree of an approximate GCD of $f(x)$ and $f^{(1)}(x)$

It was shown in Chapter 3 that the polynomial root solver requires that an approximate GCD of a polynomial and its derivative be calculated, where the calculation of the degree of an approximate GCD is essential to the calculation of an approximate GCD. Although the calculation of the degree of an approximate GCD of $f(x)$ and $g(x)$ was discussed in Methods $1, 2, 3, 4$ and 5, which can also be applied to the calculation of the degree d of an approximate GCD of $f(x)$ and its derivative $g(x) = f^{(1)}(x)$, these methods did not include the constraint that an approximate GCD of a polynomial and its derivative is being calculated. This section therefore extends the analysis in Methods 4 and 5 to the situation when $g(x) = f^{(1)}(x)$ and considers an extra condition that arises from this constraint.

Let $f(x)$ be an inexact polynomial, and $g(x)$ be equal to its derivative $f^{(1)}(x)$, which is given by

$$f(x) = \sum_{i=0}^{m} a_i x^{m-i} \qquad \text{and} \qquad g(x) = \sum_{i=0}^{m-1} (m-i) a_i x^{m-i-1},$$

It has been shown in Section 4.4 that it is necessary to process $f(x)$ and $g(x)$ before an approximate GCD is computed. In particular, it is required to normalise

$f(x)$ and $g(x)$ by the geometric means of their coefficients, and $\bar{f}(x)$ is therefore redefined as

$$\bar{f}(x)=\sum_{i=0}^{m}\bar{a}_i x^{m-i}, \quad \bar{a}_i=\frac{a_i}{\left(\prod_{j=0}^{m}|a_j|\right)^{\frac{1}{m+1}}}, \quad \prod_{i=0}^{m}|\bar{a}_i|=1, \tag{6.11}$$

and $\bar{g}(x)$ is redefined as

$$\bar{g}(x)=\sum_{i=0}^{m-1}\bar{b}_i x^{m-i-1}, \quad \bar{b}_i=\frac{(m-i)a_i}{\left(\prod_{j=0}^{m-1}|(m-j)a_j|\right)^{\frac{1}{m}}}, \quad \prod_{i=0}^{m-1}|\bar{b}_i|=1. \tag{6.12}$$

The polynomial $\bar{g}(x)$ is proportional to, and not equal to, $\bar{f}^{(1)}(x)$ because

$$\bar{f}^{(1)}(x)=\sum_{i=0}^{m-1}(m-i)\bar{a}_i x^{m-i-1} \tag{6.13}$$

and in particular, it follows from (6.12) and (6.13) that

$$\bar{b}_i=\lambda(m-i)\bar{a}_i, \tag{6.14}$$

where

$$\begin{aligned}\lambda=\frac{\bar{b}_i}{(m-i)\bar{a}_i} &= \frac{(m-i)a_i\left(\prod_{j=0}^{m}|a_j|\right)^{\frac{1}{m+1}}}{(m-i)a_i\left(\prod_{j=0}^{m-1}(m-j)\prod_{j=0}^{m-1}|a_j|\right)^{\frac{1}{m}}}\\ &= \frac{\left(\prod_{j=0}^{m}|a_j|\right)^{\frac{1}{m+1}}}{\left(\frac{m!}{|a_m|}\prod_{j=0}^{m}|a_j|\right)^{\frac{1}{m}}}\\ &= \frac{\left(\frac{|a_m|}{m!}\right)^{\frac{1}{m}}}{\left(\prod_{j=0}^{m}|a_j|\right)^{\frac{1}{m(m+1)}}},\end{aligned} \tag{6.15}$$

and hence λ is equal to a constant, such that

$$\bar{g}(x)=\lambda\bar{f}^{(1)}(x). \tag{6.16}$$

If the approximate quotient polynomials $u_k(x)$ and $v_k(x)$, and an approximate common divisor polynomial $c_k(x)$ of degree k, of the inexact polynomials $\bar{f}(x)$ and

$\bar{g}(x)$ are given by

$$u_k(x) = \sum_{i=0}^{m-k} u_{k,i} x^{m-k-i}, \quad v_k(x) = \sum_{i=0}^{m-k-1} v_{k,i} x^{m-k-i-1}, \quad c_k(x) = \sum_{i=0}^{k} c_{k,i} x^{k-i}, \tag{6.17}$$

then

$$\bar{f}(x) \approx c_k(x) u_k(x) \qquad \text{and} \qquad \bar{g}(x) \approx c_k(x) v_k(x), \qquad k = 1, \dots, m-1. \tag{6.18}$$

A constraint between the inexact polynomials $\bar{f}(x)$ and $\bar{f}^{(1)}(x)$ yields

$$\begin{aligned} \bar{f}^{(1)}(x) = \frac{\mathrm{d}\bar{f}(x)}{\mathrm{d}x} &\approx \frac{\mathrm{d}\left(c_k(x)u_k(x)\right)}{\mathrm{d}x} \\ &\approx c_k(x)u_k^{(1)}(x) + c_k^{(1)}(x)u_k(x), \quad k = 1, \dots, m-1, \end{aligned} \tag{6.19}$$

where

$$\begin{aligned} c_k^{(1)}(x) &= \sum_{i=0}^{k-1} (k-i) c_{k,i} x^{k-1-i}, \\ u_k^{(1)}(x) &= \sum_{i=0}^{m-k-1} (m-k-i) u_{k,i} x^{m-k-1-i}, \end{aligned}$$

and it follows from (6.16), (6.18) and (6.19) that

$$c_k(x) v_k(x) \approx \lambda \left(c_k(x) u_k^{(1)}(x) + c_k^{(1)}(x) u_k(x) \right), \tag{6.20}$$

which establishes the approximate constraint between $u_k(x)$, $v_k(x)$ and $c_k(x)$ when an approximate GCD of a polynomial and its derivative is considered.

It is also demonstrated in Section 4.4 that scaling polynomials can improve the computational results, and thus the substitution

$$x = \theta y, \tag{6.21}$$

where y is the new independent variable and θ is a real constant, is then made into (6.11) and (6.12). It is therefore necessary to express (6.20) in the independent variable y, that is, the substitution (6.21) has been made. This also requires consideration of the scale factor α. Specifically, the optimal values α_o and θ_o, of α and θ respectively, are the solutions of the minimisation problem,

$$\alpha_{o}, \theta_{o} = \arg\min_{\alpha,\theta} \left\{ \frac{\max\left\{\max_{i=0,\ldots,m}\left|\bar{a}_i\theta^{m-i}\right|, \max_{j=0,\ldots,m-1}\left|\alpha\bar{b}_j\theta^{m-j-1}\right|\right\}}{\min\left\{\min_{i=0,\ldots,m}\left|\bar{a}_i\theta^{m-i}\right|, \min_{j=0,\ldots,m-1}\left|\alpha\bar{b}_j\theta^{m-j-1}\right|\right\}} \right\},$$

which can be transformed to a standard linear programming problem, as shown in (4.23), and thus all computations are performed on the polynomials

$$\tilde{f}(y) = \sum_{i=0}^{m} \tilde{a}_i y^{m-i} \qquad \text{and} \qquad \alpha_{o}\tilde{g}(y) = \alpha_{o} \sum_{i=0}^{m-1} \tilde{b}_i y^{m-i-1}, \tag{6.22}$$

whose coefficients are

$$\tilde{a}_i = \bar{a}_i\theta_{o}^{m-i} \qquad \text{and} \qquad \tilde{b}_i = \bar{b}_i\theta_{o}^{m-i-1}.$$

Since it follows from (6.13) and (6.21) that

$$\tilde{f}^{(1)}(y) = \bar{f}^{(1)}(x = \theta_{o}y) = \sum_{i=0}^{m-1}\left((m-i)\bar{a}_i\theta_{o}^{m-i-1}\right)y^{m-i-1},$$

the relationship between $\alpha_{o}\tilde{g}(y)$ and $\tilde{f}^{(1)}(y)$ is established, based on (6.14), as

$$\alpha_{o}\tilde{g}(y) = (\alpha_{o}\lambda)\tilde{f}^{(1)}(y). \tag{6.23}$$

It is assumed that $f(x)$ is inexact, and thus an approximate common divisor $\tilde{c}_k(y)$ of $\tilde{f}(y)$ and $\tilde{g}(y)$, of degree k, satisfies

$$\tilde{f}(y) \approx \tilde{c}_k(y)\tilde{u}_k(y) \qquad \text{and} \qquad \alpha_{o}\tilde{g}(y) \approx \tilde{c}_k(y)\tilde{v}_k(y), \qquad k = 1, \ldots, m-1, \tag{6.24}$$

where

$$\tilde{u}_k(y) = \sum_{i=0}^{m-k} \tilde{u}_{k,i}y^{m-k-i}, \quad \tilde{u}_{k,i} = u_{k,i}\theta_{o}^{m-k-i}, \tag{6.25}$$

$$\tilde{v}_k(y) = \sum_{i=0}^{m-k-1} \tilde{v}_{k,i}y^{m-k-i-1}, \quad \tilde{v}_{k,i} = v_{k,i}\theta_{o}^{m-k-i-1}, \tag{6.26}$$

$$\tilde{c}_k(y) = \sum_{i=0}^{k} \tilde{c}_{k,i}y^{k-i}, \quad \tilde{c}_{k,i} = c_{k,i}\theta_{o}^{k-i}. \tag{6.27}$$

are the transformed polynomials from (6.17) using (6.21).

A constraint between $\tilde{f}(y)$ and $\tilde{f}^{(1)}(y)$ is completed by the substitution of (6.21) into the expression (6.19), which yields

$$\tilde{f}^{(1)}(y) \approx \tilde{c}_k(y)\tilde{u}_k^{(1)}(y) + \tilde{c}_k^{(1)}(y)\tilde{u}_k(y), \quad k = 1, \ldots, m-1, \tag{6.28}$$

where

$$\begin{aligned} \tilde{c}_k^{(1)}(y) &= \sum_{i=0}^{k-1} \left((k-i)c_{k,i}\theta_o^{k-i-1}\right) y^{k-i-1}, \\ \tilde{u}_k^{(1)}(y) &= \sum_{i=0}^{m-k-1} \left((m-k-i)u_{k,i}\theta_o^{m-k-i-1}\right) y^{m-k-i-1}, \end{aligned}$$

and it therefore follows from (6.23), (6.24) and (6.28) that

$$\tilde{c}_k(y)\tilde{v}_k(y) \approx (\alpha_o\lambda)\left(\tilde{c}_k(y)\tilde{u}_k^{(1)}(y) + \tilde{c}_k^{(1)}(y)\tilde{u}_k(y)\right), \tag{6.29}$$

which establishes the connection between $\tilde{u}_k(y), \tilde{v}_k(y)$ and $\tilde{c}_k(y)$, that is, the connection between the approximate quotient polynomials and an approximate common divisor of degree $k = 1, \ldots, m-1$.

Since the product of two polynomials, which is equal to the convolution of their coefficients, can be written as the product of a Toeplitz matrix and a vector, the vector of coefficients $\tilde{\mathbf{f}}^{(1)}(\theta_o)$ of $\tilde{f}^{(1)}(y)$ in (6.28) can be approximated by

$$\begin{aligned}
\tilde{\mathbf{f}}^{(1)}(\theta_o) &\approx \begin{bmatrix} c_{k,0}\theta_o^k & & \\ c_{k,1}\theta_o^{k-1} & \ddots & \\ \vdots & \ddots & c_{k,0}\theta_o^k \\ c_{k,k-1}\theta_o & \ddots & c_{k,1}\theta_o^{k-1} \\ c_{k,k} & \ddots & \vdots \\ & \ddots & c_{k,k-1}\theta_o \\ & & c_{k,k} \end{bmatrix} \begin{bmatrix} (m-k)u_{k,0}\theta_o^{m-k-1} \\ (m-k-1)u_{k,1}\theta_o^{m-k-2} \\ \vdots \\ 2u_{k,m-k-2}\theta_o \\ u_{k,m-k-1} \end{bmatrix} + \\
&\quad \begin{bmatrix} kc_{k,0}\theta_o^{k-1} & & \\ (k-1)c_{k,1}\theta_o^{k-2} & \ddots & \\ \vdots & \ddots & kc_{k,0}\theta_o^{k-1} \\ 2c_{k,k-2}\theta_o & \ddots & (k-1)c_{k,1}\theta_o^{k-2} \\ c_{k,k-1} & \ddots & \vdots \\ & \ddots & 2c_{k,k-2}\theta_o \\ & & c_{k,k-1} \end{bmatrix} \begin{bmatrix} u_{k,0}\theta_o^{m-k} \\ u_{k,1}\theta_o^{m-k-1} \\ \vdots \\ u_{k,m-k-1}\theta_o \\ u_{k,m-k} \end{bmatrix} \\
&= A_k(\theta_o)\tilde{\mathbf{u}}_k^{(1)}(\theta_o) + B_k(\theta_o)\tilde{\mathbf{u}}_k(\theta_o) \qquad (6.30)
\end{aligned}$$

where

$$\begin{aligned}
\tilde{\mathbf{u}}_k(\theta_o) &= \begin{bmatrix} u_{k,0}\theta_o^{m-k} & u_{k,1}\theta_o^{m-k-1} & \cdots & u_{k,m-k-1}\theta_o & u_{k,m-k} \end{bmatrix}^T \in \mathbb{R}^{m-k+1}, \\
\tilde{\mathbf{u}}_k^{(1)}(\theta_o) &= \Big[(m-k)u_{k,0}\theta_o^{m-k-1} \quad (m-k-1)u_{k,1}\theta_o^{m-k-2} \quad \cdots \\
&\qquad \cdots \quad 2u_{k,m-k-2}\theta_o \quad u_{k,m-k-1} \Big]^T \in \mathbb{R}^{m-k},
\end{aligned}$$

$$A_k(\theta_o) = \begin{bmatrix} c_{k,0}\theta_o^k & & \\ c_{k,1}\theta_o^{k-1} & \ddots & \\ \vdots & \ddots & c_{k,0}\theta_o^k \\ c_{k,k-1}\theta_o & \ddots & c_{k,1}\theta_o^{k-1} \\ c_{k,k} & \ddots & \vdots \\ & \ddots & c_{k,k-1}\theta_o \\ & & c_{k,k} \end{bmatrix} \in \mathbb{R}^{m\times(m-k)},$$

$$B_k(\theta_o) = \begin{bmatrix} kc_{k,0}\theta_o^{k-1} & & \\ (k-1)c_{k,1}\theta_o^{k-2} & \ddots & \\ \vdots & \ddots & kc_{k,0}\theta_o^{k-1} \\ 2c_{k,k-2}\theta_o & \ddots & (k-1)c_{k,1}\theta_o^{k-2} \\ c_{k,k-1} & \ddots & \vdots \\ & \ddots & 2c_{k,k-2}\theta_o \\ & & c_{k,k-1} \end{bmatrix} \in \mathbb{R}^{m\times(m-k+1)}.$$

It is readily verified that $\tilde{\mathbf{u}}_k^{(1)}(\theta_o)$ and $\tilde{\mathbf{u}}_k(\theta_o)$ are related by a diagonal matrix $R \in \mathbb{R}^{(m-k)\times(m-k+1)}$,

$$\begin{aligned}
\theta_o \tilde{\mathbf{u}}_k^{(1)}(\theta_o) &= \begin{bmatrix} (m-k)u_{k,0}\theta_o^{m-k} \\ (m-k-1)u_{k,1}\theta_o^{m-k-1} \\ \vdots \\ 2u_{k,m-k-2}\theta_o^2 \\ u_{k,m-k-1}\theta_o \end{bmatrix} \\
&= \begin{bmatrix} m-k & & & & 0 \\ & m-k-1 & & & 0 \\ & & \ddots & & 0 \\ & & & 2 & 0 \\ & & & 1 & 0 \end{bmatrix} \begin{bmatrix} u_{k,0}\theta_o^{m-k} \\ u_{k,1}\theta_o^{m-k-1} \\ \vdots \\ u_{k,m-k-1}\theta_o \\ u_{k,m-k} \end{bmatrix} \\
&= R\tilde{\mathbf{u}}_k(\theta_o).
\end{aligned} \tag{6.31}$$

It follows from (6.30) and (6.31) that

$$\tilde{\mathbf{f}}^{(1)}(\theta_o) \approx \frac{1}{\theta_o}\Big(A_k(\theta_o)R + \theta_o B_k(\theta_o)\Big)\tilde{\mathbf{u}}_k(\theta_o). \tag{6.32}$$

Similarly, the vector of coefficients $\alpha_o\tilde{\mathbf{g}}(\theta_o)$ of $\alpha_o\tilde{g}(y)$ can be approximated by

$$\begin{aligned}
\alpha_o\tilde{\mathbf{g}}(\theta_o) &\approx \begin{bmatrix} c_{k,0}\theta_o^k & & \\ c_{k,1}\theta_o^{k-1} & \ddots & \\ \vdots & \ddots & c_{k,0}\theta_o^k \\ c_{k,k-1}\theta_o & \ddots & c_{k,1}\theta_o^{k-1} \\ c_{k,k} & \ddots & \vdots \\ & \ddots & c_{k,k-1}\theta_o \\ & & c_{k,k} \end{bmatrix} \begin{bmatrix} v_{k,0}\theta_o^{m-k-1} \\ v_{k,1}\theta_o^{m-k-2} \\ \vdots \\ v_{k,m-k-2}\theta_o \\ v_{k,m-k-1} \end{bmatrix} \\
&= A_k(\theta_o)\tilde{\mathbf{v}}_k(\theta_o),
\end{aligned} \tag{6.33}$$

where

$$\tilde{\mathbf{v}}_k(\theta_o) = \begin{bmatrix} v_{k,0}\theta_o^{m-k-1} & v_{k,1}\theta_o^{m-k-2} & \cdots & v_{k,m-k-2}\theta_o & v_{k,m-k-1} \end{bmatrix}^T \in \mathbb{R}^{m-k}.$$

The combination of (6.23), (6.32) and (6.33) yields

$$\left(\frac{\theta_o}{\alpha_o\lambda}\right) A_k(\theta_o)\tilde{\mathbf{v}}_k(\theta_o) - \left(A_k(\theta_o)R + \theta_o B_k(\theta_o)\right)\tilde{\mathbf{u}}_k(\theta_o) \approx 0. \tag{6.34}$$

If $U_k(\theta_o) \in \mathbb{R}^{m\times(m-k+1)}$ and $V_k(\theta_o) \in \mathbb{R}^{m\times(m-k)}$ are defined as

$$U_k(\theta_o) = \theta_o B_k(\theta_o) = \begin{bmatrix} kc_{k,0}\theta_o^k & & \\ (k-1)c_{k,1}\theta_o^{k-1} & \ddots & \\ \vdots & \ddots & kc_{k,0}\theta_o^k \\ 2c_{k,k-2}\theta_o^2 & \ddots & (k-1)c_{k,1}\theta_o^{k-1} \\ c_{k,k-1}\theta_o & \ddots & \vdots \\ & \ddots & 2c_{k,k-2}\theta_o^2 \\ & & c_{k,k-1}\theta_o \end{bmatrix},$$

and

$$V_k(\theta_o) = \left(\frac{\theta_o}{\alpha_o\lambda}\right) A_k(\theta_o) = \left(\frac{1}{\alpha_o\lambda}\right) \begin{bmatrix} c_{k,0}\theta_o^{k+1} & & \\ c_{k,1}\theta_o^k & \ddots & \\ \vdots & \ddots & c_{k,0}\theta_o^{k+1} \\ c_{k,k-1}\theta_o^2 & \ddots & c_{k,1}\theta_o^k \\ c_{k,k}\theta_o & \ddots & \vdots \\ & \ddots & c_{k,k-1}\theta_o^2 \\ & & c_{k,k}\theta_o \end{bmatrix},$$

respectively, then it follows from (6.34) that

$$\begin{bmatrix} V_k(\theta_o) & A_k(\theta_o)R + U_k(\theta_o) \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{v}}_k(\theta_o) \\ -\tilde{\mathbf{u}}_k(\theta_o) \end{bmatrix} \approx 0, \quad k = 1, \ldots, m-1. \tag{6.35}$$

The constraint (6.35) is therefore used for the calculation of the degree d of an approximate GCD of $f(x)$ and $f^{(1)}(x)$. This uses the error measure

$$\frac{\left\|V_k(\theta_o)\tilde{\mathbf{v}}_k(\theta_o) - \left(A_k(\theta_o)R + U_k(\theta_o)\right) \tilde{\mathbf{u}}_k(\theta_o)\right\|}{\left\|V_k(\theta_o)\tilde{\mathbf{v}}_k(\theta_o)\right\| + \left\|\left(A_k(\theta_o)R + U_k(\theta_o)\right) \tilde{\mathbf{u}}_k(\theta_o)\right\|}, \qquad k = 1, \ldots, m-1, \qquad (6.36)$$

which is derived from (6.35), and the value of k for which this function achieves its minimum value is equal to d.

Consider the calculation of the terms in (6.36), such as the vectors $\tilde{\mathbf{u}}_k(\theta_o)$ and $\tilde{\mathbf{v}}_k(\theta_o)$, and the matrices $A_k(\theta_o), U_k(\theta_o)$ and $V_k(\theta_o)$. Since (6.36) contains terms that require the matrices $S_k(\tilde{f}, \alpha_o\tilde{g})$ for their calculation, where the kth Sylvester resultant matrix $S_k(\tilde{f}, \alpha_o\tilde{g})$ is of order $(2m-k) \times (2m-2k+1)$, and equals to

$$\begin{aligned} S_k(\tilde{f}, \alpha_o\tilde{g}) &= \left[\begin{array}{ccc|ccc} \bar{a}_0\theta_o^m & & & \alpha_o\bar{b}_0\theta_o^{m-1} & & \\ \bar{a}_1\theta_o^{m-1} & \ddots & & \alpha_o\bar{b}_1\theta_o^{m-2} & \ddots & \\ \vdots & \ddots & \bar{a}_0\theta_o^m & \vdots & \ddots & \alpha_o\bar{b}_0\theta_o^{m-1} \\ \bar{a}_{m-1}\theta_o & \ddots & \bar{a}_1\theta_o^{m-1} & \alpha_o\bar{b}_{m-2}\theta_o & \ddots & \alpha_o\bar{b}_1\theta_o^{m-2} \\ \bar{a}_m & \ddots & \vdots & \alpha_o\bar{b}_{m-1} & \ddots & \vdots \\ & \ddots & \bar{a}_{m-1}\theta_o & & \ddots & \alpha_o\bar{b}_{m-2}\theta_o \\ & & \bar{a}_m & & & \alpha_o\bar{b}_{m-1} \end{array}\right] \\ &= \left[\begin{array}{c|c} F_k(\theta_o) & G_k(\alpha_o, \theta_o) \end{array}\right], \end{aligned}$$

it follows from (6.24) that

$$S_k(\tilde{f}, \alpha_o\tilde{g}) \begin{bmatrix} \tilde{\mathbf{v}}_k(\theta_o) \\ -\tilde{\mathbf{u}}_k(\theta_o) \end{bmatrix} = \begin{bmatrix} F_k(\theta_o) & G_k(\alpha_o, \theta_o) \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{v}}_k(\theta_o) \\ -\tilde{\mathbf{u}}_k(\theta_o) \end{bmatrix} \approx 0, \qquad (6.37)$$

and more details in subresultant matrices are in Section 4.1. It is clear that $S_k(\tilde{f}, \alpha_o\tilde{g})$ can be used to calculate $\tilde{\mathbf{u}}_k(\theta_o)$ and $\tilde{\mathbf{v}}_k(\theta_o)$, as well as the degree d of an approximate GCD of $f(x)$ and $g(x) = f^{(1)}(x)$, using Methods 4 and 5, as discussed in Section 6.1. Also it is known that the column indices $i^* = i^*(k)$ associated with the smallest angle and residual for each value of $k = 1, \dots, m-1$, are computed by Methods 4 and 5 respectively.

Assume that h_{k,i^*}, the i^*th column of $S_k(\tilde{f}, \alpha_o \tilde{g})$, is removed from the matrix, where i^* is from either Method 4 or Method 5, and $H_{k,i^*} \in \mathbb{R}^{(2m-k)\times(2m-2k)}$ is the matrix from the other columns of $S_k(\tilde{f}, \alpha_o \tilde{g})$,

$$H_{k,i^*} = \begin{bmatrix} s_{k,1} & \cdots & s_{k,i-1} & s_{k,i+1} & \cdots & s_{k,2m-2k+1} \end{bmatrix},$$

where $s_{k,i} \in \mathbb{R}^{2m-k}$ is the ith column of $S_k(\tilde{f}, \alpha_o \tilde{g})$. The removal of the i^*th column of $S_k(\tilde{f}, \alpha_o \tilde{g})$ to be the right hand side therefore yields the equation

$$H_{k,i^*} x \approx h_{k,i^*}, \quad k = 1, \dots, m-1, \tag{6.38}$$

where

$$x = \begin{bmatrix} x_1 & \cdots & x_{i-1} & x_{i+1} & \cdots & x_{2m-2k+1} \end{bmatrix}^T \in \mathbb{R}^{2m-2k},$$

and

$$\begin{bmatrix} \tilde{\mathbf{v}}_k(\theta_o) \\ -\tilde{\mathbf{u}}_k(\theta_o) \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_{i-1} \\ -1 \\ x_{i+1} \\ \vdots \\ x_{2m-2k+1} \end{bmatrix} \in \mathbb{R}^{2m-2k+1}. \tag{6.39}$$

The elements of $\tilde{\mathbf{u}}_k(\theta_o)$ and $\tilde{\mathbf{v}}_k(\theta_o)$ are calculated from (6.39) with the index i^* for which the minima occurs in (6.38) for each value of k, using Methods 4 or 5. The coefficients $\tilde{c}_{k,i}$ of $\tilde{c}_k(y)$ for the construction of $A_k(\theta_o), U_k(\theta_o)$ and $V_k(\theta_o)$ can be calculated from the approximate polynomial decomposition (6.24), which is written as

$$\begin{bmatrix} P_k(u_k,\theta_o) \\ Q_k(v_k,\theta_o) \end{bmatrix} \tilde{\mathbf{c}}_k(\theta_o) \approx \begin{bmatrix} \tilde{\mathbf{f}}(\theta_o) \\ \alpha_o\tilde{\mathbf{g}}(\theta_o) \end{bmatrix}, \qquad k=1,\ldots,m-1, \tag{6.40}$$

where $P_k(u_k,\theta_o)$ and $Q_k(v_k,\theta_o)$ are Toeplitz matrices whose elements are formed from the polynomials $\tilde{u}_k(y)$ and $\tilde{v}_k(y)$ in (6.26) and (6.27) respectively,

$$P_k(u_k,\theta_o) = \begin{bmatrix} u_{k,0}\theta_o^{m-k} & & \\ u_{k,1}\theta_o^{m-k-1} & \ddots & \\ \vdots & \ddots & u_{k,0}\theta_o^{m-k} \\ u_{k,m-k-1}\theta_o & \ddots & u_{k,1}\theta_o^{m-k-1} \\ u_{k,m-k} & \ddots & \vdots \\ & \ddots & u_{k,m-k-1}\theta_o \\ & & u_{k,m-k} \end{bmatrix} \in \mathbb{R}^{(m+1)\times(k+1)},$$

$$Q_k(v_k,\theta_o) = \begin{bmatrix} v_{k,0}\theta_o^{m-k-1} & & \\ v_{k,1}\theta_o^{m-k-2} & \ddots & \\ \vdots & \ddots & v_{k,0}\theta_o^{m-k-1} \\ v_{k,m-k-2}\theta_o & \ddots & v_{k,1}\theta_o^{m-k-2} \\ v_{k,m-k-1} & \ddots & \vdots \\ & \ddots & v_{k,m-k-2}\theta_o \\ & & v_{k,m-k-1} \end{bmatrix} \in \mathbb{R}^{m\times(k+1)},$$

and

$$\begin{aligned}
\tilde{\mathbf{c}}_k(\theta_o) &= \begin{bmatrix} c_{k,0}\theta_o^k & c_{k,1}\theta_o^{k-1} & \cdots & c_{k,k-1}\theta_o & c_{k,k} \end{bmatrix}^T \\
&= \begin{bmatrix} \tilde{c}_{k,0} & \tilde{c}_{k,1} & \cdots & \tilde{c}_{k,k-1} & \tilde{c}_{k,k} \end{bmatrix}^T \in \mathbb{R}^{k+1}, \\
\tilde{\mathbf{f}}(\theta_o) &= \begin{bmatrix} \bar{a}_0\theta_o^m & \bar{a}_1\theta_o^{m-1} & \cdots & \bar{a}_{m-1}\theta_o & \bar{a}_m \end{bmatrix}^T \\
&= \begin{bmatrix} \tilde{a}_0 & \tilde{a}_1 & \cdots & \tilde{a}_{m-1} & \tilde{a}_m \end{bmatrix}^T \in \mathbb{R}^{m+1}, \\
\tilde{\mathbf{g}}(\theta_o) &= \begin{bmatrix} \bar{b}_0\theta_o^{m-1} & \bar{b}_1\theta_o^{m-2} & \cdots & \bar{b}_{m-2}\theta_o & \bar{b}_{m-1} \end{bmatrix}^T \\
&= \begin{bmatrix} \tilde{b}_0 & \tilde{b}_1 & \cdots & \tilde{b}_{m-2} & \tilde{b}_{m-1} \end{bmatrix}^T \in \mathbb{R}^{m}.
\end{aligned}$$

The least squares solution of (6.40) is computed,

$$\tilde{\mathbf{c}}_k(\theta_o) = \begin{bmatrix} P_k(u_k,\theta_o) \\ Q_k(v_k,\theta_o) \end{bmatrix}^{\dagger} \begin{bmatrix} \tilde{\mathbf{f}}(\theta_o) \\ \alpha_o\tilde{\mathbf{g}}(\theta_o) \end{bmatrix}, \qquad k = 1,\ldots,m-1,$$

which enables the coefficients of the approximate common divisor polynomial $\tilde{c}_k(y)$ to be calculated in order to construct the matrices $A_k(\theta_o), U_k(\theta_o)$ and $V_k(\theta_o)$. This allows (6.36) to be computed for all values of k.

Although Methods 1, 2 and 3 can be used to compute the degree of an approximate GCD of $f(x)$ and $f^{(1)}(x)$, Method 1 fails to return the correct answer in most cases, and Methods 2 and 3 suffer disadvantages, which are improved by Methods 4 and 5. The following three methods, rather than Methods 1, 2 and 3, are therefore used to determine the degree d of an approximate GCD of a polynomial and its derivative:

Method 4 The first principal angle between the space $\mathcal{H}_{k,j}$ spanned by the columns of $H_{k,j}$ and the space $\mathcal{L}_{k,j}$ spanned by $h_{k,j}$.

Method 5 The residual of (6.38).

Method 6 The satisfaction of the constraint (6.35).

Methods 4 and 5 use Algorithm 6.1 to calculate d and the column index i^* for which the minima occurs in (6.38) for each $k = 1, \ldots, m-1$, and Method 6 uses the error measure in (6.36) for the calculation of the degree d. In particular, the error measures in (6.36) are equal to η_k and $\xi_k, k = 1, \ldots, m-1$, when Method 6 uses the optimal columns computed from Methods 4 and 5 respectively. It is important to note that Methods 4 and 5 may not yield the same optimal column for some values of k, and Methods 4, 5 and 6 may not yield the same value of d. This is however interesting because these different values of d are clearly certified, in which case the decision is made by the method called *Majority Voting*. Algorithm 6.2 shows the implementation of Methods 4, 5 and 6 for the calculation of the degree of an approximate GCD of $f(x)$ and $f^{(1)}(x)$ in the presence of noise.

Algorithm 6.2: The calculation of the degree of an approximate GCD of a polynomials and its derivative

Input An inexact polynomial $f(x)$.

Output Four estimates, d_ϕ, d_r, d_η and d_ξ, of the degree of an approximate GCD of $f(x)$ and $f^{(1)}(x)$.

Begin

1. Calculate the first derivative of $f(x)$ as $g(x) = f^{(1)}(x)$, and the constant λ from (6.15) .

2. Preprocess $f(x)$ and $g(x)$ to yield the polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$, as shown in Section 4.4.

```
3. For $k = 1, \dots, m-1$    % Loop for all the subresultant matrices
   (a) For $i = 1, \dots, 2m-2k+1$    % Loop for the columns
         (i) Define the column $h_{k,i}$ and matrix $H_{k,i}$ from $S_k(\tilde{f}, \alpha_o \tilde{g})$.
         (ii) Calculate the angle $\phi_{k,i}$ and the residual $r_{k,i}$.
       End $i$
   (b) Calculate
             $\phi_k = \min_i \{\phi_{k,i} : i = 1, \dots, 2m-2k+1\}$
             $r_k = \min_i \{r_{k,i} : i = 1, \dots, 2m-2k+1\}$
       and the indices $p_k$ and $q_k$ for which the minima occur for each value of $k$,
       respectively.
   (c) Form $H_{k,i}$ and $h_{k,i}$ when $i = p_k$, and solve, in the least squares sense,
       (6.38).
   (d) Compute $\tilde{\mathbf{u}}_k(\theta_o)$ and $\tilde{\mathbf{v}}_k(\theta_o)$ from (6.39) and $\tilde{\mathbf{c}}_k(\theta_o)$ from (6.40).
   (e) Construct the matrices $R, A_k(\theta_o), U_k(\theta_o)$ and $V_k(\theta_o)$, and calculate the er-
       ror measure $\eta_k$ in (6.36).
   (f) Repeat steps (c), (d) and (e) when $i = q_k$, and calculate the error measure
       $\xi_k$ in (6.36).
   End $k$
4. Calculate four estimates of the degree of an approximate GCD of $f(x)$ and
```

$g(x)$, namely,

$$\begin{aligned} d_\phi &= \{k : \max(\phi_{k+1} - \phi_k);\ k = 1, \ldots, m-2\} \\ d_r &= \{k : \max(r_{k+1} - r_k);\ k = 1, \ldots, m-2\} \\ d_\eta &= \min_k \{\eta_k;\ k = 1, \ldots, m-1\} \\ d_\xi &= \min_k \{\xi_k;\ k = 1, \ldots, m-1\} \end{aligned}$$

End

6.3 Examples

This section contains several examples in which the degree of an approximate GCD of $f(x)$ and $g(x)$ is calculated using Methods 4 and 5, and the degree of an approximate GCD of $f(x)$ and $f^{(1)}(x)$ is calculated using Methods 4, 5 and 6.

Example 6.1. Consider the exact polynomials $\hat{f}(x)$ and $\hat{g}(x)$, whose roots and multiplicities are specified in Table 6.1. It is seen that $m = 16, n = 21$ and the degree of their GCD is $\hat{d} = 7$.

Root of $\hat{f}(x)$	Multiplicity
4.8181e+000	3
-2.9457e+000	2
-8.5379e+000	2
-1.3787e-002	9

Root of $\hat{g}(x)$	Multiplicity
4.8181e+000	8
-2.9457e+000	5
-8.5379e+000	8

Table 6.1: The roots and multiplicities of $\hat{f}(x)$ and $\hat{g}(x)$ for Example 6.1.

Each polynomial was perturbed by noise, such that the componentwise signal-to-noise ratio ε_c^{-1} is 10^8, and the resulting polynomials were normalised by the geometric means of their coefficients. They were then preprocessed, thereby yielding the Sylvester matrix $S(\tilde{f}, \alpha_o \tilde{g})$, where $\alpha_o = 0.065453, \theta_o = 0.89125$. This procedure was repeated, thereby obtaining another set of perturbed polynomials, using $\varepsilon_c^{-1} = 10^4$, in which case $\alpha_o = 0.065449, \theta_o = 0.89126$.

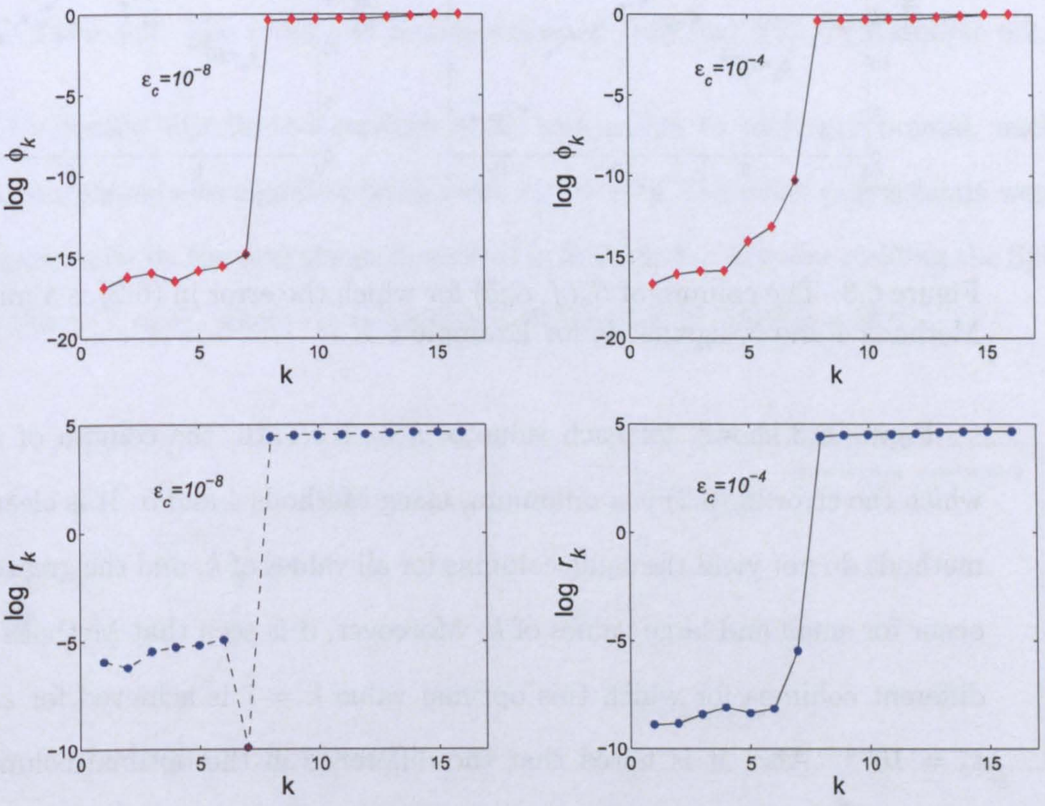


Figure 6.2: The variation of $\log \phi_k$ and $\log r_k$ with k for Example 6.1.

Figure 6.2 shows the variation of $\log \phi_k$ and $\log r_k$, which are defined in (6.5) and (6.9) respectively, with k. It is seen that for $\varepsilon_c = 10^{-8}$ and $\varepsilon_c = 10^{-4}$, the maximum changes in $\log \phi_k$ and $\log r_k$ occur when $k = 7$, which is correct because $\hat{d} = 7$.

Although the values of d_ϕ and d_r are clearly defined, the changes in $\log \phi_k$ and $\log r_k$ at $k = 7$ are much bigger when $\varepsilon_c = 10^{-8}$ compared to those changes when $\varepsilon_c = 10^{-4}$ because the former perturbations to the coefficients of $\hat{f}(x)$ and $\hat{g}(x)$ are smaller.

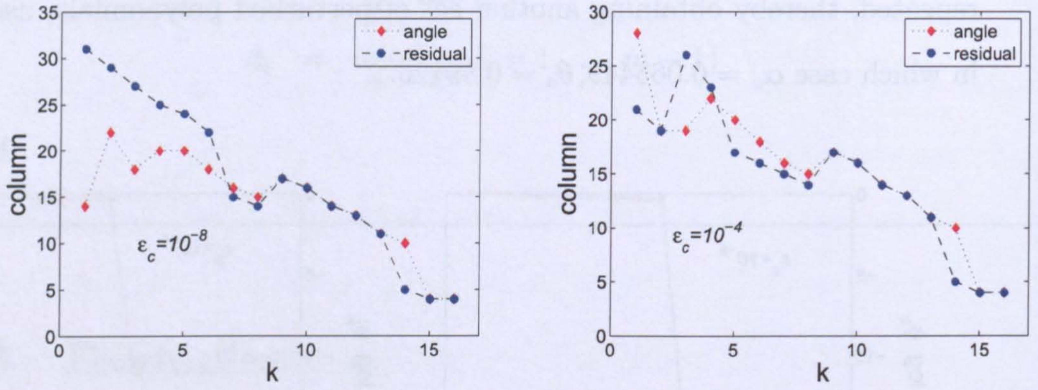


Figure 6.3: The column of $S_k(\tilde{f}, \alpha_o\tilde{g})$ for which the error in (6.2) is a minimum, using Methods 4 and 5, against k, for Example 6.1.

Figure 6.3 shows, for each value of $k = 1, \ldots, 16$, the column of $S_k(\tilde{f}, \alpha_o\tilde{g})$ for which the error in (6.2) is a minimum, using Methods 4 and 5. It is clear that the two methods do not yield the same columns for all values of k, and the greatest differences occur for small and large values of k. Moreover, it is seen that Methods 4 and 5 yield different columns for which this optimal value $k = 7$ is achieved for $\varepsilon_c = 10^{-8}$ and $\varepsilon_c = 10^{-4}$. Also, it is noted that the difference in the optimal columns makes no change on the determination of the degree of an approximate GCD of $f(x)$ and $g(x)$ by Methods 4 and 5. □

Example 6.2. Consider the exact polynomials $\hat{f}(x)$ and $\hat{g}(x)$, whose roots and multiplicities are specified in Table 6.2. It is seen that $m = 16, n = 27$ and the degree of their GCD is $\hat{d} = 6$.

Root of $\hat{f}(x)$	Multiplicity
-7.4420e-005	2
-9.9656e-005	5
-6.3668e-005	5
-4.5823e-005	4

Root of $\hat{g}(x)$	Multiplicity
-7.4420e-005	1
-9.9656e-005	1
-6.3668e-005	4
-7.4936e-001	5
3.0465e-001	5
-4.5435e-001	5
9.1342e-001	2
-1.6942e-001	4

Table 6.2: The roots and multiplicities of $\hat{f}(x)$ and $\hat{g}(x)$ for Example 6.2.

Uniformly distributed random noise was added to each polynomial, such that the componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^4$. The noisy polynomials were then preprocessed by the operations described in Section 4.4, thereby yielding the Sylvester matrix $S(\tilde{f}, \alpha_o \tilde{g})$, where $\alpha_o = 1.8931e + 012$ and $\theta_o = 9.5861e - 003$.

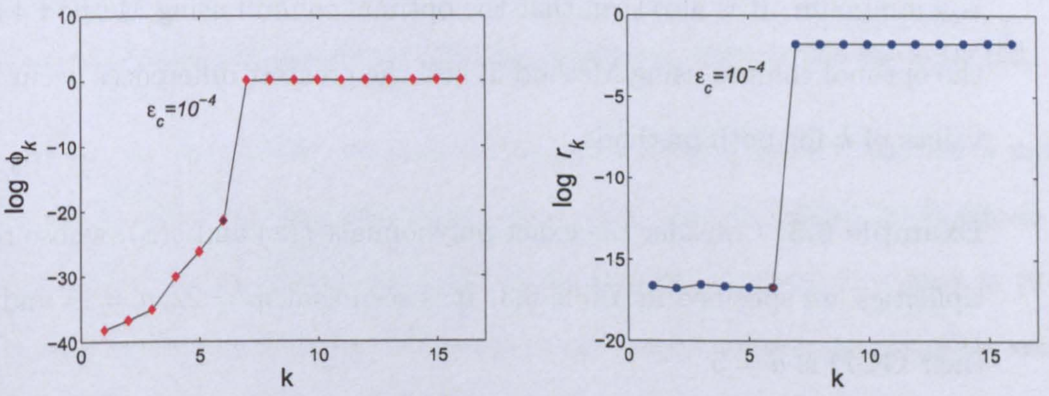


Figure 6.4: The variation of $\log \phi_k$ and $\log r_k$ with k, for Example 6.2.

Figure 6.4 shows the variation of $\log \phi_k$ and $\log r_k$ with k using Methods 4 and 5, respectively. It is seen that the maximum gradient in each graph occurs when $k = 6$, and thus $d_\phi = d_r = \hat{d} = 6$. Also, the degree of an approximate GCD is clearly defined

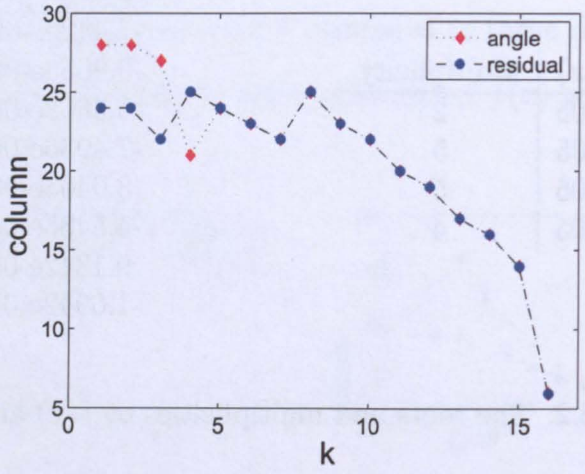


Figure 6.5: The column of $S_k(\tilde{f}, \alpha_o \tilde{g})$ for which the error in (6.2) is a minimum, using Methods 4 and 5, against k with $\varepsilon_c = 10^{-4}$, for Example 6.2.

for both methods, even though $\varepsilon_c^{-1} = 10^4$ is relatively small. Figure 6.5 shows, for Methods 4 and 5, the column of $S_k(\tilde{f}, \alpha_o \tilde{g}), k = 1, \ldots, 16$, for which the error in (6.2) is a minimum. It is also seen that the optimal column using Method 4 is the same as the optimal column using Method 5, and the greatest differences occur only for small values of k for both methods. □

Example 6.3. Consider the exact polynomials $\hat{f}(x)$ and $\hat{g}(x)$, whose roots and multiplicities are specified in Table 6.3. It is seen that $m = 22, n = 14$ and the degree of their GCD is $\hat{d} = 5$.

Uniformly distributed random noise was added to each polynomial, such that the componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^4$. The noisy polynomials were then normalised by the geometric means of their coefficients initially, after which they were preprocessed, thereby yielding the Sylvester matrix $S(\tilde{f}, \alpha_o \tilde{g})$, where $\alpha_o = 4.6574e-009$ and $\theta_o = 1.6913e-003$.

Root of $\hat{f}(x)$	Multiplicity
7.9617e-005	4
-8.8440e-005	5
-8.0504e-005	5
-2.0403e-005	4
7.8861e-005	4

Root of $\hat{g}(x)$	Multiplicity
7.9617e-005	3
-8.8440e-005	1
-8.0504e-005	1
-4.4249e+000	5
-6.1417e+000	4

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

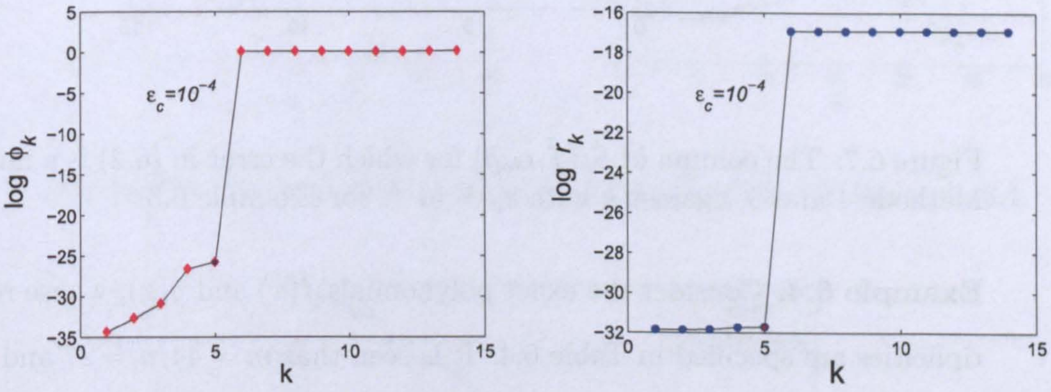


Figure 6.6: The variation of $\log \phi_k$ and $\log r_k$ with k, for Example 6.3.

Figure 6.6 shows the variation of $\log \phi_k$ and $\log r_k$ with k, and it is seen that $d_\phi = d_r = \hat{d} = 5$, such that these values are clearly defined by Methods 4 and 5. Figure 6.7 shows that the column of $S_k(\tilde{f}, \alpha_o \tilde{g})$ for which the error in (6.2) is a minimum is similar to Figure 6.3 because the largest differences occur for small and large values of k for Methods 4 and 5. Also they yield identical results for other values of k, including $k = 5$. $\square$

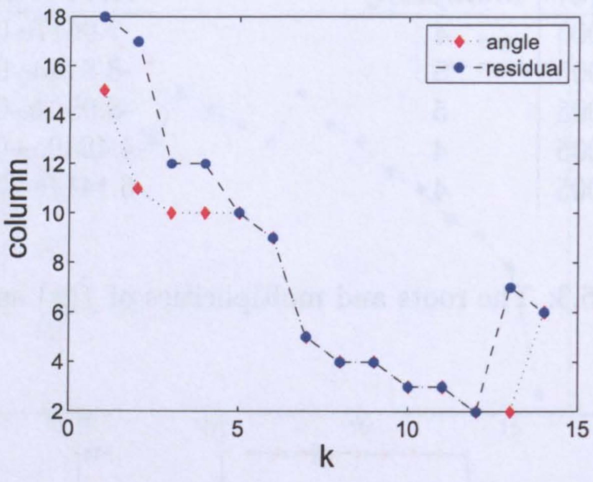


Figure 6.7: The column of $S_k(\tilde{f}, \alpha_o\tilde{g})$ for which the error in (6.2) is a minimum, using Methods 4 and 5, against k with $\varepsilon_c = 10^{-4}$, for Example 6.3.

Example 6.4. Consider the exact polynomials $\hat{f}(x)$ and $\hat{g}(x)$, whose roots and multiplicities are specified in Table 6.4. It is seen that $m = 44, n = 27$ and the degree of their GCD is $\hat{d} = 24$.

Root of $\hat{f}(x)$	Multiplicity
-9.6104e+000	11
-7.2187e-001	9
9.1180e+000	7
1.4302e+000	11
-8.4822e+000	2
-2.7506e+000	4

Root of $\hat{g}(x)$	Multiplicity
-9.6104e+000	9
-7.2187e-001	8
9.1180e+000	10

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

Noise was added in the componentwise sense to each polynomial, such that the componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^8$. The noisy polynomials were then normalised by the geometric means of their coefficients initially, after which they were preprocessed, thereby yielding the Sylvester matrix $S(\tilde{f}, \alpha_o \tilde{g})$, where $\alpha_o = 2.2631e+004$ and $\theta_o = 3.0935$.

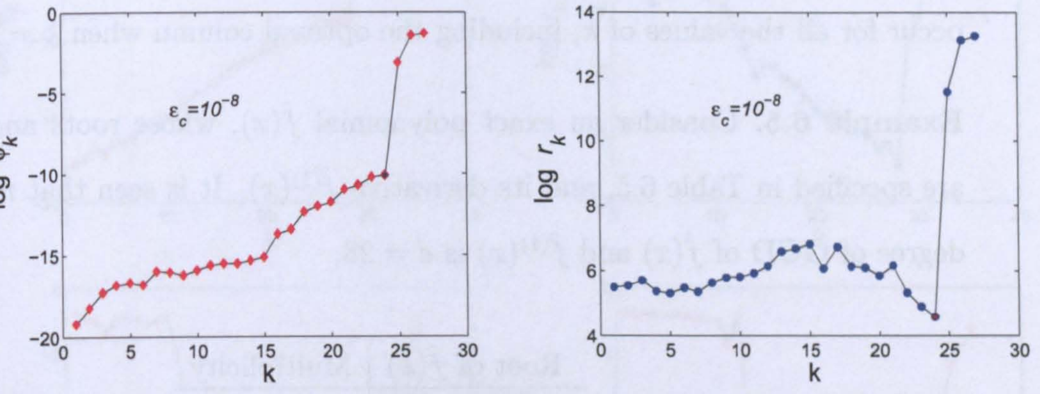


Figure 6.8: The variation of $\log \phi_k$ and $\log r_k$ with k, for Example 6.4.

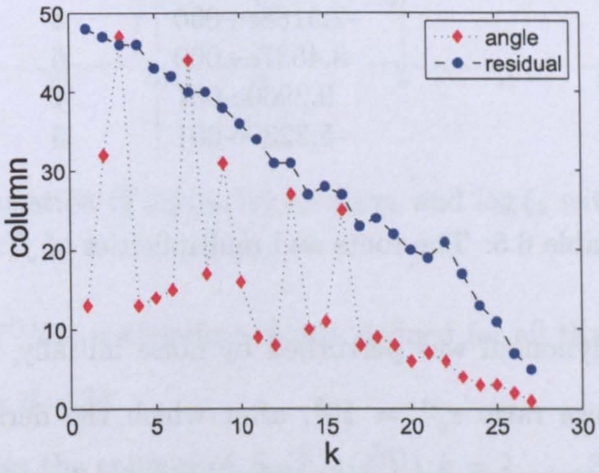


Figure 6.9: The column of $S_k(\tilde{f}, \alpha_o \tilde{g})$ for which the error in (6.2) is a minimum, using Methods 4 and 5, against k with $\varepsilon_c = 10^{-8}$, for Example 6.4.

Figure 6.8 shows the variation of $\log \phi_k$ and $\log r_k$ with $k = 1, \ldots, 27$, using Methods 4 and 5, and Figure 6.9 shows the column of $S_k(\tilde{f}, \alpha_o \tilde{g})$ for which the error in (6.2) is a minimum. It is clearly seen from Figure 6.8 that the maximum gradient in each graph occurs when $k = 24$, which means the degree of an approximate GCD is clearly defined for both methods, such that $d_\phi = d_r = \hat{d} = 24$. Furthermore, Figure 6.9 is different from Figures 6.3, 6.5 and 6.7, because the differences in the results occur for all the values of k, including the optimal column when $k = 24$. $\square$

Example 6.5. Consider an exact polynomial $\hat{f}(x)$, whose roots and multiplicities are specified in Table 6.5, and its derivative $\hat{f}^{(1)}(x)$. It is seen that $m = 36$ and the degree of GCD of $\hat{f}(x)$ and $\hat{f}^{(1)}(x)$ is $\hat{d} = 28$.

Root of $\hat{f}(x)$	Multiplicity
-1.3708e+000	1
-3.2431e+000	2
4.4145e+000	3
-9.7269e+000	4
-2.5188e+000	5
8.4537e+000	6
9.2960e-001	7
-5.2230e-001	8

Table 6.5: The roots and multiplicities of $\hat{f}(x)$ for Example 6.5.

This polynomial was perturbed by noise initially, such that the componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^8$, after which the derivative $f^{(1)}(x)$ was calculated from the noisy polynomial $f(x)$. They were then preprocessed, thereby yielding the Sylvester matrix $S(\tilde{f}, \alpha_o \tilde{f}^{(1)})$, where $\alpha_o = 1.3964$ and $\theta_o = 2.1731$.

Figure 6.10 shows the variation of $\log \phi_k, \log r_k, \log \eta_k$ and $\log \xi_k$, which are calculated from Algorithm 6.2, with k. It is seen that the maximum changes in $\log \phi_k$ and $\log r_k$ occur when $k = 28$, and the global minima in $\log \eta_k$ and $\log \xi_k$ are also achieved when $k = 28$, which is correct because $\hat{d} = 28$. The degree of an approximate

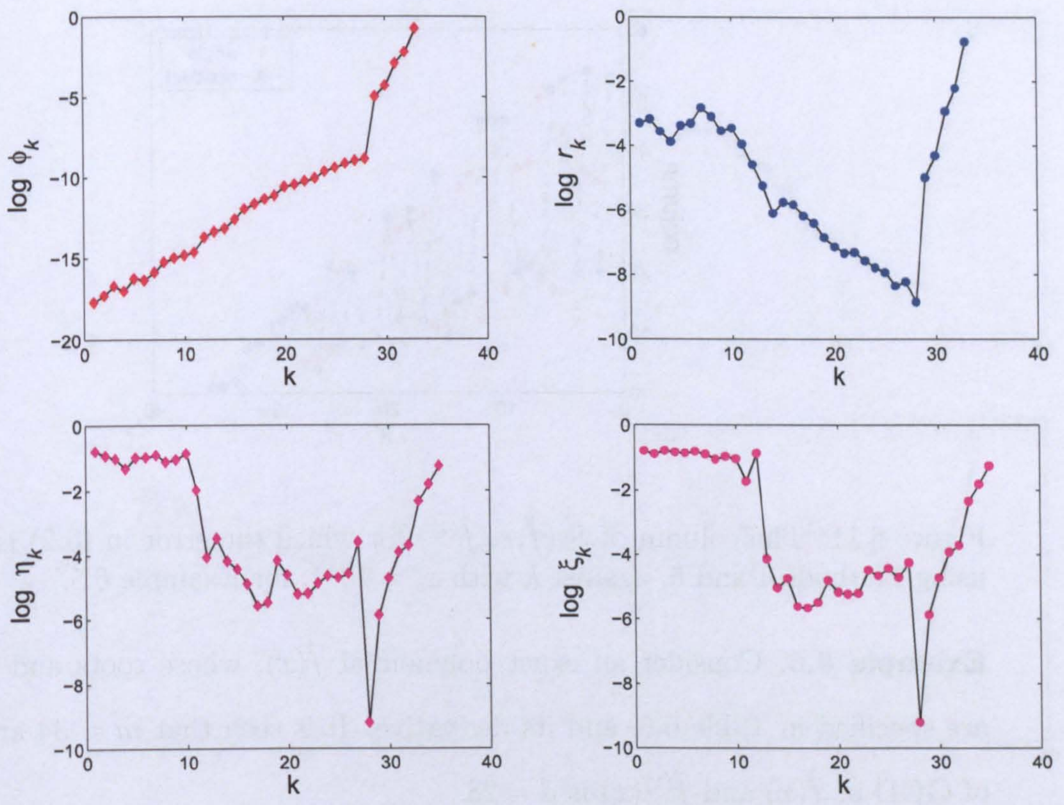


Figure 6.10: The variation of $\log \phi_k, \log r_k, \log \eta_k$ and $\log \xi_k$ with k and $\varepsilon_c = 10^{-8}$ for Example 6.5.

GCD of $f(x)$ and $f^{(1)}(x)$ is therefore clearly defined for all three methods, such that $d_\phi = d_r = d_\eta = d_\xi = \hat{d} = 28$.

Figure 6.11 shows the column of $S_k(\tilde{f}, \alpha_o \tilde{f}^{(1)}), k = 1, \ldots, 35$, for which the error in (6.2) is a minimum, using Methods 4 and 5. It is noted that the two methods do not yield the same column for all values of k, and the greatest differences occur when $k < 28$. Because Method 6 uses these different columns that are moved to the right hand side of (6.2), there are slight differences between $\log \eta_k$ and $\log \xi_k$ especially for small values of k, which can be seen from the lower graphs of Figure 6.10 □

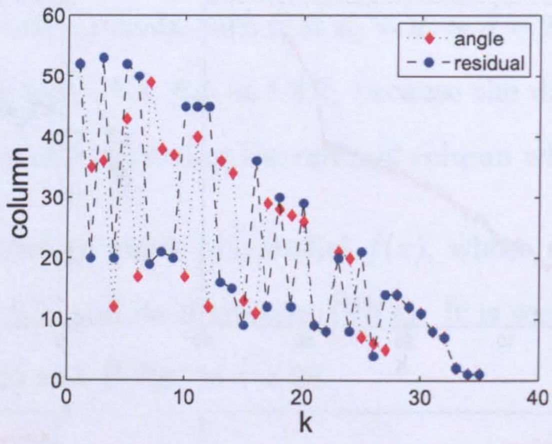


Figure 6.11: The column of $S_k(\tilde{f}, \alpha_o \tilde{f}^{(1)})$ for which the error in (6.2) is a minimum, using Methods 4 and 5, against k with $\varepsilon_c = 10^{-8}$, for Example 6.5.

Example 6.6. Consider an exact polynomial $\hat{f}(x)$, whose roots and multiplicities are specified in Table 6.6, and its derivative. It is seen that $m = 34$ and the degree of GCD of $\hat{f}(x)$ and $\hat{f}^{(1)}(x)$ is $\hat{d} = 28$.

Root of $\hat{f}(x)$	Multiplicity
-3.4624e+000	2
2.6891e+000	2
8.4689e+000	2
-2.5214e+000	8
-1.6262e+000	9
6.1616e+000	11

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

This polynomial was perturbed by noisy initially, such that the componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^8$, after which the derivative $f^{(1)}(x)$ was calculated from the noisy polynomial $f(x)$. They were then preprocessed, thereby yielding the Sylvester matrix $S(\tilde{f}, \alpha_o \tilde{f}^{(1)})$, where $\alpha_o = 1.6483$ and $\theta_o = 3.2921$.

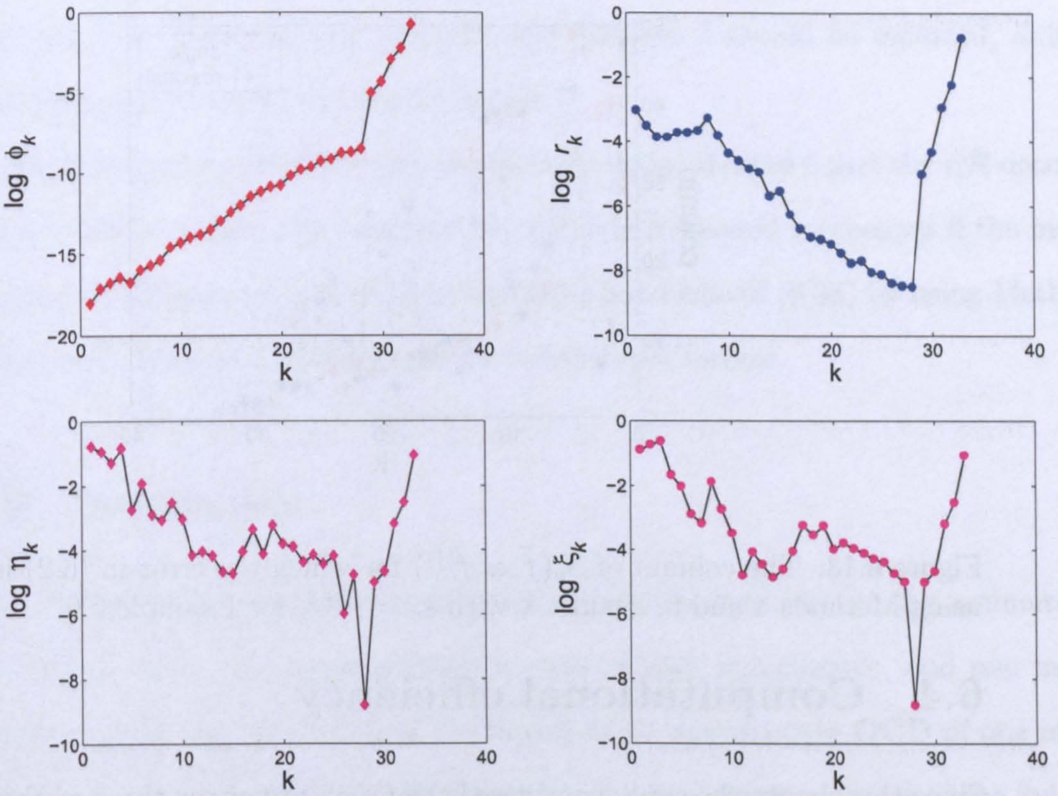


Figure 6.12: The variation of $\log\phi_k, \log r_k, \log\eta_k$ and $\log\xi_k$ with k and $\varepsilon_c = 10^{-8}$ for Example 6.6.

Figure 6.12 shows the variation of $\log\phi_k, \log r_k, \log\eta_k$ and $\log\xi_k$, with k, and it is seen that the four graphs return the same answer because the maximum gradients in $\log\phi_k$ and $\log r_k$ and the global minima in $\log\eta_k$ and $\log\xi_k$ occur when $k = 28$. Also, the degree of an approximate GCD of $f(x)$ and $f^{(1)}(x)$ is clearly defined, and thus $d_\phi = d_r = d_\eta = d_\xi = \hat{d} = 28$. Figure 6.13, which shows the column of $S_k(\tilde{f}, \alpha_o\tilde{f}^{(1)})$, for which the error in (6.2) is a minimum, is similar to Figure 6.11 because the largest differences occur for most values of k except the large values of k and the optimal columns when $k = \hat{d}$, using Methods 4 and 5, are the same. □

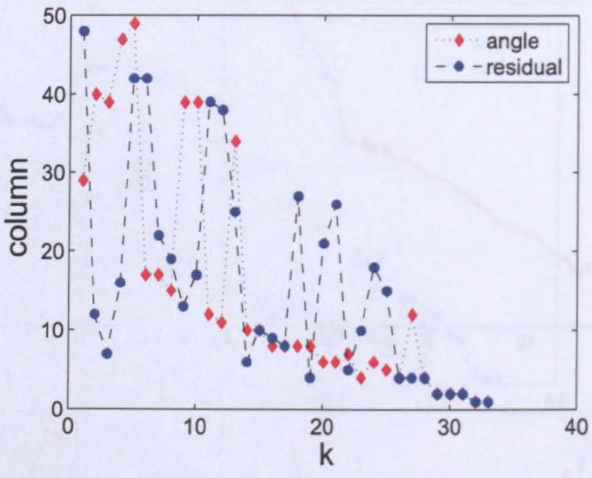


Figure 6.13: The column of $S_k(\tilde{f}, \alpha_o \tilde{f}^{(1)})$ for which the error in (6.2) is a minimum, using Methods 4 and 5, against k with $\varepsilon_c = 10^{-8}$, for Example 6.6.

6.4 Computational efficiency

Since this chapter has only considered the feasibility of the three methods, algorithm issues associated with the solution of (6.2) have not been addressed, however, it is possible to consider the issues of computational efficiency as the future work.

The major cost of the designed root solver is the estimation of the degree of the approximate GCD because these methods require that (6.2) be solved repeatedly, where the coefficient matrices of successive equations differ by one column only. Hence the algorithms must be implemented efficiently.

Method 5 can therefore be done by computing the QR decomposition of $S(f, g)$ once, and then using update procedures to calculate the QR decomposition of $S_k(f, g)$, $k = 2, \ldots, \min(m, n)$, such as (6.2) solved by the function `qrdelete` in MATLAB [1]. The computation in Method 4 would be more expensive because the SVD does not

[1]This function deletes a column or row from the QR factorization.

have an update. It may therefore be desirable to use the QR decomposition, rather than the SVD, meaning that perhaps only Method 5 should be included, although both Methods 4 and 5 yield good answers.

Since major improvements are obtained by using Method 5 and the QR decomposition to allow update, the computation in Method 6 would be cheaper if the method chooses the column of $S_k(f, g)$ to be the right hand side of (6.38) by using Method 5, rather than Method 4. This should be investigated further.

6.5 Summary

This chapter has presented two methods (Methods 4 and 5) for the estimation of the degree of an approximate GCD of two inexact polynomials, and one method (Method 6) for the estimation of the degree of an approximate GCD of one inexact polynomial and its derivative. All methods use subresultant matrices of the Sylvester resultant matrix, but Methods 4 and 5 differ in the criterion used to define the error in an approximate linear algebraic equation of (6.2), and Method 6 uses the constraint (6.35) between $f(x)$ and $f^{(1)}(x)$ and requires the optimal columns calculated from Methods 4 or 5.

Six examples were presented and it was shown that these methods yield good results for both situations. The examples suggest that they return the same degree $d := d_\phi = d_r$ of an approximate GCD of $f(x)$ and $g(x)$, or $d := d_\phi = d_r = d_\eta = d_\xi$ of an approximate GCD of $f(x)$ and $f^{(1)}(x)$, even though the column of the subresultant matrix associated with d may differ between Methods 4 and 5.

Chapter 7

The coefficients of an approximate GCD

The designed polynomial root solver presented in Chapter 3 has involved a sequence of the approximate GCD computations, where an approximate GCD obtained from the ith iteration is used for the $i+1$th iteration, and thus a very important part of this root solver is the determination of an approximate GCD of two inexact polynomials. In particular, the degree of an approximate GCD should be determined initially, after which the coefficients of an approximate GCD are calculated. The calculation of the degree of an approximate GCD has been covered in Chapters 5 and 6, which is a non-trivial computation because it reduces to the estimation of the rank loss of a noisy resultant matrix. The calculation of the coefficients of an approximate GCD is described in this chapter.

The use of approximate polynomial factorisation is considered for the calculation of the coefficients of an approximate GCD. Suppose that the degree of an approximate GCD $d(x)$ of two polynomials $f(x)$ and $g(x)$ is known, and thus there exist quotient polynomials $u(x)$ and $v(x)$, such that

$$f(x) \approx d(x)u(x) \qquad \text{and} \qquad g(x) \approx d(x)v(x),$$

and it can be written in matrix form,

$$\begin{bmatrix} U \\ V \end{bmatrix} \mathbf{d} \approx \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix}, \tag{7.1}$$

where U, V are the Toeplitz matrices, and **d**, **f**, **g** are the coefficients vectors of $d(x)$, $f(x), g(x)$ respectively. The coefficients of $d(x)$ are therefore calculated from the approximation (7.1), which must be perturbed in order to induce an exact solution. In this case, structured perturbations are required such that the structure of the coefficient matrix in (7.1) is preserved in order to guarantee that the matrix-vector product represents a polynomial multiplication. This kind of perturbation is calculated by the method of structured nonlinear total least norm (SNTLN) [57], which is an extension of the method of structure total least norm (STLN) [56].

In [57] Rosen et al. remark that:

> "STLN is a problem formulation for obtaining an approximate solution to the overdetermined linear system $Ax \approx b$ preserving the given affine structure in A or $[A|b]$, where error can occur in both the vector b and the matrix A. The approximate solution can be obtained to minimize the error in the L_p norm, where $p = 1, 2$, or ∞. In the extension of STLN to nonlinear problems, the elements of A may be differentiable nonlinear functions of a parameter vector, whose value needs to be approximated. We call this extension structured nonlinear total least norm (SNTLN). "

The method of SNTLN, which yields a non-linear equation that is solved by the Newton-Raphson method, is therefore used to obtain the approximate solution. The computational results demonstrate that the method of SNTLN recovers good approximations to the values of the coefficients of an approximate GCD of two polynomials, in the presence of noise in the data.

7.1 The method of SNTLN

Since the calculation of the degree of an approximate GCD of two polynomials $f(x)$ and $g(x)$ has been considered in Chapters 5 and 6, this section assumes that the degree d of an approximate GCD is known, and it describes the method of SNTLN for the calculation of its coefficients. For simplicity, this section only considers the calculation of the coefficients of an approximate GCD of $\tilde{f}(y)$ and $\tilde{g}(y)$ that are preprocessed from $f(x)$ and $g(x)$, respectively, by the operations described in Section 4.4.

It is recalled that the given inexact polynomials are defined in (4.24), which are repeated here for convenience,

$$\tilde{f}(y) = \sum_{i=0}^{m} \tilde{a}_i y^{m-i} \qquad \text{and} \qquad \alpha_o \tilde{g}(y) = \alpha_o \sum_{i=0}^{n} \tilde{b}_i y^{n-i}, \tag{7.2}$$

whose coefficients are

$$\tilde{a}_i = \bar{a}_i \theta_o^{m-i} \qquad \text{and} \qquad \tilde{b}_i = \bar{b}_i \theta_o^{n-i}, \tag{7.3}$$

where $\bar{a}_i$ and $\bar{b}_i$ are defined in (4.17) and (4.18), and α_o, θ_o are solution of the minimisation problem (4.22).

It is assumed that $\tilde{f}(y)$ and $\tilde{g}(y)$ are inexact, and thus an approximate GCD $\tilde{c}_d(y)$ of $\tilde{f}(y)$ and $\tilde{g}(y)$, of degree d, satisfies

$$\tilde{f}(y) \approx \tilde{c}_d(y)\tilde{u}_d(y) \qquad \text{and} \qquad \alpha_o \tilde{g}(y) \approx \tilde{c}_d(y)\tilde{v}_d(y), \tag{7.4}$$

where

$$\begin{aligned}\tilde{u}_d(y) &= \sum_{i=0}^{m-d} \tilde{u}_{d,i} y^{m-d-i}, \quad \tilde{u}_{d,i} = u_{d,i}\theta_{\text{o}}^{m-d-i},\\ \tilde{v}_d(y) &= \sum_{i=0}^{n-d} \tilde{v}_{d,i} y^{n-d-i}, \quad \tilde{v}_{d,i} = v_{d,i}\theta_{\text{o}}^{n-d-i},\\ \tilde{c}_d(y) &= \sum_{i=0}^{d} \tilde{c}_{d,i} y^{d-i}, \quad \tilde{c}_{d,i} = c_{d,i}\theta_{\text{o}}^{d-i},\end{aligned} \tag{7.5}$$

are the transformed polynomials from $u_d(x), v_d(x)$ and $c_d(x)$ that are given by

$$u_d(x) = \sum_{i=0}^{m-d} u_{d,i}x^{m-d-i}, \quad v_d(x) = \sum_{i=0}^{n-d} v_{d,i}x^{n-d-i}, \quad c_d(x) = \sum_{i=0}^{d} c_{d,i}x^{d-i},$$

respectively, using the substitution $x = \theta_{\text{o}}y$. Equation (7.4) is therefore required for the use of the method of SNTLN to compute the coefficients of an approximate GCD.

It is stated that the method of SNTLN requires initial estimates of the quotient polynomials $\tilde{u}_d(y)$ and $\tilde{v}_d(y)$, and the calculation of these estimates is similar to that for $\tilde{f}(y)$ and $\tilde{f}^{(1)}(y)$, which is described in Section 6.2.

The subresultant matrix $S_d(\tilde{f}, \alpha_{\text{o}}\tilde{g})$ is formed, where the coefficients of $\tilde{f}(y)$ and $\tilde{g}(y)$ are defined in (7.2). Likewise, as shown in Section 6.2, assume that h_{d,j^*}, the j^*th column of $S_d(\tilde{f}, \alpha_{\text{o}}\tilde{g})$, is the optimal column that is removed from the matrix, and $H_{d,j^*} \in \mathbb{R}^{(m+n-d+1)\times(m+n-2d+1)}$ is the matrix from the other columns of $S_d(\tilde{f}, \alpha_{\text{o}}\tilde{g})$ defined as

$$H_{d,j^*} = \begin{bmatrix} s_{d,1} & \cdots & s_{d,j^*-1} & s_{d,j^*+1} & \cdots & s_{d,m+n-2d+2} \end{bmatrix},$$

where $s_{d,i} \in \mathbb{R}^{m+n-2d+2}$ is the ith column of $S_d(\tilde{f}, \alpha_{\text{o}}\tilde{g})$. The removal of the j^*th column of $S_d(\tilde{f}, \alpha_{\text{o}}\tilde{g})$ to the right hand side therefore yields the equation

$$H_{d,j^*}x \approx h_{d,j^*}, \tag{7.6}$$

where

$$x = \begin{bmatrix} x_1 & \cdots & x_{j^*-1} & x_{j^*+1} & \cdots & x_{m+n-2d+2} \end{bmatrix}^T \in \mathbb{R}^{m+n-2d+1},$$

and

$$\begin{bmatrix} \tilde{\mathbf{v}}_d(\theta_o) \\ -\tilde{\mathbf{u}}_d(\theta_o) \end{bmatrix} = \begin{bmatrix} \tilde{v}_{d,0} \\ \vdots \\ \tilde{v}_{d,n-d} \\ -\tilde{u}_{d,0} \\ \vdots \\ -\tilde{u}_{d,m-d} \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_{j^*-1} \\ -1 \\ x_{j^*+1} \\ \vdots \\ x_{m+n-2d+2} \end{bmatrix} \in \mathbb{R}^{m+n-2d+2}. \tag{7.7}$$

Hence, the vectors of coefficients $\tilde{\mathbf{u}}_d(\theta_o)$ and $\tilde{\mathbf{v}}_d(\theta_o)$ of $\tilde{u}_d(y)$ and $\tilde{v}_d(y)$, are computed from (7.7) with the index j^* for which the minima occurs in (7.6), respectively. Moreover, the coefficients of $u_d(x)$ and $v_d(x)$ can be obtained from (7.5) and (7.7)

$$u_{d,i} = \frac{\tilde{u}_{d,i}}{\theta_o^{m-d-i}} \quad \text{and} \quad v_{d,i} = \frac{\tilde{v}_{d,i}}{\theta_o^{n-d-i}}. \tag{7.8}$$

The method of SNTLN requires that $\tilde{f}(y)$ and $\tilde{g}(y)$, and the initial estimates of the quotient polynomials, be rewritten. Specifically, it follows from (7.2) and (7.3) that

$$\tilde{f}(y) = \sum_{i=0}^{m} (\bar{a}_i \theta_o^{m-i}) y^{m-i} \quad \text{and} \quad \tilde{g}(y) = \sum_{i=0}^{n} (\bar{b}_i \theta_o^{n-i}) y^{n-i}, \tag{7.9}$$

and thus it is expected to include θ as a parameter to be optimized by SNTLN. The substitution of $\theta = \theta_o$ into (7.9) therefore yields

$$\tilde{f}(y,\theta) = \sum_{i=0}^{m} (\bar{a}_i \theta^{m-i}) y^{m-i} \quad \text{and} \quad \tilde{g}(y,\theta) = \sum_{i=0}^{n} (\bar{b}_i \theta^{n-i}) y^{n-i}, \tag{7.10}$$

where the arbitrary value θ will be refined by the method of SNTLN, using θ_o as the initial estimate. Similarly, the initial estimates of the quotient polynomials are rewritten as

$$\tilde{u}_d(y,\theta) = \sum_{i=0}^{m-d}(u_{d,i}\theta^{m-d-i})y^{m-d-i} \quad \text{and} \quad \tilde{v}_d(y,\theta) = \sum_{i=0}^{n-d}(v_{d,i}\theta^{n-d-i})y^{n-d-i},$$

where $u_{d,i}$ and $v_{d,i}$ are computed from (7.8), and y is the independent variable in these polynomials.

The approximate decompositions (7.4) are therefore replaced by

$$\tilde{f}(y,\theta) \approx \tilde{c}_d(y,\theta)\tilde{u}_d(y,\theta) \quad \text{and} \quad \alpha_o\tilde{g}(y,\theta) \approx \tilde{c}_d(y,\theta)\tilde{v}_d(y,\theta), \tag{7.11}$$

where

$$\tilde{c}_d(y,\theta) = \sum_{i=0}^{d}(c_{d,i}\theta^{d-i})y^{d-i},$$

and thus (7.11) can be written in matrix form,

$$\begin{bmatrix} U_d(u_d,\theta) \\ V_d(v_d,\theta) \end{bmatrix} \tilde{\mathbf{c}}_d(\theta) \approx \begin{bmatrix} \tilde{\mathbf{f}}(\theta) \\ \alpha_o\tilde{\mathbf{g}}(\theta) \end{bmatrix}, \tag{7.12}$$

where

$$\begin{aligned}
u_d &= \begin{bmatrix} u_{d,0} & u_{d,1} & \cdots & u_{d,m-d-1} & u_{d,m-d} \end{bmatrix}^T \in \mathbb{R}^{m-d+1}, \\
v_d &= \begin{bmatrix} v_{d,0} & v_{d,1} & \cdots & v_{d,n-d-1} & v_{d,n-d} \end{bmatrix}^T \in \mathbb{R}^{n-d+1}, \\
\tilde{\mathbf{f}}(\theta) &= \begin{bmatrix} \bar{a}_0\theta^m & \bar{a}_1\theta^{m-1} & \cdots & \bar{a}_{m-1}\theta & \bar{a}_m \end{bmatrix}^T \in \mathbb{R}^{m+1}, \\
\tilde{\mathbf{g}}(\theta) &= \begin{bmatrix} \bar{b}_0\theta^n & \bar{b}_1\theta^{n-1}, & \cdots & \bar{b}_{n-1}\theta & \bar{b}_n \end{bmatrix}^T \in \mathbb{R}^{n+1}, \\
\tilde{\mathbf{c}}_d(\theta) &= \begin{bmatrix} c_{d,0}\theta^d & c_{d,1}\theta^{d-1} & \cdots & c_{d,k-1}\theta & c_{d,k} \end{bmatrix}^T \in \mathbb{R}^{d+1}.
\end{aligned} \tag{7.13}$$

The coefficient matrix in (7.12) is of order $(m+n+2)\times(d+1)$, where $U_d(u_d,\theta)$ and

$V_d(v_d,\theta)$ are given by, respectively,

$$U_d(u_d,\theta) = \begin{bmatrix} u_{d,0}\theta^{m-d} & & \\ u_{d,1}\theta^{m-d-1} & \ddots & \\ \vdots & \ddots & u_{d,0}\theta^{m-d} \\ u_{d,m-d-1}\theta & \ddots & u_{d,1}\theta^{m-d-1} \\ u_{d,m-d} & \ddots & \vdots \\ & \ddots & u_{d,m-d-1}\theta \\ & & u_{d,m-d} \end{bmatrix} \in \mathbb{R}^{(m+1)\times(d+1)},$$

$$V_d(v_d,\theta) = \begin{bmatrix} v_{d,0}\theta^{n-d} & & \\ v_{d,1}\theta^{n-d-1} & \ddots & \\ \vdots & \ddots & v_{d,0}\theta^{n-d} \\ v_{d,n-d-1}\theta & \ddots & v_{d,1}\theta^{n-d-1} \\ v_{d,n-d} & \ddots & \vdots \\ & \ddots & v_{d,n-d-1}\theta \\ & & v_{d,n-d} \end{bmatrix} \in \mathbb{R}^{(n+1)\times(d+1)}.$$

The approximation equation (7.12) is not satisfied exactly because the polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$ are inexact. It is therefore necessary to add a structured matrix to the coefficient matrix on the left hand side, and a vector to the right hand side, of this approximate equation, which is therefore replaced by

$$\begin{bmatrix} U_d(u_d,\theta) + P_d(z_d,\theta) \\ V_d(v_d,\theta) + Q_d(z_d,\theta) \end{bmatrix} \tilde{\mathbf{c}}_d(\theta) = \begin{bmatrix} \tilde{\mathbf{f}}(\theta) + \mathbf{p}_d(s_d,\theta) \\ (\alpha_o + \beta_o)\tilde{\mathbf{g}}(\theta) + \mathbf{q}_d(t_d,\theta) \end{bmatrix}, \qquad (7.14)$$

where

$$P_d(z_d,\theta) = \begin{bmatrix} z_{d,0}\theta^{m-d} & & \\ z_{d,1}\theta^{m-d-1} & \ddots & \\ \vdots & \ddots & z_{d,0}\theta^{m-d} \\ z_{d,m-d-1}\theta & \ddots & z_{d,1}\theta^{m-d-1} \\ z_{d,m-d} & \ddots & \vdots \\ & \ddots & z_{d,m-d-1}\theta \\ & & z_{d,m-d} \end{bmatrix} \in \mathbb{R}^{(m+1)\times(d+1)},$$

$$Q_d(z_d,\theta) = \begin{bmatrix} z_{d,m-d+1}\theta^{n-d} & & \\ z_{d,m-d+2}\theta^{n-d-1} & \ddots & \\ \vdots & \ddots & z_{d,m-d+1}\theta^{n-d} \\ z_{d,m+n-2d}\theta & \ddots & z_{d,m-d+2}\theta^{n-d-1} \\ z_{d,m+n-2d+1} & \ddots & \vdots \\ & \ddots & z_{d,m+n-2d}\theta \\ & & z_{d,m+n-2d+1} \end{bmatrix} \in \mathbb{R}^{(n+1)\times(d+1)},$$

are the matrices that contain the perturbations $z_{d,i}$,

$$z_d = \begin{bmatrix} z_{d,0} & \cdots & z_{d,m-d} & z_{d,m-d+1} & \cdots & z_{d,m+n-2d+1} \end{bmatrix}^T \in \mathbb{R}^{m+n-2d+2},$$

which is the vector of structured perturbations that are added to the coefficients $u_{d,i}$ and $v_{d,i}$, the vectors $\mathbf{p}_d = \mathbf{p}_d(s_d,\theta)$ and $\mathbf{q}_d = \mathbf{q}_d(t_d,\theta)$,

$$\begin{aligned} \mathbf{p}_d &= \begin{bmatrix} s_{d,0}\theta^m & s_{d,1}\theta^{m-1} & \cdots & s_{d,m-1}\theta & s_{d,m} \end{bmatrix}^T \in \mathbb{R}^{m+1} \\ \mathbf{q}_d &= \begin{bmatrix} t_{d,0}\theta^n & t_{d,1}\theta^{n-1} & \cdots & t_{d,n-1}\theta & t_{d,n} \end{bmatrix}^T \in \mathbb{R}^{n+1}. \end{aligned}$$

are the vectors of coefficients that are added to the coefficients of $\tilde{f}(y,\theta)$ and $\tilde{g}(y,\theta)$

with

$$\begin{aligned} s_d &= \begin{bmatrix} s_{d,0} & s_{d,1} & \cdots & s_{d,m-1} & s_{d,m} \end{bmatrix}^T \in \mathbb{R}^{m+1}, \\ t_d &= \begin{bmatrix} t_{d,0} & t_{d,1} & \cdots & t_{d,n-1} & t_{d,n} \end{bmatrix}^T \in \mathbb{R}^{n+1}, \end{aligned}$$

and β_o is added to α_o.

Eq.(7.14) is a non-linear equation for the vectors z_d, s_d, t_d and $\tilde{\mathbf{c}}_d(\theta)$, and the scalars β_o and θ. This non-linear equation is solved by the Newton-Raphson method.

The residual that is associated with an approximate solution of (7.14) is

$$\begin{aligned} r(z_d, s_d, t_d, \tilde{\mathbf{c}}_d, \beta_o, \theta) &= \begin{bmatrix} \tilde{\mathbf{f}}(\theta) + \mathbf{p}_d(s_d, \theta) \\ (\alpha_o + \beta_o)(\tilde{\mathbf{g}}(\theta) + \mathbf{q}_d(t_d, \theta)) \end{bmatrix} \\ &\quad - \begin{bmatrix} U_d(u_d, \theta) + P_d(z_d, \theta) \\ V_d(v_d, \theta) + Q_d(z_d, \theta) \end{bmatrix} \tilde{\mathbf{c}}_d(\theta), \qquad (7.15) \end{aligned}$$

and thus

$$r(z_d + \delta z_d, s_d + \delta s_d, t_d + \delta t_d, \tilde{\mathbf{c}}_d + \delta\tilde{\mathbf{c}}_d, \beta_o + \delta\beta_o, \theta + \delta\theta),$$

is equal to

$$\begin{bmatrix} \tilde{\mathbf{f}}(\theta + \delta\theta) + \mathbf{p}_d(s_d + \delta s_d, \theta + \delta\theta) \\ (\alpha_o + \beta_o + \delta\beta_o)(\tilde{\mathbf{g}}(\theta + \delta\theta) + \mathbf{q}_d(t_d + \delta t_d, \theta + \delta\theta)) \end{bmatrix} \qquad (7.16)$$

$$- \begin{bmatrix} U_d(u_d, \theta + \delta\theta) + P_d(z_d + \delta z_d, \theta + \delta\theta) \\ V_d(v_d, \theta + \delta\theta) + Q_d(z_d + \delta z_d, \theta + \delta\theta) \end{bmatrix} \tilde{\mathbf{c}}_d(\theta + \delta\theta). \qquad (7.17)$$

The Newton-Raphson method requires that the lowest order term of the Taylor expansion of this expression be considered, and it is simplest if the terms in (7.16) and (7.17) are considered separately.

Since

$$\tilde{\mathbf{f}}(\theta+\delta\theta)+\mathbf{p}_d(s_d+\delta s_d,\theta+\delta\theta)\approx\tilde{\mathbf{f}}+\mathbf{p}_d+\frac{\partial\tilde{\mathbf{f}}}{\partial\theta}\delta\theta+\frac{\partial\mathbf{p}_d}{\partial\theta}\delta\theta+\sum_{i=0}^{m}\frac{\partial\mathbf{p}_d}{\partial s_{d,i}}\delta s_{d,i},\qquad(7.18)$$

and

$$\begin{aligned}&(\alpha_o+\beta_o+\delta\beta_o)(\tilde{\mathbf{g}}(\theta+\delta\theta)+\mathbf{q}_d(t_d+\delta t_d,\theta+\delta\theta))\\ \approx\;&(\alpha_o+\beta_o)(\tilde{\mathbf{g}}+\mathbf{q}_d)+(\alpha_o+\beta_o)\left(\frac{\partial\tilde{\mathbf{g}}}{\partial\theta}\delta\theta+\frac{\partial\mathbf{q}_d}{\partial\theta}\delta\theta+\sum_{i=0}^{n}\frac{\partial\mathbf{q}_d}{\partial t_{d,i}}\delta t_{d,i}\right)\\ &+(\tilde{\mathbf{g}}+\mathbf{q}_d)\delta\beta_o,\end{aligned}\qquad(7.19)$$

to first order, where

$$\frac{\partial\tilde{\mathbf{f}}}{\partial\theta}=\begin{bmatrix}m\bar{a}_0\theta^{m-1}\\(m-1)\bar{a}_1\theta^{m-2}\\\vdots\\\bar{a}_{m-1}\\0\end{bmatrix},\qquad\frac{\partial\mathbf{p}_d}{\partial\theta}=\begin{bmatrix}ms_{d,0}\theta^{m-1}\\(m-1)s_{d,1}\theta^{m-2}\\\vdots\\s_{d,m-1}\\0\end{bmatrix},$$

and

$$\frac{\partial\tilde{\mathbf{g}}}{\partial\theta}=\begin{bmatrix}n\bar{b}_0\theta^{n-1}\\(n-1)\bar{b}_1\theta^{n-2}\\\vdots\\\bar{b}_{n-1}\\0\end{bmatrix},\qquad\frac{\partial\mathbf{q}_d}{\partial\theta}=\begin{bmatrix}nt_{d,0}\theta^{n-1}\\(n-1)t_{d,1}\theta^{n-2}\\\vdots\\t_{d,n-1}\\0\end{bmatrix},$$

it follows that (7.18) and (7.19) are the first order approximation of (7.16). It is verified that there exist square diagonal matrices $S=S(\theta)$ and $T=T(\theta)$ such that

$$\mathbf{p}_d=Ss_d\qquad\text{and}\qquad\mathbf{q}_d=Tt_d,$$

where

$$S = S(\theta) = \operatorname{diag}\begin{bmatrix} \theta^m & \theta^{m-1} & \cdots & \theta & 1 \end{bmatrix} \in \mathbb{R}^{(m+1)\times(m+1)}, \tag{7.20}$$

$$T = T(\theta) = \operatorname{diag}\begin{bmatrix} \theta^n & \theta^{n-1} & \cdots & \theta & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}, \tag{7.21}$$

and thus

$$\sum_{i=0}^{m} \frac{\partial \mathbf{p}_d}{\partial s_{d,i}} \delta s_{d,i} = S\delta s_d \qquad \text{and} \qquad \sum_{i=0}^{n} \frac{\partial \mathbf{q}_d}{\partial t_{d,i}} \delta t_{d,i} = T\delta t_d. \tag{7.22}$$

Consider now the first order approximation of (7.17). The coefficient matrix of this term is

$$\begin{aligned} &A_d(u_d, v_d, \theta+\delta\theta) + B_d(z_d + \delta z_d, \theta+\delta\theta) \\ &= \begin{bmatrix} U_d(u_d, \theta+\delta\theta) + P_d(z_d+\delta z_d, \theta+\delta\theta) \\ V_d(v_d, \theta+\delta\theta) + Q_d(z_d+\delta z_d, \theta+\delta\theta) \end{bmatrix}, \end{aligned}$$

which is of order $(m+n+2)\times(d+1)$, where

$$A_d(u_d, v_d, \theta) = \begin{bmatrix} U_d(u_d,\theta) \\ V_d(v_d,\theta) \end{bmatrix} \quad \text{and} \quad B_d(z_d, \theta) = \begin{bmatrix} P_d(z_d,\theta) \\ Q_d(z_d,\theta) \end{bmatrix}.$$

It follows that

$$\begin{aligned} &\Big(A_d(u_d, v_d, \theta+\delta\theta) + B_d(z_d+\delta z_d, \theta+\delta\theta)\Big)\tilde{\mathbf{c}}_d(\theta+\delta\theta) \\ \approx\ &\left(A_d + \frac{\partial A_d}{\partial\theta}\delta\theta + B_d + \frac{\partial B_d}{\partial\theta}\delta\theta + \sum_{i=0}^{m+n-2d+1} \frac{\partial B_d}{\partial z_{d,i}}\delta z_{d,i}\right)\left(\tilde{\mathbf{c}}_d + \frac{d\tilde{\mathbf{c}}_d}{d\theta}\delta\theta\right) \\ \approx\ &(A_d + B_d)\tilde{\mathbf{c}}_d + \left(\frac{\partial A_d}{\partial\theta}\delta\theta + \frac{\partial B_d}{\partial\theta}\delta\theta + \sum_{i=0}^{m+n-2d+1} \frac{\partial B_d}{\partial z_{d,i}}\delta z_{d,i}\right)\tilde{\mathbf{c}}_d \\ &+(A_d + B_d)\frac{d\tilde{\mathbf{c}}_d}{d\theta}\delta\theta, \end{aligned} \tag{7.23}$$

to first order, where

$$\frac{\partial A_d}{\partial \theta} = \begin{bmatrix} \frac{\partial U_d}{\partial \theta} \\ \frac{\partial V_d}{\partial \theta} \end{bmatrix}, \quad \frac{\partial B_d}{\partial \theta} = \begin{bmatrix} \frac{\partial P_d}{\partial \theta} \\ \frac{\partial Q_d}{\partial \theta} \end{bmatrix}, \quad \frac{d\tilde{\mathbf{c}}_d}{d\theta} = \begin{bmatrix} dc_{d,0}\theta^{d-1} \\ (d-1)c_{d,1}\theta^{d-2} \\ \vdots \\ c_{d,d-1} \\ 0 \end{bmatrix}.$$

The matrices $\frac{\partial U_d}{\partial \theta}$, $\frac{\partial V_d}{\partial \theta}$, $\frac{\partial P_d}{\partial \theta}$ and $\frac{\partial Q_d}{\partial \theta}$ are given by, respectively,

$$\frac{\partial U_d}{\partial \theta} = \begin{bmatrix} (m-d)u_{d,0}\theta^{m-d-1} & & \\ (m-d-1)u_{d,1}\theta^{m-d-2} & \ddots & \\ \vdots & \ddots & (m-d)u_{d,0}\theta^{m-d-1} \\ u_{d,m-d-1} & \ddots & (m-d-1)u_{d,1}\theta^{m-d-2} \\ 0 & \ddots & \vdots \\ & \ddots & u_{d,m-d-1} \\ & & 0 \end{bmatrix},$$

$$\frac{\partial V_d}{\partial \theta} = \begin{bmatrix} (n-d)v_{d,0}\theta^{n-d-1} & & \\ (n-d-1)v_{d,1}\theta^{n-d-2} & \ddots & \\ \vdots & \ddots & (n-d)v_{d,0}\theta^{n-d-1} \\ v_{d,n-d-1} & \ddots & (n-d-1)v_{d,1}\theta^{n-d-2} \\ 0 & \ddots & \vdots \\ & \ddots & v_{d,n-d-1} \\ & & 0 \end{bmatrix},$$

$$\frac{\partial P_d}{\partial \theta} = \begin{bmatrix} (m-d)z_{d,0}\theta^{m-d-1} & & \\ (m-d-1)z_{d,1}\theta^{m-d-2} & \ddots & \\ \vdots & \ddots & (m-d)z_{d,0}\theta^{m-d-1} \\ z_{d,m-d-1} & \ddots & (m-d-1)z_{d,1}\theta^{m-d-2} \\ 0 & \ddots & \vdots \\ & \ddots & z_{d,m-d-1} \\ & & 0 \end{bmatrix},$$

$$\frac{\partial Q_d}{\partial \theta} = \begin{bmatrix} (n-d)z_{d,m-d+1}\theta^{n-d-1} & & \\ (n-d-1)z_{d,m-d+2}\theta^{n-d-2} & \ddots & \\ \vdots & \ddots & (n-d)z_{d,m-d+1}\theta^{n-d-1} \\ z_{d,m+n-2d} & \ddots & (n-d-1)z_{d,m-d+2}\theta^{n-d-2} \\ 0 & \ddots & \vdots \\ & \ddots & z_{d,m+n-2d} \\ & & 0 \end{bmatrix}.$$

Also, there exists a matrix $Z_d(\tilde{\mathbf{c}}_d, \theta) \in \mathbb{R}^{(m+n+2)\times(m+n-2d+2)}$,

$$Z_d(\tilde{\mathbf{c}}_d, \theta) = \begin{bmatrix} Z_{d,1}(\tilde{\mathbf{c}}_d, \theta) \\ Z_{d,2}(\tilde{\mathbf{c}}_d, \theta) \end{bmatrix}, \tag{7.24}$$

where $Z_{d,1}(\tilde{\mathbf{c}}_d, \theta) \in \mathbb{R}^{(m+1)\times(m+n-2d+2)}$ and $Z_{d,2}(\tilde{\mathbf{c}}_d, \theta) \in \mathbb{R}^{(n+1)\times(m+n-2d+2)}$, such that

$$Z_d(\tilde{\mathbf{c}}_d, \theta)z_d = B_d(z_d, \theta)\tilde{\mathbf{c}}_d, \tag{7.25}$$

for all $\tilde{\mathbf{c}}_d, z_d$ and θ. It therefore follows that on differentiating both sides of this equation with respect to z_d and keeping θ constant,

$$Z_d(\tilde{\mathbf{c}}_d, \theta)\delta z_d = \left(\sum_{i=0}^{m+n-2d+1} \frac{\partial B_d}{\partial z_{d,i}} \delta z_{d,i} \right) \tilde{\mathbf{c}}_d. \tag{7.26}$$

The matrices $Z_{d,1}(\tilde{\mathbf{c}}_d, \theta)$ and $Z_{d,2}(\tilde{\mathbf{c}}_d, \theta)$ can be expressed as functions of the Toeplitz matrices $C_{d,1}(\tilde{\mathbf{c}}_d)$ and $C_{d,2}(\tilde{\mathbf{c}}_d)$ respectively,

$$\begin{aligned} Z_{d,1}(\tilde{\mathbf{c}}_d, \theta) &= \begin{bmatrix} C_{d,1}(\tilde{\mathbf{c}}_d)\Theta_{d,1} & 0_{m+1,n-d+1} \end{bmatrix}, \\ Z_{d,2}(\tilde{\mathbf{c}}_d, \theta) &= \begin{bmatrix} 0_{n+1,m-d+1} & C_{d,2}(\tilde{\mathbf{c}}_d)\Theta_{d,2} \end{bmatrix}, \end{aligned}$$

where

$$C_{d,1}(\tilde{\mathbf{c}}_d) = \begin{bmatrix} c_{d,0}\theta^d & & \\ c_{d,1}\theta^{d-1} & \ddots & \\ \vdots & \ddots & c_{d,0}\theta^d \\ c_{d,d-1}\theta & \ddots & c_{d,1}\theta^{d-1} \\ c_{d,d} & \ddots & \vdots \\ & \ddots & c_{d,d-1}\theta \\ & & c_{d,d} \end{bmatrix} \in \mathbb{R}^{(m+1)\times(m-d+1)},$$

$$C_{d,2}(\tilde{\mathbf{c}}_d) = \begin{bmatrix} c_{d,0}\theta^d & & \\ c_{d,1}\theta^{d-1} & \ddots & \\ \vdots & \ddots & c_{d,0}\theta^d \\ c_{d,d-1}\theta & \ddots & c_{d,1}\theta^{d-1} \\ c_{d,d} & \ddots & \vdots \\ & \ddots & c_{d,d-1}\theta \\ & & c_{d,d} \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n-d+1)},$$

$$\begin{aligned} \Theta_{d,1} &= \operatorname{diag}\begin{bmatrix} \theta^{m-d} & \theta^{m-d-1} & \cdots & \theta & 1 \end{bmatrix} \in \mathbb{R}^{(m-d+1)\times(m-d+1)}, \\ \Theta_{d,2} &= \operatorname{diag}\begin{bmatrix} \theta^{n-d} & \theta^{n-d-1} & \cdots & \theta & 1 \end{bmatrix} \in \mathbb{R}^{(n-d+1)\times(n-d+1)}. \end{aligned}$$

Example 7.1. Let $m = 5, n = 3$ and $d = 2$. Thus

$$z_2 = \begin{bmatrix} z_{2,0} & z_{2,1} & z_{2,2} & z_{2,3} & z_{2,4} & z_{2,5} \end{bmatrix}^T, \qquad \tilde{\mathbf{c}}_2 = \begin{bmatrix} c_{2,0}\theta^2 & c_{2,1}\theta & c_{2,2} \end{bmatrix}^T,$$

$$P_2(z_2) = \begin{bmatrix} z_{2,0}\theta^3 & & \\ z_{2,1}\theta^2 & z_{2,0}\theta^3 & \\ z_{2,2}\theta & z_{2,1}\theta^2 & z_{2,0}\theta^3 \\ z_{2,3} & z_{2,2}\theta & z_{2,1}\theta^2 \\ & z_{2,3} & z_{2,2}\theta \\ & & z_{2,3} \end{bmatrix}, \qquad Q_2(z_2) = \begin{bmatrix} z_{2,4}\theta & & \\ z_{2,5} & z_{2,4}\theta & \\ & z_{2,5} & z_{2,4}\theta \\ & & z_{2,5} \end{bmatrix},$$

$$Z_{2,1}(\tilde{\mathbf{c}}_2, \theta)z_2 = \begin{bmatrix} c_{2,0}\theta^5 & & & & 0 & 0 \\ c_{2,1}\theta^4 & c_{2,0}\theta^4 & & & 0 & 0 \\ c_{2,2}\theta^3 & c_{2,1}\theta^3 & c_{2,0}\theta^3 & & 0 & 0 \\ & c_{2,2}\theta^2 & c_{2,1}\theta^2 & c_{2,0}\theta^2 & 0 & 0 \\ & & c_{2,2}\theta & c_{2,1}\theta & 0 & 0 \\ & & & c_{2,2} & 0 & 0 \end{bmatrix} \begin{bmatrix} z_{2,0} \\ z_{2,1} \\ z_{2,2} \\ z_{2,3} \\ z_{2,4} \\ z_{2,5} \end{bmatrix},$$

and

$$Z_{2,2}(\tilde{\mathbf{c}}_2, \theta)z_2 = \begin{bmatrix} 0 & 0 & 0 & 0 & c_{2,0}\theta^3 & \\ 0 & 0 & 0 & 0 & c_{2,1}\theta^2 & c_{2,0}\theta^2 \\ 0 & 0 & 0 & 0 & c_{2,2}\theta & c_{2,1}\theta \\ 0 & 0 & 0 & 0 & & c_{2,2} \end{bmatrix} \begin{bmatrix} z_{2,0} \\ z_{2,1} \\ z_{2,2} \\ z_{2,3} \\ z_{2,4} \\ z_{2,5} \end{bmatrix}.$$

It is readily checked that (7.25) is satisfied. □

The substitution of the first order approximations (7.18), (7.19) and (7.23), and the simplifications (7.22) and (7.26), into (7.16) and (7.17) yield

$$
\begin{aligned}
& r(z_d+\delta z_d, s_d+\delta s_d, t_d+\delta t_d, \tilde{\mathbf{c}}_d+\delta\tilde{\mathbf{c}}_d, \beta_{\mathrm{o}}+\delta\beta_{\mathrm{o}}, \theta+\delta\theta) \\
\approx\ & r(z_d, s_d, t_d, \tilde{\mathbf{c}}_d, \beta_{\mathrm{o}}, \theta) \\
& + \left[\begin{array}{c} \frac{\partial \tilde{\mathbf{f}}}{\partial\theta}\delta\theta + \frac{\partial \mathbf{p}_d}{\partial\theta}\delta\theta + S\delta s_d \\ (\alpha_{\mathrm{o}}+\beta_{\mathrm{o}})\left(\frac{\partial\tilde{\mathbf{g}}}{\partial\theta}\delta\theta + \frac{\partial\mathbf{q}_d}{\partial\theta}\delta\theta + T\delta t_d\right) + (\tilde{\mathbf{g}}+\mathbf{q}_d)\delta\beta_{\mathrm{o}} \end{array}\right] \\
& - \left(\frac{\partial A_d}{\partial\theta}\delta\theta + \frac{\partial B_d}{\partial\theta}\delta\theta\right)\tilde{\mathbf{c}}_d - Z_d\delta z_d - (A_d+B_d)\frac{d\tilde{\mathbf{c}}_d}{d\theta}\delta\theta
\end{aligned}
$$

$$
\begin{aligned}
\approx\ & r(z_d, s_d, t_d, \tilde{\mathbf{c}}_d, \beta_{\mathrm{o}}, \theta) \\
& + \left[\begin{array}{cccc} S & 0_{m+1,n+1} & 0_{m+1,1} & \frac{\partial\tilde{\mathbf{f}}}{\partial\theta} + \frac{\partial\mathbf{p}_d}{\partial\theta} \\ 0_{n+1,m+1} & (\alpha_{\mathrm{o}}+\beta_{\mathrm{o}})T & \tilde{\mathbf{g}}+\mathbf{q}_d & (\alpha_{\mathrm{o}}+\beta_{\mathrm{o}})\left(\frac{\partial\tilde{\mathbf{g}}}{\partial\theta}+\frac{\partial\mathbf{q}_d}{\partial\theta}\right) \end{array}\right] \left[\begin{array}{c} \delta s_d \\ \delta t_d \\ \delta\beta_{\mathrm{o}} \\ \delta\theta \end{array}\right] \\
& - \left(\frac{\partial A_d}{\partial\theta}\tilde{\mathbf{c}}_d + \frac{\partial B_d}{\partial\theta}\tilde{\mathbf{c}}_d\right)\delta\theta - Z_d\delta z_d - (A_d+B_d)\frac{d\tilde{\mathbf{c}}_d}{d\theta}\delta\theta. \qquad (7.27)
\end{aligned}
$$

The jth iteration, $j = 0, 1, 2, \ldots$, in the Newton-Raphson method for the solution of (7.14) therefore yields

$$
\begin{aligned}
& \left[\begin{array}{c|cccc} Z_d & \begin{array}{ccc} -S & 0_{m+1,n+1} & 0_{m+1,1} \\ 0_{n+1,m+1} & -(\alpha_{\mathrm{o}}+\beta_{\mathrm{o}})T & -(\tilde{\mathbf{g}}+\mathbf{q}_d) \end{array} \end{array}\right. \\
& \left.\begin{array}{c} -\left(\frac{\partial\tilde{\mathbf{f}}}{\partial\theta}+\frac{\partial\mathbf{p}_d}{\partial\theta}\right) + \left(\frac{\partial U_d}{\partial\theta}+\frac{\partial P_d}{\partial\theta}\right)\tilde{\mathbf{c}}_d + (U_d+P_d)\frac{d\tilde{\mathbf{c}}_d}{d\theta} \\ -(\alpha_{\mathrm{o}}+\beta_{\mathrm{o}})\left(\frac{\partial\tilde{\mathbf{g}}}{\partial\theta}+\frac{\partial\mathbf{q}_d}{\partial\theta}\right) + \left(\frac{\partial V_d}{\partial\theta}+\frac{\partial Q_d}{\partial\theta}\right)\tilde{\mathbf{c}}_d + (V_d+Q_d)\frac{d\tilde{\mathbf{c}}_d}{d\theta} \end{array}\right]^{(j)} \left[\begin{array}{c} \delta z_d \\ \delta s_d \\ \delta t_d \\ \delta\beta_{\mathrm{o}} \\ \delta\theta \end{array}\right]^{(j)} \\
=\ & r^{(j)}(z_d, s_d, t_d, \tilde{\mathbf{c}}_d, \beta_{\mathrm{o}}, \theta). \qquad (7.28)
\end{aligned}
$$

The improved estimates of $z_d, s_d, t_d, \beta_{\text{o}}$ and θ are calculated from

$$\begin{bmatrix} z_d \\ s_d \\ t_d \\ \beta_{\text{o}} \\ \theta \end{bmatrix}^{(j+1)} = \begin{bmatrix} z_d \\ s_d \\ t_d \\ \beta_{\text{o}} \\ \theta \end{bmatrix}^{(j)} + \begin{bmatrix} \delta z_d \\ \delta s_d \\ \delta t_d \\ \delta\beta_{\text{o}} \\ \delta\theta \end{bmatrix}^{(j)},$$

where the initial values in the iteration are

$$z_d^{(0)} = 0, \qquad s_d^{(0)} = 0, \qquad t_d^{(0)} = 0, \qquad \beta_{\text{o}}^{(0)} = 0, \qquad \theta^{(0)} = \theta_{\text{o}}, \tag{7.29}$$

and $\tilde{\mathbf{c}}_d^{(0)}(\theta_{\text{o}})$, the initial value of $\tilde{\mathbf{c}}_d(\theta)$, is given by the least squares solution of (7.12),

$$\tilde{\mathbf{c}}_d^{(0)}(\theta_{\text{o}}) = \begin{bmatrix} U_d(u_d, \theta_{\text{o}}) \\ V_d(v_d, \theta_{\text{o}}) \end{bmatrix}^{\dagger} \begin{bmatrix} \tilde{\mathbf{f}}(\theta_{\text{o}}) \\ \alpha_{\text{o}}\tilde{\mathbf{g}}(\theta_{\text{o}}) \end{bmatrix}, \tag{7.30}$$

where $X^{\dagger} = (X^T X)^{-1} X^T$.

The initial value of the residual is therefore

$$\begin{aligned} & r(0, 0, 0, \tilde{\mathbf{c}}_d, 0, \theta_{\text{o}}) \\ = & \begin{bmatrix} \tilde{\mathbf{f}}(\theta_{\text{o}}) \\ \alpha_{\text{o}}\tilde{\mathbf{g}}(\theta_{\text{o}}) \end{bmatrix} - \begin{bmatrix} U_d(u_d, \theta_{\text{o}}) \\ V_d(v_d, \theta_{\text{o}}) \end{bmatrix} \tilde{\mathbf{c}}_d^{(0)}(\theta_{\text{o}}) \\ = & \left[I_{m+n+2} - \begin{bmatrix} U_d(u_d, \theta_{\text{o}}) \\ V_d(v_d, \theta_{\text{o}}) \end{bmatrix} \begin{bmatrix} U_d(u_d, \theta_{\text{o}}) \\ V_d(v_d, \theta_{\text{o}}) \end{bmatrix}^{\dagger} \right] \begin{bmatrix} \tilde{\mathbf{f}}(\theta_{\text{o}}) \\ \alpha_{\text{o}}\tilde{\mathbf{g}}(\theta_{\text{o}}) \end{bmatrix}. \end{aligned} \tag{7.31}$$

Equation (7.28) is of the form

$$Cw = g, \tag{7.32}$$

where $C \in \mathbb{R}^{(m+n+2)\times(2m+2n-2d+6)}$, $w \in \mathbb{R}^{2m+2n-2d+6}$ and $g \in \mathbb{R}^{m+n+2}$ are given by

$$C = \left[\begin{array}{c|ccc} Z_d & -S & 0_{m+1,n+1} & 0_{m+1,1} \\ & 0_{n+1,m+1} & -(\alpha_o + \beta_o)T & -(\tilde{\mathbf{g}} + \mathbf{q}_d) \end{array} \right.$$

$$\left. \begin{array}{c} -\left(\frac{\partial \tilde{\mathbf{f}}}{\partial \theta} + \frac{\partial \mathbf{p}_d}{\partial \theta}\right) + \left(\frac{\partial U_d}{\partial \theta} + \frac{\partial P_d}{\partial \theta}\right)\tilde{\mathbf{c}}_d + (U_d + P_d)\frac{d\tilde{\mathbf{c}}_d}{d\theta} \\ -(\alpha_o + \beta_o)\left(\frac{\partial \tilde{\mathbf{g}}}{\partial \theta} + \frac{\partial \mathbf{q}_d}{\partial \theta}\right) + \left(\frac{\partial V_d}{\partial \theta} + \frac{\partial Q_d}{\partial \theta}\right)\tilde{\mathbf{c}}_d + (V_d + Q_d)\frac{d\tilde{\mathbf{c}}_d}{d\theta} \end{array} \right]^{(j)}, \qquad (7.33)$$

$$w = \left[\begin{array}{ccccc} \delta z_d^{(j)} & \delta s_d^{(j)} & \delta t_d^{(j)} & \delta \beta_o^{(j)} & \delta \theta^{(j)} \end{array} \right]^T, \qquad (7.34)$$

$$g = r^{(j)}(z_d, s_d, t_d, \tilde{\mathbf{c}}_d, \beta_o, \theta). \qquad (7.35)$$

It is clear that (7.32) is under-determined and it therefore has an infinite number of solutions. It is desired to compute the solution of (7.32) that is closest to the given inexact data, and it is therefore required to minimise

$$\left\| \begin{bmatrix} z_d^{(j+1)} & - & z_d^{(0)} \\ s_d^{(j+1)} & - & s_d^{(0)} \\ t_d^{(j+1)} & - & t_d^{(0)} \\ \beta_o^{(j+1)} & - & \beta_o^{(0)} \\ \theta^{(j+1)} & - & \theta_o \end{bmatrix} \right\| = \left\| \begin{bmatrix} z_d^{(j)} & + & \delta z_d^{(j)} \\ s_d^{(j)} & + & \delta s_d^{(j)} \\ t_d^{(j)} & + & \delta t_d^{(j)} \\ \beta_o^{(j)} & + & \delta \beta_o^{(j)} \\ \theta^{(j)} & + & \delta \theta^{(j)} - \theta_o \end{bmatrix} \right\|$$

$$= \left\| I_{2m+2n-2d+6} \begin{bmatrix} \delta z_d^{(j)} \\ \delta s_d^{(j)} \\ \delta t_d^{(j)} \\ \delta \beta_o^{(j)} \\ \delta \theta^{(j)} \end{bmatrix} - \begin{bmatrix} -z_d^{(j)} \\ -s_d^{(j)} \\ -t_d^{(j)} \\ -\beta_o^{(j)} \\ -(\theta^{(j)} - \theta_o) \end{bmatrix} \right\|$$

subject to (7.32), using (7.29). If E and f are defined as

$$E = I_{2m+2n-2d+6} \qquad (7.36)$$

$$f = -\left[\begin{array}{ccccc} z_d^{(j)} & s_d^{(j)} & t_d^{(j)} & \beta_o^{(j)} & \theta^{(j)} - \theta_o \end{array} \right]^T \in \mathbb{R}^{2m+2n-2d+6}, \qquad (7.37)$$

then it is required to solve, at each iteration, the least squares equality (LSE) constrained problem,

$$\min_{w} \|Ew - f\| \qquad \text{subject to} \quad Cw = g,$$

where C, f and g are updated between successive iterations, and the initial value of f and g are $f = 0$ and $g = r(0, 0, 0, \tilde{c}_d, 0, \theta_o)$, which is defined in (7.31).

The QR decomposition can be used to solve the LSE problem at each iteration [24]. Specifically, let

$$C^T = QR = Q \begin{bmatrix} R_1 \\ 0 \end{bmatrix}, \tag{7.38}$$

where $Q \in \mathbb{R}^{(2m+2n-2d+6)\times(2m+2n-2d+6)}$ is an orthogonal matrix, $R \in \mathbb{R}^{(2m+2n-2d+6)\times(m+n+2}$ and $R_1 \in \mathbb{R}^{(m+n+2)\times(m+n+2)}$ is a non-singular upper triangular matrix, be the QR decomposition of C^T. If

$$Q^T w = \begin{bmatrix} \upsilon \\ \nu \end{bmatrix},$$

where $\upsilon \in \mathbb{R}^{m+n+2}$ and $\nu \in \mathbb{R}^{m+n-2d+4}$, the constraint $Cw = g$ becomes

$$R_1^T \upsilon = g,$$

due to

$$\begin{aligned} Cw &= (QR)^T w \\ &= R^T Q^T w \\ &= \begin{bmatrix} R_1^T & 0 \end{bmatrix} \begin{bmatrix} \upsilon \\ \nu \end{bmatrix} \\ &= R_1^T \upsilon, \end{aligned}$$

and thus $\upsilon = R_1^{-T} g$.

Similarly, if

$$EQ = \begin{bmatrix} E_1 & E_2 \end{bmatrix},$$

where $E_1 \in \mathbb{R}^{(2m+2n-2d+6)\times(m+n+2)}$ and $E_2 \in \mathbb{R}^{(2m+2n-2d+6)\times(m+n-2d+4)}$, the objective function $\|Ew = f\|$ becomes

$$\begin{aligned} \|Ew - f\| &= \|EQQ^T w - f\| \\ &= \left\| \begin{bmatrix} E_1 & E_2 \end{bmatrix} \begin{bmatrix} v \\ \nu \end{bmatrix} - f \right\| \\ &= \|E_1 v + E_2 \nu - f\| \\ &= \|E_2 \nu - (f - E_1 v)\|, \end{aligned}$$

and thus it is minimised when

$$\nu = E_2^{\dagger}(f - E_1 v), \tag{7.39}$$

from which it follows that the solution of the LSE problem is

$$w = Q \begin{bmatrix} v \\ \nu \end{bmatrix}. \tag{7.40}$$

Algorithm 7.1 shows the implementation of this algorithm for the calculation of an approximate GCD of two inexact polynomials using the method SNTLN.

Algorithm 7.1: The calculation of an approximate GCD of two inexact polynomials

Input Inexact polynomials $f(x)$ and $g(x)$ and d, the degree of their approximate GCD.

Output An approximate GCD of $f(x)$ and $g(x)$, and the modified polynomials $\breve{f}(y)$ and $\breve{g}(y)$ in the independent variable y, after the substitution $x = \theta y$.

Begin

1. Calculate α_o and θ_o using methods of linear programming, and preprocess $f(x)$ and $g(x)$ to yield the polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$, as shown in Section 4.4.

2. % Calculation only for the dth subresultant matrix $S_d(\tilde{f}, \tilde{g})$.

   % Initialise the data for the solution of the LSE problem.

   (a.1) Calculate the coefficients $u_{d,i}$ and $v_{d,i}$ of $u(x)$ and $v(x)$ using $S_d(\tilde{f}, \tilde{g})$, respectively.

   (a.2) Form the matrices $U_d(u_d, \theta_o)$ and $V_d(v_d, \theta_o)$, and their derivative $\frac{\partial U_d}{\partial \theta}$ and $\frac{\partial V_d}{\partial \theta}$ at $\theta = \theta_o$.

   (a.3) Form the vectors $\tilde{\mathbf{f}}(\theta_o)$ and $\tilde{\mathbf{g}}(\theta_o)$, and evaluate $\frac{\partial \tilde{\mathbf{f}}}{\partial \theta}$ and $\frac{\partial \tilde{\mathbf{f}}}{\partial \theta}$ at $\theta = \theta_o$.

   (a.4) Calculate the initial estimate $\tilde{\mathbf{c}}_d^{(0)}(\theta_o)$ of $\tilde{\mathbf{c}}_d(\theta)$ from (7.30), and the initial residual $r(0, 0, 0, \tilde{\mathbf{c}}_d, 0, \theta_o)$ from (7.31).

   (a.5) Calculate the initial values of the derivative $\frac{d\tilde{\mathbf{c}}_d}{d\theta}$ for $\theta = \theta_o$.

   (a.6) Initialise some variables in (7.29), and set

   $$P_d(z_d, \theta_o) = 0, \qquad Q_d(z_d, \theta_o) = 0, \qquad \mathbf{p}_d = 0, \qquad \mathbf{q}_d = 0,$$

   and their derivative

   $$\frac{\partial P_d}{\partial \theta} = 0, \qquad \frac{\partial Q_d}{\partial \theta} = 0, \qquad \frac{\partial \mathbf{p}_d}{\partial \theta} = 0, \qquad \frac{\partial \mathbf{q}_d}{\partial \theta} = 0.$$

   (a.7) Calculate $Z_d(\tilde{\mathbf{c}}_d^{(0)}, \theta_o)$, where $Z_d(\tilde{\mathbf{c}}_d, \theta)$ is defined in (7.24).

   (a.8) Set $g = r(0, 0, 0, \tilde{\mathbf{c}}_d, 0, \theta_o)$, $f = 0$ and initialise S and T, which are defined in (7.20) and (7.21) respectively. Initialise C from (7.33), and define E, which is defined in (7.36).

3. Iteration = 0. % The counter for the number of iterations

Repeat % Use the QR decomposition to solve the LSE problem at each iteration

(b.1) Iteration = Iteration + 1.

(b.2) Compute the QR decomposition of C^T from (7.38) in order to obtain Q and R_1,

(b.3) Set $v = R_1^{-T} g$.

(b.4) Compute ν using (7.39).

(b.5) Compute the solution w from (7.40)

(b.6) Set

$$z_d := z_d + \delta z_d, \qquad s_d := s_d + \delta s_d, \qquad t_d := t_d + \delta t_d$$

and

$$\beta_o := \beta_o + \delta\beta_o \qquad \theta := \theta + \delta\theta.$$

(b.7) Update $\tilde{\mathbf{c}}_d(\theta)$ from (7.13) and calculate the derivative $\frac{d\tilde{\mathbf{c}}_d}{d\theta}$.

(b.8) Update $\tilde{\mathbf{f}}(\theta)$ and $\tilde{\mathbf{g}}(\theta)$, and evaluate $\frac{\partial\tilde{\mathbf{f}}}{\partial\theta}$ and $\frac{\partial\tilde{\mathbf{f}}}{\partial\theta}$ from θ.

Update $P_d(z_d, \theta_o), Q_d(z_d, \theta_o)$, $\frac{\partial P_d}{\partial\theta}$ and $\frac{\partial Q_d}{\partial\theta}$ from z_d and θ.

Update $\mathbf{p}_d$ and $\frac{\partial\mathbf{p}_d}{\partial\theta}$ from s_d and θ.

Update $\mathbf{q}_d$ and $\frac{\partial\mathbf{q}_d}{\partial\theta}$ from t_d and θ. Update $Z_d(\tilde{\mathbf{c}}_d, \theta)$ from $\tilde{\mathbf{c}}_d$ and θ.

(b.9) Update S,T and C, which are defined in (7.20), (7.21) and (7.33), respectively.

(b.10) Compute the residual $r(z_d, s_d, t_d, \tilde{\mathbf{c}}_d, \beta_o, \theta)$, which is defined in (7.15), and thus update g. Update f from z_d, s_d, t_d, β_o and θ.

(b.11) Calculate

$$e_d := \begin{bmatrix} \tilde{\mathbf{f}}(\theta) + \mathbf{p}_d(s_d, \theta) \\ (\alpha_o + \beta_o)(\tilde{\mathbf{g}}(\theta) + \mathbf{q}_d(t_d, \theta)) \end{bmatrix}$$

Until $\frac{\|r(z_d, s_d, t_d, \tilde{\mathbf{c}}_d, \beta_o, \theta)\|}{\|e_d\|} \leq 10^{-16}$ **OR** Iteration > 50.

4. Set $res_d = \frac{\|r(z_d, s_d, t_d, \tilde{\mathbf{c}}_d, \beta_o, \theta)\|}{\|e_d\|}$, $\alpha = \alpha_o + \beta_o, \check{\mathbf{f}} = \tilde{\mathbf{f}}(\theta) + \mathbf{p}_d(s_d, \theta), \check{\mathbf{g}} = \tilde{\mathbf{g}}(\theta) + \mathbf{q}_d(t_d, \theta)$ and $\check{\mathbf{d}} = \tilde{\mathbf{c}}_d(\theta)$.

End

7.2 Examples

This section contains several examples that show the use of the method of SNTLN for the calculation of an approximate GCD of two inexact polynomials. It is necessary to explain some notation that is used in the following examples. If d is the degree of an approximate GCD, then

- $S_d(f, g)$, the dth subresultant matrix, is formed from the given inexact polynomials $f(x)$ and $g(x)$,
- $S_d(\tilde{f}, \alpha_o \tilde{g})$ is formed from the processed polynomials $\tilde{f}(y)$ and $\alpha_o \tilde{g}(y)$, which are defined in (7.2).
- $S_d(\check{f}, \alpha \check{g})$ is formed from the polynomials $\check{f}(y)$ and $\alpha \check{g}(y)$, which are calculated from the method of SNTLN, that is, $\check{f}(y)$ and $\check{g}(y)$ have a non-constant GCD.

Example 7.2. Consider the exact polynomials $\hat{f}(x)$ and $\hat{g}(x)$, whose roots and multiplicities are specified in Table 7.1. It is seen that $m = 26, n = 18$ and the degree of their GCD is $\hat{d} = 11$.

Root of $\hat{f}(x)$	Multiplicity
-3.5540e-001	3
-9.7181e+000	1
2.4576e+000	3
-5.3781e+000	5
5.4870e-001	1
4.4998e+000	1
2.1483e+000	5
1.7673e+000	2
-1.3313e+000	1
-5.1165e+000	4

Root of $\hat{g}(x)$	Multiplicity
-3.5540e-001	5
-9.7181e+000	1
2.4576e+000	2
-5.3781e+000	3
5.4870e-001	3
4.4998e+000	4

Table 7.1: The roots and multiplicities of $\hat{f}(x)$ and $\hat{g}(x)$ for Example 7.2.

Uniformly distributed random noise was added to each polynomial, such that the componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^8$. The noisy polynomials were then preprocessed by the operations described in Section 4.4, thereby yielding the Sylvester matrix $S(\tilde{f}, \alpha_o \tilde{g})$, where $\alpha_o = 10.2108$ and $\theta_o = 2.1097$.

Figure 7.1(i) shows the variation of $\log \phi_d$ and $\log r_d$ with k, and it is seen that $d_\phi = d_r = \hat{d} = 11$ [1], such that these values are clearly defined by Methods 4 and 5, which are described in Section 6.1. Figure 7.1(ii), (iii) and (iv) show the normalised singular values of $S_d(\check{f}, \alpha\check{g})$, $S_d(f, g)$ and $S_d(\tilde{f}, \alpha_o\tilde{g})$, where $\check{f} = \check{f}(y)$ and $\alpha\check{g} = \alpha\check{g}(y)$, are calculated from the method of SNTLN using Algorithm 7.1. It is seen that the method of SNTLN significantly improves the result because

[1]The degree of an approximate GCD is computed by the method of the first principal angle and the method of residual.

- the rank of $S_d(\breve{f}, \alpha\breve{g})$ is correct, and it is clearly defined,
- the rank of $S_d(f, g)$ is incorrect, but it is defined,
- the rank of $S_d(\tilde{f}, \alpha_o\tilde{g})$ is correct, but it is poorly defined.

□

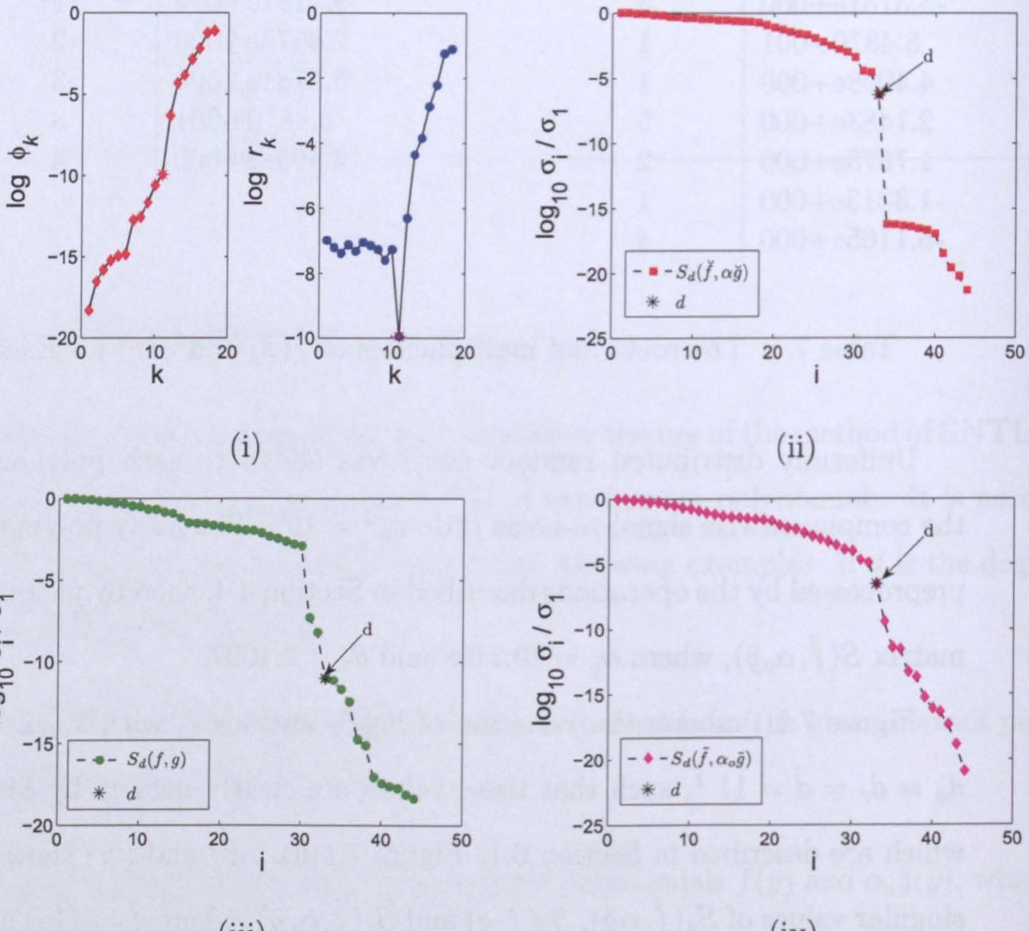


Figure 7.1: (i) The variation of $\log \phi_k$ and $\log r_k$ with k, and the normalised singular values of (ii) $S_d(\breve{f}, \alpha\breve{g})$, (iii)$S_d(f, g)$, (iv) $S_d(\tilde{f}, \alpha_o\tilde{g})$, with $\varepsilon_c = 10^{-8}$ for Example 7.2.

Example 7.3. Consider the exact polynomials $\hat{f}(x)$ and $\hat{g}(x)$, whose roots and multiplicities are specified in Table 7.2. It is seen that $m = 16, n = 23$ and the degree of their GCD is $\hat{d} = 13$.

Root of $\hat{f}(x)$	Multiplicity
6.5743e+000	3
3.7189e+000	2
-4.6535e+000	2
9.3897e+000	4
-6.3245e+000	2
-4.0012e+000	3

Root of $\hat{g}(x)$	Multiplicity
6.5743e+000	3
3.7189e+000	5
-4.6535e+000	2
9.3897e+000	1
-6.3245e+000	5
-4.0012e+000	4
4.4140e-001	3

Table 7.2: The roots and multiplicities of $\hat{f}(x)$ and $\hat{g}(x)$ for Example 7.3.

Noise was added in the componentwise sense to each polynomial, such that the componentwise signal-to-noise ratio ε_c^{-1} is 10^8. The noisy polynomials were then normalised by the geometric means of their coefficients, after which they were preprocessed, thereby yielding the Sylvester matrix $S(\tilde{f}, \alpha_o \tilde{g})$, where $\alpha_o = 9.9244e-003$ and $\theta_o = 3.6608$.

It is shown in Figure 7.2(i) that the degree of an approximate GCD is clearly defined using Methods 4 and 5, such that $d_\phi = d_r = \hat{d} = 13$. Also it is seen from Figure 7.2(ii) that the rank loss of $S_d(\check{f}, \alpha\check{g})$ is more clearly defined at $d = 13$ than the rank loss of $S_d(\tilde{f}, \alpha_o\tilde{g})$, that is, it is important that the method of SNTLN should be used to calculate the modified polynomials $\check{f} = \check{f}(y)$ and $\alpha\check{g} = \alpha\check{g}(y)$, and an approximate GCD.

Since $m = 16$ and $n = 23$, the integer k ranges from 1 to $\min(m, n)$, and thus $k = 1, \ldots, \min(m, n)$. The experiment is repeated for each value of k, and thus res_k and the value of $\|Ew - f\|$ are calculated using Algorithm 7.1, assuming Method 4

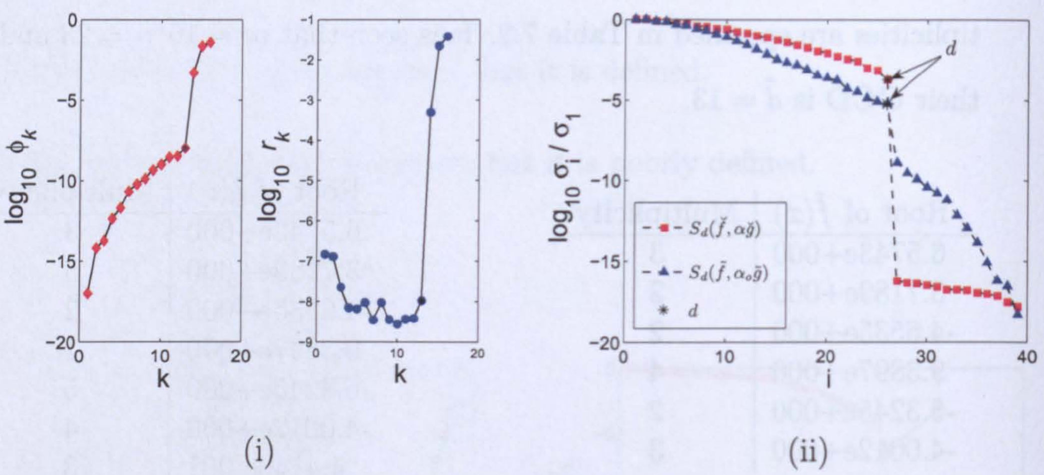


Figure 7.2: (i) The variation of $\log \phi_k$ and $\log r_k$ with k, (ii) the normalised singular values of $S_d(\tilde{f}, \alpha\breve{g})$ and $S_d(\tilde{f}, \alpha_o\tilde{g})$, with $\varepsilon_c = 10^{-8}$ for Example 7.3.

is used to determined which columns of $S_k(\tilde{f}, \alpha_o\tilde{g})$ are removed to right hand side of (7.6). Figure 7.3(i) shows that the variation of res_d with k enables the degree d of an approximate GCD of $\tilde{f}(y)$ and $\tilde{g}(y)$ to be calculated because

$$res_k \approx 0, \qquad k = 1, \ldots, d,$$

$$res_k \gg 0, \qquad k = d+1, \ldots, 16,$$

such that $d = 13$, and Figure 7.3(ii) shows that the degree d is also equal to 13, the value of k for which $\|Ew - f\|$ achieves its minimum value. Similar results are shown in Figure 7.3(iii) and (iv) when the columns of $S_k(\tilde{f}, \alpha_o\tilde{g})$ are determined by Method 5.

It would appear from Figure 7.3 that the method of SNTLN can be used to calculate the degree d of an approximate GCD of $\tilde{f}(y)$ and $\tilde{g}(y)$ because:

- $res_k \approx 0$ when $k < d$ and res_k has maximum gradient at $k = d$,

- $\|Ew - f\|$ achieves its minimum value at $k = d$.

It is however now demonstrated that this is not necessarily true.

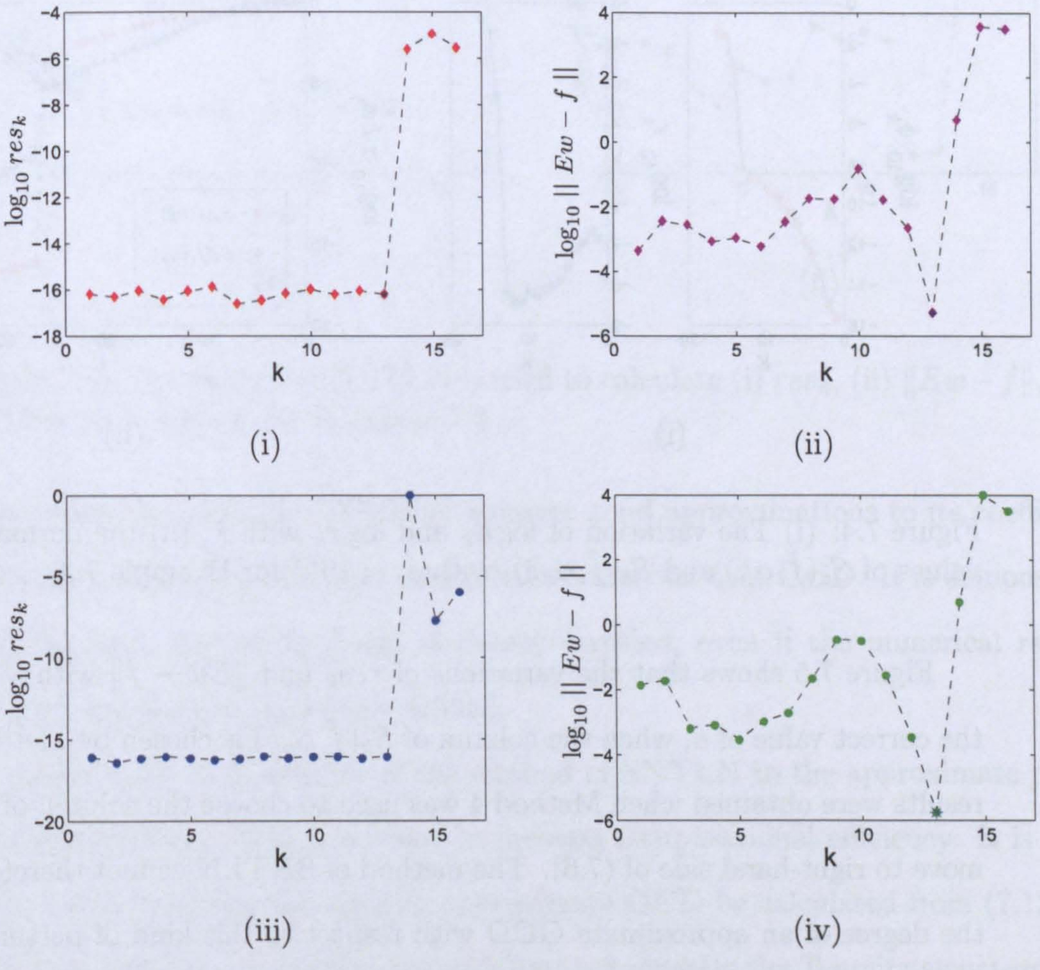
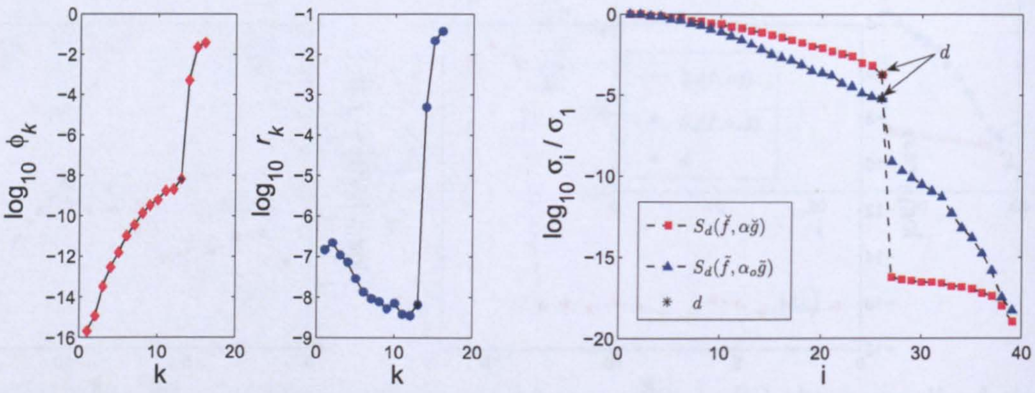


Figure 7.3: The method of SNTLN used to calculate (i) res_k, (ii) $\|Ew - f\|$, based on Method 4, (iii) res_k, (iv) $\|Ew - f\|$, based on Method 5, with k, for Example 7.3.

If another perturbation is added to $\hat{f}(x)$ and $\hat{g}(x)$ with the same componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^8$, and the new inexact polynomials $f(x)$ and $g(x)$ are preprocessed by the operations described in Section 4.4, then new polynomials $\tilde{f}(y)$ and $\tilde{g}(y)$ are obtained. It is seen that Figure 7.4 is similar to Figure 7.2, which means the determination of d by Methods 4 and 5 and the rank estimate of $S_d(\breve{f}, \alpha\breve{g})$ are stable with respect to perturbations in the coefficients of the polynomials.



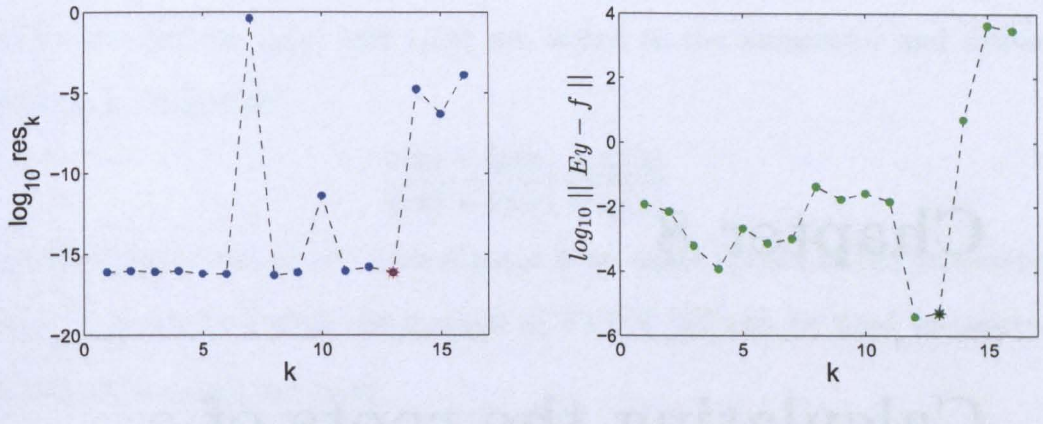
(i) (ii)

Figure 7.4: (i) The variation of $\log \phi_k$ and $\log r_k$ with k, (ii) the normalised singular values of $S_d(\breve{f}, \alpha\breve{g})$ and $S_d(\tilde{f}, \alpha_o\tilde{g})$, with $\varepsilon_c = 10^{-8}$ for Example 7.3.

Figure 7.5 shows that the variations of res_k and $\|Ew - f\|$ with k fail to return the correct value of d, when the column of $S_k(\tilde{f}, \alpha_o\tilde{g})$ is chosen by Method 5. Similar results were obtained when Method 4 was used to choose the column of $S_k(\tilde{f}, \alpha_o\tilde{g})$ to move to right-hand side of (7.6). The method of SNTLN cannot therefore determine the degree of an approximate GCD with respect to this kind of perturbation in the coefficients of the polynomials. □

7.3 Summary

This chapter has considered the use of the method of SNTLN applied to the approximate polynomial factorisation of two inexact polynomials for the calculation of an approximate GCD. If the degree of an approximate GCD is given, it has been



(i) (ii)

Figure 7.5: The method of SNTLN is used to calculate (i) res_k, (ii) $\|Ew - f\|$, based on Method 5, with k, for Example 7.3.

shown that the method of SNTLN recovers good approximations to its coefficients, and polynomials $\breve{f}(y)$ and $\breve{g}(y)$, which have a non-constant GCD. It is demonstrated that the rank loss of $S_d(\breve{f}, \alpha\breve{g})$ is clearly certified, even if the numerical ranks of $S_d(f, g)$ and $S_d(\tilde{f}, \alpha_o\tilde{g})$ are not defined.

Apart from the feasibility of the method of SNTLN to the approximate polynomial factorisation, there is a scope to increase computational efficiency. It is known that this method requires that an approximate GCD be calculated from (7.12), and thus it is useful to investigate an algorithm that exploits the Toeplitz structure in the left hand side of (7.12).

Chapter 8

Calculating the roots of a polynomial

A simple polynomial root solver has been introduced in Chapter 3 to calculate the multiplicities of the roots through a sequence of approximate GCD computations, after which the values of the roots are calculated through polynomial division operations. Since the calculation of an approximation GCD of an inexact polynomial pair has been considered in Chapter 7, it is now appropriate to consider the polynomial division $p(x)/q(x)$, which reduces to the deconvolution of $p(x)$ and $q(x)$.

Assume that the ratio of $p(x)/q(x)$ is a polynomial, and random perturbations $\delta p(x)$ and $\delta q(x)$ applied to $p(x)$ and $q(x)$ respectively, cause

$$\frac{p(x) + \delta p(x)}{q(x) + \delta q(x)}$$

to be a rational function. This means that deconvolution of two polynomials is an ill-posed problem, and thus it is difficult to obtain a computationally stable solution. A structure preserving matrix method is therefore used to guarantee that deconvolution of two polynomials is a polynomial and not a rational function. In this case, structured perturbations $z_p(x)$ and $z_q(x)$ are added to the numerator and denominator respectively, such that

$$\frac{p(x) + \delta p(x) + z_p(x)}{q(x) + \delta q(x) + z_q(x)}$$

is a polynomial, that is, the denominator is an exact divisor of the numerator. It is shown in Section 8.1 that the method of STLN [56] can be used to construct the perturbations $z_p(x)$ and $z_q(x)$.

With reference to the designed root solver, it is then necessary to solve a sequence of polynomial equations, all of whose roots are simple. The solutions of these equations are then refined by the method of non-linear least squares (NLLS). This work using the method of NLLS follows closely the work of Zeng [72], and it is shown in Section 8.2 that the equation that is solved by the method NLLS is based on the pejorative manifold of a polynomial that has multiple roots. This manifold has been introduced in Section 2.3 in the consideration of the numerical stability of the roots of a polynomial.

8.1 The deconvolution of two polynomials

This section describes the method of STLN for the solution of the deconvolution problem. It is stated in Section 3.2 that a sequence of deconvolutions are required for the polynomial root solver, such that a polynomial is involved for the kth and $(k+1)$th deconvolutions. It is therefore necessary to consider the application of a linear structure preserving matrix method for several deconvolutions together,

$$\tau_i(x) = \frac{d_{i-1}(x)}{d_i(x)}, \qquad i = 1, \dots, m_*, \tag{8.1}$$

where m_* is an arbitrary number, and the polynomial $d_k(x)$ appears in the kth and $(k+1)$th deconvolutions. The degrees of these polynomials are

$$\begin{aligned} \deg d_i(x) &= m_i, \qquad && i = 0, \ldots, m_*, \\ \deg \tau_i(x) &= n_i, && i = 1, \ldots, m_*, \end{aligned}$$

where

$$\sum_{i=0}^{m_*-1} (m_i + 1) = M, \qquad \sum_{i=0}^{m_*} (m_i + 1) = M_1, \qquad M_1 = M + (m_{m_*} + 1),$$

and

$$\sum_{i=1}^{m_*} (n_i + 1) = N, \qquad n_i = m_{i-1} - m_i, \qquad i = 1, \ldots, m_*.$$

If $\mathbf{d}_i \in \mathbb{R}^{m_i+1}, i = 0, \ldots, m_*$, and $\Gamma_i \in \mathbb{R}^{n_i+1}, i = 1, \ldots, m_*$, are the vectors of the coefficients of $d_i(x)$ and $\tau_i(x)$, respectively, then (8.1) can be written in matrix form as

$$\begin{bmatrix} D_1(\mathbf{d}_1) & & & & \\ & D_2(\mathbf{d}_2) & & & \\ & & \ddots & & \\ & & & D_{m_*-1}(\mathbf{d}_{m_*-1}) & \\ & & & & D_{m_*}(\mathbf{d}_{m_*}) \end{bmatrix} \begin{bmatrix} \Gamma_1 \\ \Gamma_2 \\ \vdots \\ \Gamma_{m_*-1} \\ \Gamma_{m_*} \end{bmatrix} \approx \begin{bmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_{m_*-2} \\ \mathbf{d}_{m_*-1} \end{bmatrix} \tag{8.2}$$

where

$$D_i(\mathbf{d}_i) \in \mathbb{R}^{(m_{i-1}+1)\times(n_i+1)}, \qquad i = 1, \ldots, m_*,$$

and the coefficient matrix in (8.2) is of order $M \times N$.

It is assumed that the coefficients of the polynomials are inexact, and thus (8.2) does not possess an exact solution. It is therefore necessary to add a structured matrix to the coefficient matrix, and a structured vector to the right hand side, of this equation. In particular, let $\mathbf{z}_i \in \mathbb{R}^{m_i+1}$ be the vector of perturbations added to the vector $\mathbf{d}_i$ of coefficients of the polynomial $d_i(x), i = 0, \ldots, m_*$, and let

$$\mathbf{z} = \begin{bmatrix} \mathbf{z}_0 & \mathbf{z}_1 & \cdots & \mathbf{z}_{m_*} \end{bmatrix}^T \in \mathbb{R}^{M_1},$$

where

$$\begin{aligned}
\mathbf{z}_0 &= \begin{bmatrix} z_0 & z_1 & \cdots & z_{m_0} \end{bmatrix}^T \in \mathbb{R}^{m_0+1}, \\
\mathbf{z}_1 &= \begin{bmatrix} z_{m_0+1} & z_{m_0+2} & \cdots & z_{m_0+m_1+1} \end{bmatrix}^T \in \mathbb{R}^{m_1+1}, \\
&\vdots \\
\mathbf{z}_i &= \begin{bmatrix} z_{m_0+\cdots+m_{i-1}+i} & \cdots & z_{m_0+\cdots+m_i+i} \end{bmatrix}^T \in \mathbb{R}^{m_i+1}, \\
&\vdots \\
\mathbf{z}_{m_*} &= \begin{bmatrix} z_M & z_{M+1} & \cdots & z_{M_1-1} \end{bmatrix}^T \in \mathbb{R}^{m_{m_*}+1}.
\end{aligned}$$

A matrix of structured perturbations is added to each of the Toeplitz matrices $D_i(\mathbf{d}_i), i = 1, \ldots, m_*$, and thus the coefficient matrix in (8.2) is replaced by

$$\begin{aligned}
B(\mathbf{z}_1, \ldots, \mathbf{z}_{m_*}) &= D(\mathbf{d}_1, \ldots, \mathbf{d}_{m_*}) + E(\mathbf{z}_1, \ldots, \mathbf{z}_{m_*}) \\
&= \begin{bmatrix} D_1(\mathbf{d}_1) & & & & \\ & D_2(\mathbf{d}_2) & & & \\ & & \ddots & & \\ & & & D_{m_*-1}(\mathbf{d}_{m_*-1}) & \\ & & & & D_{m_*}(\mathbf{d}_{m_*}) \end{bmatrix} + \\
&\quad \begin{bmatrix} E_1(\mathbf{z}_1) & & & & \\ & E_2(\mathbf{z}_2) & & & \\ & & \ddots & & \\ & & & E_{m_*-1}(\mathbf{z}_{m_*-1}) & \\ & & & & E_{m_*}(\mathbf{z}_{m_*}) \end{bmatrix},
\end{aligned}$$

where $B(\mathbf{z}_1,\ldots,\mathbf{z}_{m_*}) \in \mathbb{R}^{M\times N}$ and $E_i(\mathbf{z}_i) \in \mathbb{R}^{(m_{i-1}+1)\times(n_i+1)}, i = 1,\ldots,m_*$, are Toeplitz matrices.

Consider now the vector on the right hand side of (8.2), the perturbed from of which is

$$\begin{bmatrix} \mathbf{d}_0+\mathbf{z}_0 \\ \mathbf{d}_1+\mathbf{z}_1 \\ \vdots \\ \mathbf{d}_{m_*-2}+\mathbf{z}_{m_*-2} \\ \mathbf{d}_{m_*-1}+\mathbf{z}_{m_*-1} \end{bmatrix} = \begin{bmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_{m_*-2} \\ \mathbf{d}_{m_*-1} \end{bmatrix} + \begin{bmatrix} I_M & \vdots & 0 \end{bmatrix} \begin{bmatrix} \mathbf{z}_0 \\ \mathbf{z}_1 \\ \vdots \\ \mathbf{z}_{m_*-1} \\ \cdots \\ \mathbf{z}_{m_*} \end{bmatrix} = \begin{bmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_{m_*-2} \\ \mathbf{d}_{m_*-1} \end{bmatrix} + O\mathbf{z},$$

where

$$O = \begin{bmatrix} I_M & \vdots & 0 \end{bmatrix} \in \mathbb{R}^{M\times M_1}.$$

It follows that the corrected form of (8.2) is

$$\Big(D(\mathbf{d}_1,\ldots,\mathbf{d}_{m_*}) + E(\mathbf{z}_1,\ldots,\mathbf{z}_{m_*})\Big)\,\Gamma = \mathbf{d} + O\mathbf{z}, \tag{8.3}$$

where

$$\Gamma = \begin{bmatrix} \Gamma_1 & \Gamma_2 & \ldots & \Gamma_{m_*-1} & \Gamma_{m_*} \end{bmatrix}^T \in \mathbb{R}^N$$

and

$$\mathbf{d} = \begin{bmatrix} \mathbf{d}_0 & \mathbf{d}_1 & \ldots & \mathbf{d}_{m_*-2} & \mathbf{d}_{m_*-1} \end{bmatrix}^T \in \mathbb{R}^M.$$

The residual due to an approximate solution of (8.3) is

$$r = r(\mathbf{z}) = \mathbf{d} + O\mathbf{z} - \Big(D(\mathbf{d}_1,\ldots,\mathbf{d}_{m_*}) + E(\mathbf{z}_1,\ldots,\mathbf{z}_{m_*})\Big)\,\Gamma, \tag{8.4}$$

and thus a first order Taylor expansion of $r(\mathbf{z})$ yields

$$\begin{aligned}
r(\mathbf{z}+\delta\mathbf{z}) &= \Big(\mathbf{d}+O(\mathbf{z}+\delta\mathbf{z})\Big) \\
&\quad -\Big(D(\mathbf{d}_1,\dots,\mathbf{d}_{m_*})+E(\mathbf{z}_1+\delta\mathbf{z}_1,\dots,\mathbf{z}_{m_*}+\delta\mathbf{z}_{m_*})\Big)(\Gamma+\delta\Gamma) \\
&= r(\mathbf{z})+O\delta\mathbf{z}-\Big(D(\mathbf{d}_1,\dots,\mathbf{d}_{m_*})+E(\mathbf{z}_1,\dots,\mathbf{z}_{m_*})\Big)\delta\Gamma \\
&\quad -\delta E(\mathbf{z}_1,\dots,\mathbf{z}_{m_*})\Gamma, \qquad (8.5)
\end{aligned}$$

where

$$\delta E(\mathbf{z}_1,\dots,\mathbf{z}_{m_*}) = \begin{bmatrix} \delta E_1(\mathbf{z}_1) & & & & \\ & \delta E_2(\mathbf{z}_2) & & & \\ & & \ddots & & \\ & & & \delta E_{m_*-1}(\mathbf{z}_{m_*-1}) & \\ & & & & \delta E_{m_*}(\mathbf{z}_{m_*}) \end{bmatrix}.$$

There exist matrices $Z_i(\Gamma_i)\in\mathbb{R}^{(m_{i-1}+1)\times(m_i+1)}$, $i=1,\dots,m_*$, such that

$$E_i(\mathbf{z}_i)\Gamma_i = Z_i(\Gamma_i)\mathbf{z}_i, \qquad i=1,\dots,m_*,$$

and thus

$$\delta E_i(\mathbf{z}_i)\Gamma_i = Z_i(\Gamma_i)\delta\mathbf{z}_i, \qquad i=1,\dots,m_*,$$

from which it follows that

$$\begin{aligned}
\delta E(\mathbf{z}_1,\ldots,\mathbf{z}_{m_*})\Gamma &= \begin{bmatrix} Z_1(\Gamma_1) & & & & \\ & Z_2(\Gamma_2) & & & \\ & & \ddots & & \\ & & & Z_{m_*-1}(\Gamma_{m_*-1}) & \\ & & & & Z_{m_*}(\Gamma_{m_*}) \end{bmatrix} \begin{bmatrix} \delta\mathbf{z}_1 \\ \delta\mathbf{z}_2 \\ \vdots \\ \delta\mathbf{z}_{m_*-1} \\ \delta\mathbf{z}_{m_*} \end{bmatrix} \\
&= \begin{bmatrix} 0 & Z_1(\Gamma_1) & & & & \\ 0 & & Z_2(\Gamma_2) & & & \\ \vdots & & & \ddots & & \\ 0 & & & & Z_{m_*-1}(\Gamma_{m_*-1}) & \\ 0 & & & & & Z_{m_*}(\Gamma_{m_*}) \end{bmatrix} \delta\mathbf{z} \\
&= Z(\Gamma_1,\ldots,\Gamma_{m_*})\delta\mathbf{z}, \qquad (8.6)
\end{aligned}$$

where $Z = Z(\Gamma_1,\ldots,\Gamma_{m_*}) \in \mathbb{R}^{M\times M_1}$ is equal to

$$Z(\Gamma_1,\ldots,\Gamma_{m_*}) = \begin{bmatrix} 0 & Z_1(\Gamma_1) & & & & \\ 0 & & Z_2(\Gamma_2) & & & \\ \vdots & & & \ddots & & \\ 0 & & & & Z_{m_*-1}(\Gamma_{m_*-1}) & \\ 0 & & & & & Z_{m_*}(\Gamma_{m_*}) \end{bmatrix}.$$

The substitution of (8.6) into (8.5) yields

$$r(\mathbf{z}+\delta\mathbf{z}) = r(\mathbf{z}) - (D+E)\delta\Gamma - (Z-O)\delta\mathbf{z},$$

and thus the Newton-Raphson method requires the iterative solution of

$$\begin{bmatrix} (D+E) & (Z-O) \end{bmatrix} \begin{bmatrix} \delta\Gamma \\ \delta\mathbf{z} \end{bmatrix} = r,$$

which is an under-determined equation, where $r = r(\mathbf{z})$ and

$$\begin{bmatrix} (D+E) & (Z-O) \end{bmatrix} \in \mathbb{R}^{M \times (N+M_1)}.$$

If $\Gamma^{(0)}$ and $\mathbf{z}^{(0)} = 0$ are the initial values of Γ and $\mathbf{z}$, respectively, in the Newton-Raphson method, then the $(j+1)$th iteration requires the minimisation of

$$\begin{aligned} \left\| \begin{matrix} \Gamma^{(j+1)} - \Gamma^{(0)} \\ \mathbf{z}^{(j+1)} \end{matrix} \right\| &= \left\| \begin{matrix} \Gamma^{(j)} + \delta\Gamma^{(j)} - \Gamma^{(0)} \\ \mathbf{z}^{(j)} + \delta\mathbf{z}^{(j)} \end{matrix} \right\| \\ &= \left\| \begin{bmatrix} \delta\Gamma^{(j)} \\ \delta\mathbf{z}^{(j)} \end{bmatrix} - \begin{bmatrix} -(\Gamma^{(j)} - \Gamma^{(0)}) \\ -\mathbf{z}^{(j)} \end{bmatrix} \right\|, \end{aligned}$$

subject to

$$\begin{bmatrix} (D+E) & (Z-O) \end{bmatrix}^{(j)} \begin{bmatrix} \delta\Gamma^{(j)} \\ \delta\mathbf{z}^{(j)} \end{bmatrix} = r^{(j)},$$

where the initial value of Γ is calculated from (8.2),

$$\Gamma^{(0)} = \begin{bmatrix} D_1(\mathbf{d}_1) & & & & \\ & D_2(\mathbf{d}_2) & & & \\ & & \ddots & & \\ & & & D_{m_*-1}(\mathbf{d}_{m_*-1}) & \\ & & & & D_{m_*}(\mathbf{d}_{m_*}) \end{bmatrix}^{\dagger} \begin{bmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_{m_*-2} \\ \mathbf{d}_{m_*-1} \end{bmatrix}, \qquad (8.7)$$

and $X^{\dagger} = (X^T X)^{-1} X^T$. The initial value of the residual is therefore

$$r^{(0)} = r^{(0)}(\mathbf{z}) = \mathbf{d} - D(\mathbf{d}_1, \ldots, \mathbf{d}_{m_*})\Gamma^{(0)}. \qquad (8.8)$$

This is an LSE problem

$$\min_y \|Sy - s\| \qquad \text{subject to} \quad Ty = t,$$

where

$$S = I_{N+M_1}, \qquad T = \begin{bmatrix} (D+E) & (Z-O) \end{bmatrix}^{(j)} \in \mathbb{R}^{M\times(N+M_1)},$$

$$y = \begin{bmatrix} \delta\Gamma^{(j)} \\ \delta\mathbf{z}^{(j)} \end{bmatrix} \in \mathbb{R}^{N+M_1}, \qquad s = \begin{bmatrix} -(\Gamma^{(j)} - \Gamma^{(0)}) \\ -\mathbf{z}^{(j)} \end{bmatrix} \in \mathbb{R}^{N+M_1},$$

and $t = r^{(j)} \in \mathbb{R}^M$.

It is known from Section 7.1 that the LSE problem can be solved by the QR decomposition, which is shown in Algorithm 8.1.

Algorithm 8.1: Deconvolution using the QR decomposition

```
Input  The $m_* + 1$ polynomials $d_i(x), i = 0, \ldots, m_*$.
Output  The $m_*$ polynomials $\tau_i(x), i = 1, \ldots, m_*$.
Begin

1. Set $\mathbf{z}^{(0)} = 0$ and calculate $\Gamma^{(0)}$ and $r^{(0)}$ from (8.7) and (8.8).

2. Set $s = 0$ and $t = r^{(0)}$, and initialise the matrices $S$ and $T$.

3. Iteration = 0.    % The counter for the number of iterations

   Repeat    % Use QR to solve the LSE problem at each iteration

   (a) Iteration = Iteration + 1.

   (b) Compute the QR decomposition of $T^T$ from (7.38) in order to obtain $Q$
       and $R_1$,

   (c) Set $v = R_1^{-T}t$.
```

```
(d) Partition $SQ$ as
        $$SQ = \begin{bmatrix} S_1 & S_2 \end{bmatrix}$$
    where $S_1 \in \mathbb{R}^{(N+M_1)\times M}$ and $S_2 \in \mathbb{R}^{(N+M_1)\times(N+M_1-M)}$.
(e) Compute $\nu = S_2^{\dagger}(s - S_1 v)$.
(f) Compute the solution
        $$y = Q \begin{bmatrix} v \\ \nu \end{bmatrix}.$$
(g) Set $\Gamma := \Gamma + \delta\Gamma$ and $\mathbf{z}_k := \mathbf{z}_k + \delta\mathbf{z}_k$.
(h) Compute the residual $r$, which is defined in (8.4), and update $T, s$ and $t = r$.
(b.11) Calculate $e = \mathbf{d} + O\mathbf{z}$.
Until $\frac{\|r\|}{\|e\|} \leq 10^{-16}$ OR Iteration $> 50$.
End
```

Since the deconvolution problem has been solved by a structure preserving matrix method, the computation of m_* deconvolutions can yield a sequence of polynomial equations with simple roots only, and more details are shown in Section 3.2. Moreover, the **roots** function in MATLAB is used to calculate the simple roots of these polynomials initially, after which the method of non-linear least squares is used to improve their estimates, and this is considered in the next section.

8.2 Non-linear least squares for multiple roots

This section describes the method of non-linear least squares (NLLS) and its application to the refinement of the roots of a polynomial. The Newton and Gauss-Newton methods are considered and compared for the theory of the method of NLLS, where it is assumed that the multiplicity of each root is known and initial estimates of the roots are given.

Consider the problem

$$\min_{x\in\mathbb{R}^n} h(x) = \frac{1}{2}r^T r = \frac{1}{2}\sum_{i=1}^{m} r_i(x)^2 \tag{8.9}$$

where $r = r(x) \in \mathbb{R}^m, x = \{x_i\} \in \mathbb{R}^n, n \leq m$ and each residual $r_i = r_i(x)$ is non-linear. It follows that

$$\frac{\partial h}{\partial x_j} = \sum_{i=1}^{m} r_i \frac{\partial r_i}{\partial x_j},$$

and thus at a stationary point

$$r^T J = \begin{bmatrix} r_1 & r_2 & \cdots & r_m \end{bmatrix} \begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} & \cdots & \frac{\partial r_1}{\partial x_n} \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} & \cdots & \frac{\partial r_2}{\partial x_n} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r_m}{\partial x_1} & \frac{\partial r_m}{\partial x_2} & \cdots & \frac{\partial r_m}{\partial x_n} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix} \tag{8.10}$$

where $J = J(x) = \nabla h \in \mathbb{R}^{m\times n}$ is the Jacobian matrix.

The second derivative of $h(x)$ is

$$\begin{aligned}\frac{\partial^2 h}{\partial x_l \partial x_j} &= \frac{\partial}{\partial x_l}\left\{\sum_{i=1}^{m} r_i \frac{\partial r_i}{\partial x_j}\right\} \\ &= \sum_{i=1}^{m}\left(\frac{\partial r_i}{\partial x_l}\frac{\partial r_i}{\partial x_j}\right) + \sum_{i=1}^{m} r_i \frac{\partial^2 r_i}{\partial x_j \partial x_l} \\ &= \sum_{i=1}^{m}(J_{li}^T J_{ij}) + \sum_{i=1}^{m} r_i \frac{\partial^2 r_i}{\partial x_j \partial x_l} \\ &= \left(J(x)^T J(x)\right)_{lj} + r^T \begin{bmatrix} \frac{\partial^2 r_1}{\partial x_j \partial x_l} \\ \vdots \\ \frac{\partial^2 r_m}{\partial x_j \partial x_l} \end{bmatrix}.\end{aligned}$$

If the Hessian matrices $G_i(x), i = 1, \ldots, m$, are defined as

$$G_i(x) = \nabla^2 r_i(x) \in \mathbb{R}^{n \times n}, \qquad G_i(x)_{jl} = \frac{\partial^2 r_i}{\partial x_j \partial x_l},$$

then

$$\nabla^2 h(x) = J(x)^T J(x) + Q(x), \qquad Q(x) = \sum_{i=1}^{m} r_i(x) G_i(x), \tag{8.11}$$

where $G_i(x) = G_i(x)^T$. The formulae for the Jacobian matrix $J(x)$ and the Hessian matrices $G_i(x), i = 1, \ldots, m$, enable Newton's method for the minimisation of $h(x)$ to be developed. Specifically, consider a quadratic Taylor expression of $h(x)$ about $x = x_k$,

$$\begin{aligned}h(x_k + \Delta x_k) &= h(x_k) + \Delta x_k^T J(x_k)^T r(x_k) \\ &\quad + \frac{1}{2}\Delta x_k^T \left(J(x_k)^T J(x_k) + Q(x_k)\right)\Delta x_k,\end{aligned} \tag{8.12}$$

which achieves its minimum values when

$$\left(J_k^T J_k + Q_k\right)\Delta x_k = -J_k^T r_k, \tag{8.13}$$

where $J_k = J(x_k), Q_k = Q(x_k), r_k = r(x_k)$ and $J_k^T J_k + Q_k \in \mathbb{R}^{n \times n}$. The vector Δx_k that satisfies this equation is called the Newton direction, and it leads to the Newton iteration,

$$x_{k+1} = x_k + \Delta x_k = x_k - \left(J_k^T J_k + Q_k\right)^{-1} J_k^T r_k. \tag{8.14}$$

If $J_k^T J_k + Q_k$ is positive definite, the initial estimate x_0 is near the solution, and the quadratic model (8.12) is accurate, then the iteration (8.14) converges quadratically [9], page 384.

It cannot, however, be guaranteed that $J_k^T J_k + Q_k$ is positive definite, and thus the quadratic model (8.12) may not have a minimum, and it may not have a stationary point. If $J_k^T J_k + Q_k$ is singular, a stationary point exists only if $J_k^T r_k$ lies in the column space of $J_k^T J_k + Q_k$.

The Gauss-Newton iteration is derived from the Newton iteration (8.14) by neglecting the matrix Q_k, that is, the second derivatives of r_k, and thus this iteration is

$$x_{k+1} = x_k + \Delta x_k = x_k - \left(J_k^T J_k\right)^{-1} J_k^T r_k. \tag{8.15}$$

The iteration (8.15) is better behaved than the iteration (8.14) because $J_k^T J_k$ is, at least, positive semi-definite, but Q_k may or may not be positive definite. It will be assumed that the rank of J_k is equal to n, that is, J_k has full column rank, such that the matrix inverse in (8.15) exists. It has been demonstrated in [72] that if the roots $x_j, j = 1, \ldots, n$, are distinct, this assumption is satisfied.

It follows from (8.11) that the approximation $J_k^T J_k + Q_k \approx J_k^T J_k$ assumes that

$$\|Q(x_k)\| = \left\| \sum_{i=1}^{m} r_i(x_k) G_i(x_k) \right\| \leq \sum_{i=1}^{m} \|r_i(x_k)\| \, \|G_i(x_k)\|,$$

is small, that is, the residuals are small and/or they are only weakly non-linear. In this circumstance, the iterations (8.14) and (8.15) behave similarly, and convergence of the Gauss-Newton method is almost quadratic. If, however, the residuals are large, then the convergence of the Gauss-Newton iteration may be substantially inferior with respect to the convergence of the Newton iteration. The application of the method of NLLS to the calculation of the values of the roots of a polynomial is now considered.

8.2.1 Calculating the values of the roots

In [72], it is shown that the method of NLLS is used for the calculation of the values of the roots of a polynomial, and the equation that is solved by the method NLLS is based on the pejorative manifold of the polynomial. The pejorative manifold of a polynomial is defined by the multiplicity structure of its roots, which has been described in Section 2.3. It also presents that the importance of the pejorative manifold arises because multiple roots are usually assumed to be ill-conditioned, but they are insensitive to perturbations that maintain the polynomial on its pejorative manifold. In particular, the roots of a polynomial are ill-conditioned when random (unstructured) perturbations are applied to its coefficients, in which case the perturbed polynomial does not lie on the pejorative manifold of its unperturbed form, but structured perturbations are required to keep a polynomial on its pejorative manifold. This property of pejorative manifolds forms the theoretical basis of the algorithm in [72] for the computation of the roots of a polynomial, and thus the work using the method of NLLS follows closely the work of Zeng [72].

Consider the polynomial $f(x)$ of degree m with coefficients $f_i \in \mathbb{R}, i = 0, \ldots, m$,

$$f(x) = \sum_{i=0}^{m} f_i x^{m-i} = f_0 x^m + f_1 x^{m-1} + \cdots + f_{m-1} x + f_m,$$

where

$$f(x) \sim \mathbf{a} = \left[\begin{array}{ccccc} a_1 & a_2 & \cdots & a_{m-1} & a_m \end{array} \right]^T = \left[\begin{array}{ccccc} \frac{f_1}{f_0} & \frac{f_2}{f_0} & \cdots & \frac{f_{m-1}}{f_0} & \frac{f_m}{f_0} \end{array} \right]^T \in \mathbb{R}^m$$

and $\sim$ denotes the correspondence between the polynomial $f = f(x)$ and $\mathbf{a}$, the vector of its normalised coefficients. If the distinct roots of $f(x)$ are $x_j \in \mathbb{R}, j = 1, \ldots, n$, and the root x_j has multiplicity m_j, then

$$\frac{f(x)}{f_0} = \prod_{j=1}^{n}(x - x_j)^{m_j} = x^m + \sum_{i=1}^{m} g_i(x_1, \ldots, x_n)x^{m-i}, \tag{8.16}$$

where

$$\sum_{j=1}^{n} m_j = m, \qquad g_i(x_1, \ldots, x_n) = \frac{f_i}{f_0} = a_i, \quad i = 1, \ldots, m. \tag{8.17}$$

Equation (8.17) leads to the equation $G(\mathbf{x}) = \mathbf{a}$, where $G(\mathbf{x}) \in \mathbb{R}^m$,

$$\begin{bmatrix} g_1(x_1, \ldots, x_n) \\ \vdots \\ g_m(x_1, \ldots, x_n) \end{bmatrix} = \begin{bmatrix} a_1 \\ \vdots \\ a_m \end{bmatrix} \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n. \tag{8.18}$$

It is shown in Section 2.3 that given a multiplicity structure $\mathbf{m} = [m_1, m_2, \ldots, m_n]$, the pejorative manifold $\mathcal{M}$ of a monic polynomial $f(x)$ of degree m with n distinct roots for $\mathbf{m}$ is defined from (2.9) and given by

$$\mathcal{M} = \{ \ G(\mathbf{x}) = \mathbf{a} \ \mid \ \mathbf{a} \in \mathbb{R}^m, \ \ \mathbf{x} \in \mathbb{R}^n, \ \ x_i \neq x_j, \ \ i \neq j \ \}.$$

It follows from the theory above that the distinct roots $x_j, j = 1, \ldots, n$, of $f(x)$ are the solution of the non-linear equation (8.18). This is a set of m equations in n unknowns, where $m > n$ if $f(x)$ contains a multiple root, and $m = n$ if and only if all the roots of $f(x)$ are simple. These equations are solved by the method of NLLS, and thus it is necessary to determine the vector $\mathbf{x}$ that solves the minimisation problem

$$\min_{\mathbf{x} \in \mathbb{R}^n} \frac{1}{2} \left\| G(\mathbf{x}) - \mathbf{a} \right\|_2^2 = \min_{\mathbf{x} \in \mathbb{R}^n} \left\{ \frac{1}{2} \sum_{i=1}^{m} (g_i(\mathbf{x}) - a_i)^2 \right\}.$$

Comparison of this function with $h(x)$ in (8.9) shows that

$$r_i(\mathbf{x}) = g_i(\mathbf{x}) - a_i, \qquad i = 1, \ldots, m, \quad \mathbf{x} \in \mathbb{R}^n,$$

and the elements of the Jacobian matrix $J = \{J_{ij}\}_{i,j=1}^{m,n}$ of the functions $r_i(\mathbf{x})$ are

$$J_{ij} = \frac{\partial r_i}{\partial x_j} = \frac{\partial g_i(\mathbf{x})}{\partial x_j}. \tag{8.19}$$

This stationarity condition (8.10) becomes

$$r^T J = [G(\mathbf{x}) - \mathbf{a}]^T J = 0, \tag{8.20}$$

which shows that the vector $G(\mathbf{x}) - \mathbf{a}$ is orthogonal to the tangent plane of the manifold $\mathcal{M}_* = \{w = G(\mathbf{x}) | \mathbf{x} \in \mathbb{R}^n\}$ at $w_* = G(\mathbf{x}_*)$ where $\mathbf{x} = \mathbf{x}_*$ is a solution of (8.20).

The coefficients of the normalised polynomial (8.16) can be obtained by repeated convolution, and this enables the expressions for $g_i(\mathbf{x}), i = 1, \ldots, m$, to be derived. Algorithm 8.2 shows pseudo-code for the calculation of the entries $g_i(\mathbf{x}), i = 1, \ldots, m$, of $G(\mathbf{x})$.

Algorithm 8.2: The calculation of $G(\mathbf{x})$

Input The integers m and n, the roots $x_j, j = 1, \ldots, n$, and the multiplicity m_j of x_j.

Output The entries $g_i(\mathbf{x}), i = 1, \ldots, m$, of the vector $G(\mathbf{x})$.

```
Begin
  $\mathrm{s} = [1]$
  for $j = 1, 2, \ldots, n$
      for $l = 1, 2, \ldots, m_j$
          $\mathrm{s} = \mathrm{conv}(\mathrm{s}, (1, -x_j))$    % s is of length $m + 1$.
```

```
        end $l$

    end $j$

    for $i = 1, 2, \ldots, m$

        $g(i) = s(i+1)$    % $g(i) = g_i(\mathbf{x})$

    end $i$

End
```

Since the elements of the Jacobian matrix J are defined in (8.19), the jth column of J is given by the vector

$$J_j = \begin{bmatrix} \frac{\partial g_1(\mathbf{x})}{\partial x_j} & \frac{\partial g_2(\mathbf{x})}{\partial x_j} & \cdots & \frac{\partial g_{m-1}(\mathbf{x})}{\partial x_j} & \frac{\partial g_m(\mathbf{x})}{\partial x_j} \end{bmatrix}^T .$$

and consider the polynomials $q_j(x), j = 1, \ldots, n$, of the degree $m-1$, such that the coefficients of $q_j(x)$ are formed from the entries of J_j,

$$\begin{aligned}
q_j(x) &= \frac{\partial g_1(\mathbf{x})}{\partial x_j} x^{m-1} + \frac{\partial g_2(\mathbf{x})}{\partial x_j} x^{m-2} + \cdots + \frac{\partial g_m(\mathbf{x})}{\partial x_j} \\
&= \frac{\partial}{\partial x_j} \left[x^m + g_1(\mathbf{x}) x^{m-1} + \cdots + g_m(\mathbf{x}) \right] \\
&= \frac{\partial}{\partial x_j} \left[(x-x_1)^{m_1} (x-x_2)^{m_2} \cdots (x-x_n)^{m_n} \right] \quad \text{from(8.16)} \\
&= -m_j (x-x_j)^{m_j-1} \left[\prod_{l \neq j} (x-x_l)^{m_l} \right] \\
&= -m_j \left[\prod_{l=1}^{n} (x-x_l)^{m_l-1} \right] \prod_{l \neq j} (x-x_l). \qquad (8.21)
\end{aligned}$$

The expression (8.21) is therefore used in the pseudo-code in Algorithm 8.3 for the calculation of the elements of J.

Algorithm 8.3: The calculation of $J(\mathbf{x})$

Input The integers m and n, the roots $x_j, j = 1, \dots, n$, and the multiplicity m_j of x_j.

Output The Jacobian matrix $J = J(\mathbf{x})$.

Begin

```
  $\mathbf{u} = [1]$
  for $j = 1, 2, \dots, n$
      for $l = 1, 2, \dots, m_j - 1$
          $\mathbf{u} = \text{conv}(\mathbf{u}, (1, -x_j))$
      end $l$
  end $j$
  for $j = 1, 2, \dots, n$
      $\mathbf{v} = -m_j \mathbf{u}$
      for $l = 1, 2, \dots, n, l \neq j$
          $\mathbf{v} = \text{conv}(\mathbf{v}, (1, -x_j))$
      end $l$
      $J(:, j) = \mathbf{v}$        % $\mathbf{v}$ is equal to the $j$th column of $J$
  end $j$
```

End

Algorithm 8.4 is a combination of Algorithm 8.2 and 8.3 for the least squares solution $\mathbf{x}_*$ of (8.18).

Algorithm 8.4: The calculation of the roots $\mathbf{x}_*$

Input The vector $\mathbf{x}_0$ of the initial estimates of the least squares solution $\mathbf{x}_*$ of (8.18), the multiplicity m_j of each distinct root $x_{0,i}, i = 1, \ldots, n$, the vector $\mathbf{a}$ of normalised coefficients, the integers m and n, and the error tolerance ε_r.

Output The least square solution $\mathbf{x}_*$ of (8.18).

Begin

```
1. Set $k = 0$.

2. Calculate the vector $G(\mathbf{x}_0)$ using Algorithm 8.2, and the residuals

       $r_i(\mathbf{x}_0) = g_i(\mathbf{x}_0) - a_i, \qquad i = 1, \ldots, m.$

3. Repeat

   (a) Calculate the Jacobian matrix $J_k = J(\mathbf{x}_k)$ using Algorithm 8.3.

   (b) Calculate $\Delta\mathbf{x}_k = -(J_k^T J_k)^{-1} J_k^T r(\mathbf{x}_k)$.

   (c) Calculate $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k$.

   (d) Calculate the vector $G(\mathbf{x}_{k+1})$ using Algorithm 8.2, and the elements of
       residual $r(\mathbf{x}_{k+1})$

       $r_i(\mathbf{x}_{k+1}) = g_i(\mathbf{x}_{k+1}) - a_i, \qquad i = 1, \ldots, m.$
```

```
        % $\mathbf{x}_{k+1}$ is the $(k+1)$th iteration of the vector $\mathbf{x}$. .
      (e) Calculate the error
              $$\delta e = \frac{\|r(\mathbf{x}_{k+1}) - r(\mathbf{x}_k)\|}{\|r(\mathbf{x}_k)\|}.$$
      (f) Set $k := k+1$
      Until $\delta e \leq \varepsilon_r$      % local minimum attained
  4. $\mathbf{x}_* := \mathbf{x}_k$.
End
```

The implementation of the designed polynomial root solver is now detailed in next section.

8.3 Overview of implementation of a polynomial root solver

The polynomial root solver was introduced in Section 3.2, and the stages in the algorithm were discussed in

(a) Chapter 4: preprocessing operations,

(b) Chapters 5 and 6: the calculation of the degree of an approximate GCD,

(c) Chapter 7: the calculation of the coefficients of an approximate GCD,

(d) Chapter 8: the calculation of the roots and their multiplicities.

If $d_0(x)$ is a given polynomial in the presence of noise, then a sequence of approximate GCD computations

$$d_i(x) = \text{GCD}\left(d_{i-1}(x), d_{i-1}^{(1)}(x)\right), \quad i = 1, \ldots, m_*,$$

are completed using Algorithm 7.1. It was stated in Section 3.2 that a sequence of polynomials $\tau_i(x), i = 1, \ldots, m_*$ are equal to the deconvolution of $d_{i-1}(x)$ and $d_i(x)$. Similarly, a sequence of polynomials $\chi_i(x), i = 1, \ldots, m_* - 1$, are equal to the deconvolution of $\tau_i(x)$ and $\tau_{i+1}(x)$, and $\chi_{m_*}(x) = \tau_{m_*}(x)$, in which case $\chi_i(x)$ contains either only simple roots, or $\chi_i(x)$ has no roots and it is equal to a constant. It is therefore clear that $\tau_i(x)$ and $\chi_i(x)$ can be calculated by Algorithm 8.1. The **roots** function in MATLAB is then used to calculate the simple roots of $\chi_i(x)$, whose roots are refined by Algorithm 8.4. These improved estimates of the roots of $\chi_i(x)$ are equal to the roots of $d_0(x)$ with multiplicities i.

Algorithm 8.5 contains pseudo-code for the implementation of this root solver, and is a combination of Algorithms 3.1, 6.2, 7.1, 8.1 and 8.4 for the calculation of the roots of a polynomial.

Algorithm 8.5: A robust polynomial root solver

```
Input An inexact polynomial $d_0(x)$.
Output The roots of $d_0(x)$.
Begin

1. Set $j = 0$.

2. While degree $d_j > 0$ do
```

```
        (a) Set $j = j + 1$.
        (b) Calculate the degree of an approximate GCD of $d_{j-1}$ and its derivative
            $d_{j-1}^{(1)}$ using Algorithm 6.2.
        (c) Calculate an approximate GCD of $d_{j-1}$ and $d_{j-1}^{(1)}$ using Algorithm 7.1,
                $d_j = \text{GCD}\left(d_{j-1}, d_{j-1}^{(1)}\right)$.
    End While
    3. Calculate $\tau_i = \frac{d_{i-1}}{d_i}, i = 1, \ldots, j$, using Algorithm 8.1.
    4. Calculate $\chi_i = \frac{\tau_i}{\tau_{i+1}}, i = 1, \ldots, j-1$, using Algorithm 8.1.
    5. Set $\chi_j = \tau_j$.
    6. Calculate the roots $\mathbf{x}_i$ of $\chi_i, i = 1, \ldots, j$, using the roots function in MATLAB.
       % They are of multiplicity $i$.
    7. Calculate improved estimates of the roots of the polynomial $d_0(x)$ using Algo-
       rithm 8.4, with initial estimates of the roots $\mathbf{x}_i$ and their multiplicity $i$.
End
```

Example 8.1. Consider an exact polynomial $\hat{d}_0(x)$ of degree $m = 31$, whose roots and multiplicities are specified in Table 8.1. Noise with componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^8$ was applied to $\hat{d}_0(x)$, thereby yielding $d_0(x)$.

Algorithm 8.5 can be used to calculate the roots of the inexact polynomial $d_0(x)$, and the result is shown in Table 8.2. The 1st column of Table 8.2 shows the computed multiplicities through a sequence of the approximate GCD computations, the 2nd and

Root of $\hat{d}_0(x)$	Multiplicity
-1.7223e+000	3
3.0024e+000	4
9.4967e+000	4
-8.4807e+000	9
1.7404e+000	11

Table 8.1: The roots and multiplicities of $\hat{d}_0(x)$ for Example 8.1.

3rd columns show initial estimates of the roots computed from Algorithm 8.1 and their relative errors respectively, and the 4th and 5th columns show improved estimates of the roots computed from Algorithm 8.4 and their relative errors respectively.

Computed Multiplicity	Initial Root	Root Error	Improved Root	Root Error
3	-1.7222e+000	3.4620e-005	-1.7223e+000	1.8178e-009
4	3.0023e+000	4.1086e-005	3.0024e+000	1.3843e-009
4	9.4965e+000	2.0573e-005	9.4967e+000	4.2115e-009
9	-8.4806e+000	9.4139e-006	-8.4807e+000	7.8752e-010
11	1.7404e+000	3.7953e-006	1.7404e+000	7.3028e-012

Table 8.2: Solving an inexact polynomial equation for Example 8.1.

It is clear that the computed multiplicities of the roots are equal to the multiplicities of the exact roots. Algorithm 8.1 returns excellent initial estimates of the roots, such that Algorithm 8.4 returns a perfect answer because the relative errors of the roots are smaller than the noise level $\varepsilon_c = 10^{-8}$. □

Example 8.2. Consider an exact polynomial $\hat{d}_0(x)$ of degree $m = 36$, whose roots and multiplicities are specified in Table 8.3. Noise with the componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^8$ was applied to $\hat{d}_0(x)$, thereby yielding $d_0(x)$.

Root of $\hat{d}_0(x)$	Multiplicity
-1.3708e+000	1
-3.2431e+000	2
4.4145e+000	3
-9.7269e+000	4
-2.5188e+000	5
8.4537e+000	6
9.2960e-001	7
-5.2230e-001	8

Table 8.3: The roots and multiplicities of $\hat{d}_0(x)$ for Example 8.2.

Algorithm 8.5 can be used to calculate the roots of the inexact polynomial $d_0(x)$, and the result is shown in Table 8.4. The 1st column of Table 8.4 shows the computed multiplicities through a sequence of the approximate GCD computations, the 2nd and 3rd columns show initial estimates of the roots computed from Algorithm 8.1 and their relative errors respectively, and the 4th and 5th columns show improved estimates of the roots computed from Algorithm 8.4 and their relative errors respectively.

Computed Multiplicity	Initial Root	Root Error	Improved Root	Root Error
1	-1.3689e+000	1.3519e-003	-1.3708e+000	5.4588e-008
2	-3.2486e+000	1.7021e-003	-3.2431e+000	1.3764e-007
3	4.4020e+000	2.8211e-003	4.4145e+000	1.2455e-007
4	-9.7769e+000	5.1389e-003	-9.7269e+000	1.4975e-007
5	-2.5176e+000	4.8811e-004	-2.5188e+000	8.7257e-009
6	8.5351e+000	9.6302e-003	8.4537e+000	1.1569e-007
7	9.2960e-001	4.5549e-006	9.2960e-001	7.8113e-010
8	-5.2212e-001	3.3691e-004	-5.2230e-001	7.5864e-010

Table 8.4: Solving an inexact polynomial equation for Example 8.2.

It is seen that the computed multiplicities of the roots are equal to the multiplicities of exact roots. Although Algorithm 8.1 returns some acceptable initial estimates of the roots, Algorithm 8.4 still performs very well because it improves the results significantly. $\square$

8.4 Summary

This chapter has considered the use of the method of STLN applied to a structured matrix for the solution of the deconvolution problem, which is one important part of implementation of the polynomial root solver. Since the relative errors of the roots that are calculated from polynomials $\chi_i(x)$ are small but not sufficient compared with noise level, an improvement for the estimates of the roots of an inexact polynomial is developed using the method of NLLS, in which case the method of NLLS solves the equation that is solved on the pejorative manifold of this polynomial. It has been shown in Examples 8.1 and 8.2 that the method of NLLS can improve the accuracy of the computed roots such that the relative errors between the exact and computed roots can attain levels that are near the noise level.

Chapter 9

Results

The implementation of a robust polynomial root solver has been considered in Chapter 8, and the success of this root solver has been shown in Examples 1.3 – 1.8 and 8.1 – 8.2 because it finds the exact roots of a noisy polynomial and their multiplicities. This is not, however, the only situation. Computational experiments showed that in some cases this root solver may find a computed solution that is the theoretically exact solution of a neighbouring polynomial equation, that is,

distance (computed polynomial to given inexact polynomial) <

distance (theoretically exact polynomial to given inexact polynomial),

and hence a schematic graph is shown in Figure 9.1. It has been shown in Section 2.1 that the backward error is based on the observation that the computed solution, which is in error, is the theoretically exact solution of a neighbouring problem, that is, a problem is 'near' the problem whose solution is desired. Thus the backward error is a measure of the distance between the problem whose solution is sought and the problem whose solution has been computed. It is therefore concluded from Figure 9.1

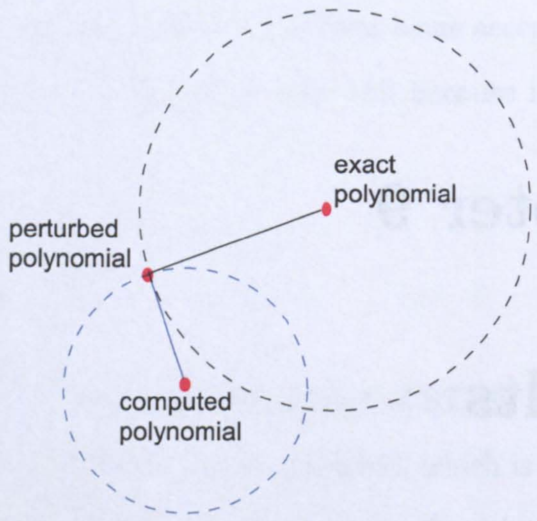


Figure 9.1: The solution of a neighbouring polynomial.

that the backward error of the computed solution is less than the error in the data, and thus the computed solution is acceptable.

This chapter considers more examples for both situations, in which case the root solver can obtain the roots of the original exact polynomial and their multiplicities, or the exact roots of a neighbouring polynomial and their multiplicities. All the results are therefore compared with some other methods, such as, Newton's method [45], Müller's method [21], Zeng's algorithm [71, 72] and the `roots` function in MATLAB. Newton's method is one of the most widely used methods of solving polynomial equations, as is Müller's method, and Zeng's algorithm is explicitly designed for the computation of multiple roots. Newton's method and Muller's method can calculate only the simple roots due to their inability to find the multiplicities of multiple roots unless special precautions are taken, while Zeng's algorithm and the `roots` function in MATLAB can compute not only the values of all multiple roots but also their multiplicities .

Zeng's algorithm uses a MATLAB package, MULTROOT,

`z=multroot(p, threshold)`,

to compute the roots and their multiplicities. If threshold is omitted such as `z=multroot(p)`, threshold $= 10^{-10}$ as default. The work in this thesis does not require the knowledge of the noise level, and thus comparison of the results that are computed by different methods requires that the noise level be omitted in MULTROOT, that is, threshold argument should not be included in MULTROOT. Different situations, however, occur:

- in the absence of noise (only roundoff error), `z=multroot(p)` returns perfect answers.
- in the presence of noise, `z=multroot(p, threshold)` returns good answers if threshold $\geq$ (signal-to-noise ratio)$^{-1}$, that is, threshold $\geq \varepsilon_c$. But if threshold $< \varepsilon_c$, the multiplicities of the roots are destroyed, and only complex conjugate simple roots are returned. It therefore follows that `z=multroot(p)` returns incorrect answers when $\varepsilon_c > 10^{-10}$, that is, the multiple roots split up into a cluster of simple roots.

Example 9.1. Consider an exact polynomial $\hat{f}(x)$ of degree $m = 27$, whose roots and multiplicities are specified in Table 9.1. Noise with componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^7$ was applied to $\hat{f}(x)$, thereby yielding $f(x)$. The designed root solver is used to compute the roots of the perturbed polynomial $f(x)$ and their multiplicities, and these results are also shown in Table 9.1.

Exact Multiplicity	Exact Root	Computed Multiplicity	Computed Root	Root Error
2	8.1031	2	8.1031e+000	2.7481e-007
8	-0.6306	8	-6.3060e-001	1.2101e-008
8	3.5078	8	3.5078e+000	3.8261e-008
9	-5.8211	9	-5.8211e+000	2.1803e-008

Table 9.1: The computed roots of an inexact polynomial for Example 9.1 using the designed root solver.

It is clear that the roots of $f(x)$ and their multiplicities are certified correctly because the relative errors of the roots fluctuate at the noise level $\varepsilon_c = 10^{-7}$ and the computed multiplicities are the same as the exact values.

Also, finding all zeros of $f(x)$ is considered by other methods, which are Newton's method, Müller's method, Zeng's algorithm and the `roots` function in MATLAB. In particular, Newton's method and Müller's method can be used to calculate the value of a multiple root, but are not sufficient to find its multiplicity. Since $f(x)$ is of degree 27, for simplicity, the roots of $f(x)$ are computed by using these methods 27 times, with different initial estimates that are uniformly distributed random variables in the range $[-10, 10]$, and the computed roots are then sorted in ascending order.

Figure 9.2 shows the solution of the inexact polynomial equation calculated by Newton's method, Müller's method, Zeng's algorithm and the `roots` function in MATLAB, separately. It is shown that Newton's method performs better than Müller's method because of smaller errors between the computed roots and exact roots in Figure 9.2(i), but both methods can not be used to compute the multiplicities of the roots. The result shown in Figure 9.2(iii) is similar to the result shown in Figure

9.2(iv) because the computed multiple roots are ill-conditioned with evidence of increasing instability as their multiplicities increase and the break up as a cluster of simple roots, that is, roundoff errors due to floating point arithmetic and errors in polynomial coefficients are sufficient to cause an incorrect and unacceptable solution.

□

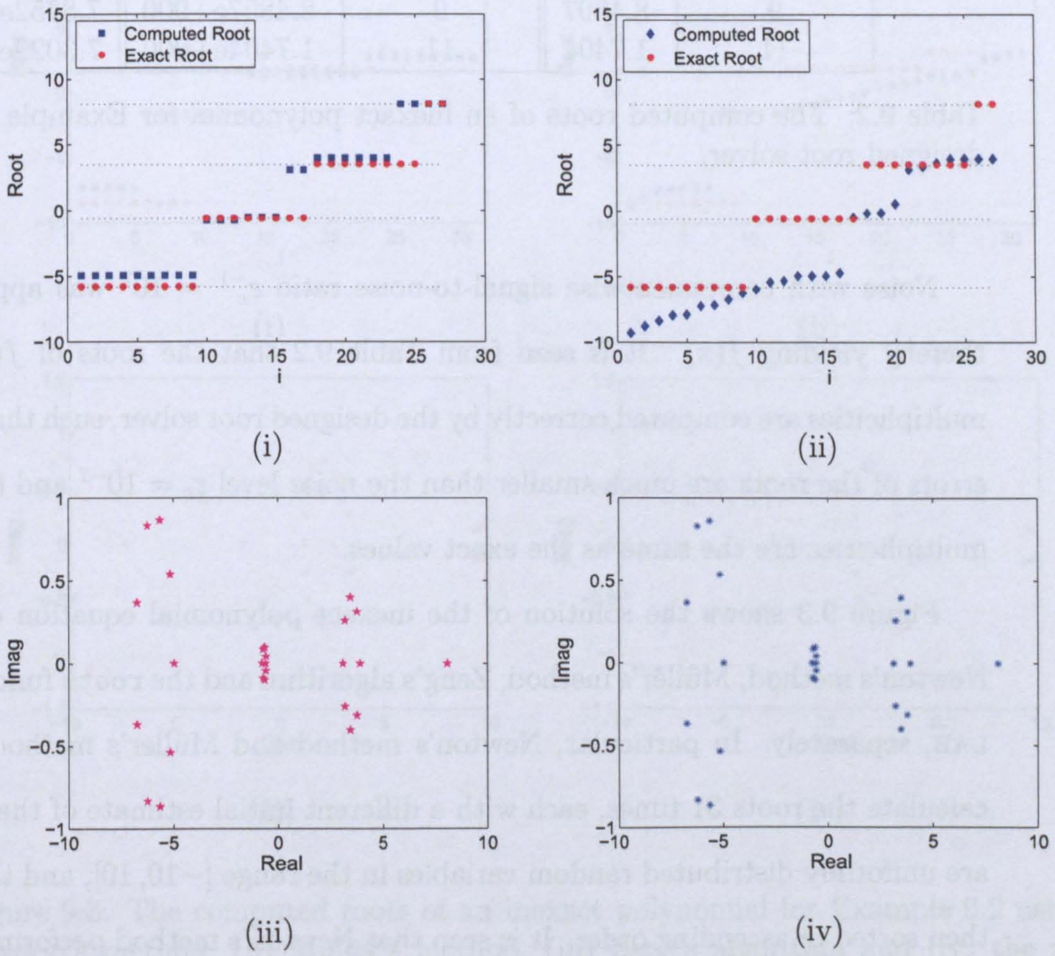


Figure 9.2: The computed roots of an inexact polynomial for Example 9.1 using (i) Newton's method, (ii) Müller's method, (iii) Zeng's algorithm and (iv) the `roots` function.

Example 9.2. Consider an exact polynomial $\hat{f}(x)$ of degree $m = 31$, whose roots and multiplicities are specified in Table 9.2.

Exact Multiplicity	Exact Root	Computed Multiplicity	Computed Root	Root Error
3	-1.7223	3	-1.7223e+000	1.8178e-008
4	3.0024	4	3.0024e+000	1.3843e-008
4	9.4967	4	9.4967e+000	4.2115e-008
9	-8.4807	9	-8.4807e+000	7.8752e-009
11	1.7404	11	1.7404e+000	7.3029e-011

Table 9.2: The computed roots of an inexact polynomial for Example 9.2 using the designed root solver.

Noise with componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^7$ was applied to $\hat{f}(x)$, thereby yielding $f(x)$. It is seen from Table 9.2 that the roots of $f(x)$ and their multiplicities are computed correctly by the designed root solver, such that the relative errors of the roots are much smaller than the noise level $\varepsilon_c = 10^{-7}$ and the computed multiplicities are the same as the exact values.

Figure 9.3 shows the solution of the inexact polynomial equation calculated by Newton's method, Müller's method, Zeng's algorithm and the `roots` function in MATLAB, separately. In particular, Newton's method and Müller's method are used to calculate the roots 31 times, each with a different initial estimate of the roots, which are uniformly distributed random variables in the range $[-10, 10]$, and the results are then sorted in ascending order. It is seen that Newton's method performs better than Müller's method because the errors between the computed roots and exact roots in Figure 9.3(i) are smaller than the errors in Figure 9.3(ii). It is clear, however, that both methods can not calculate the multiplicities of these roots. Zeng's algorithm and the `roots` function return an incorrect and unacceptable result, but yield very similar answers that are shown in Figure 9.3(iii) and Figure 9.3(iv), respectively. It is clear that the multiple roots split up into a cluster of simple roots because of roundoff errors due to floating point arithmetic and errors in polynomial coefficients. □

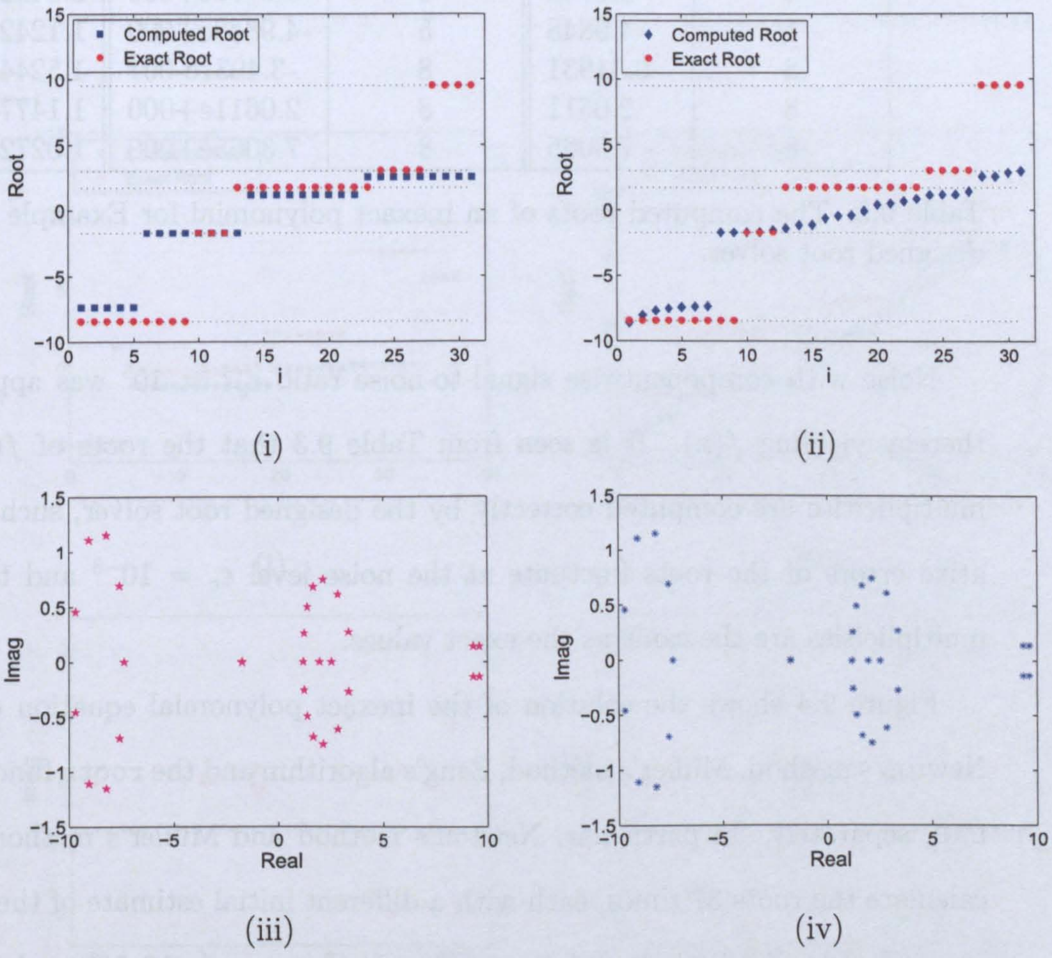


Figure 9.3: The computed roots of an inexact polynomial for Example 9.2 using (i) Newton's method, (ii) Müller's method, (iii) Zeng's algorithm and (iv) the `roots` function.

Example 9.3. Consider an exact polynomial $\hat{f}(x)$ of degree $m = 37$, whose roots and multiplicities are specified in Table 9.3.

Exact Multiplicity	Exact Root	Computed Multiplicity	Computed Root	Root Error
1	-1.2102	1	-1.2102e+000	1.2186e-008
1	0.13371	1	1.3371e-001	1.6739e-007
3	-0.099065	3	-9.9065e-002	1.9463e-007
3	9.0702	3	9.0702e+000	2.8422e-008
5	-4.9848	5	-4.9848e+000	1.1242e-009
8	-0.34931	8	-3.4931e-001	1.5244e-008
8	2.0611	8	2.0611e+000	1.1477e-009
8	7.3065	8	7.3065e+000	1.0272e-008

Table 9.3: The computed roots of an inexact polynomial for Example 9.3 using the designed root solver.

Noise with componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^8$ was applied to $\hat{f}(x)$, thereby yielding $f(x)$. It is seen from Table 9.3 that the roots of $f(x)$ and their multiplicities are computed correctly by the designed root solver, such that the relative errors of the roots fluctuate at the noise level $\varepsilon_c = 10^{-8}$ and the computed multiplicities are the same as the exact values.

Figure 9.4 shows the solution of the inexact polynomial equation calculated by Newton's method, Müller's method, Zeng's algorithm and the `roots` function in MATLAB, separately. In particular, Newton's method and Müller's method are used to calculate the roots 37 times, each with a different initial estimate of the roots, which are uniformly distributed random variables in the range $[-10, 10]$, and the computed roots are then sorted in ascending order, as shown in Figure 9.4(i) and (ii), respectively. In spite of the multiplicities of the roots, it seems that Newton's method works better than Müller's method because Newton's method can find the exact small roots when $i = 1, \ldots, 33$, and Müller's method yields inexact roots that vary with the initial estimates of the roots. Zeng's algorithm and the `roots` function return an incorrect and unacceptable result, but yield very similar answers that are shown in Figure 9.4(iii) and (iv), respectively. It is clear that all multiple roots, especially for the roots with big absolute values, split up into a cluster of simple roots because of roundoff errors due to floating point arithmetic and errors in polynomial coefficients. □

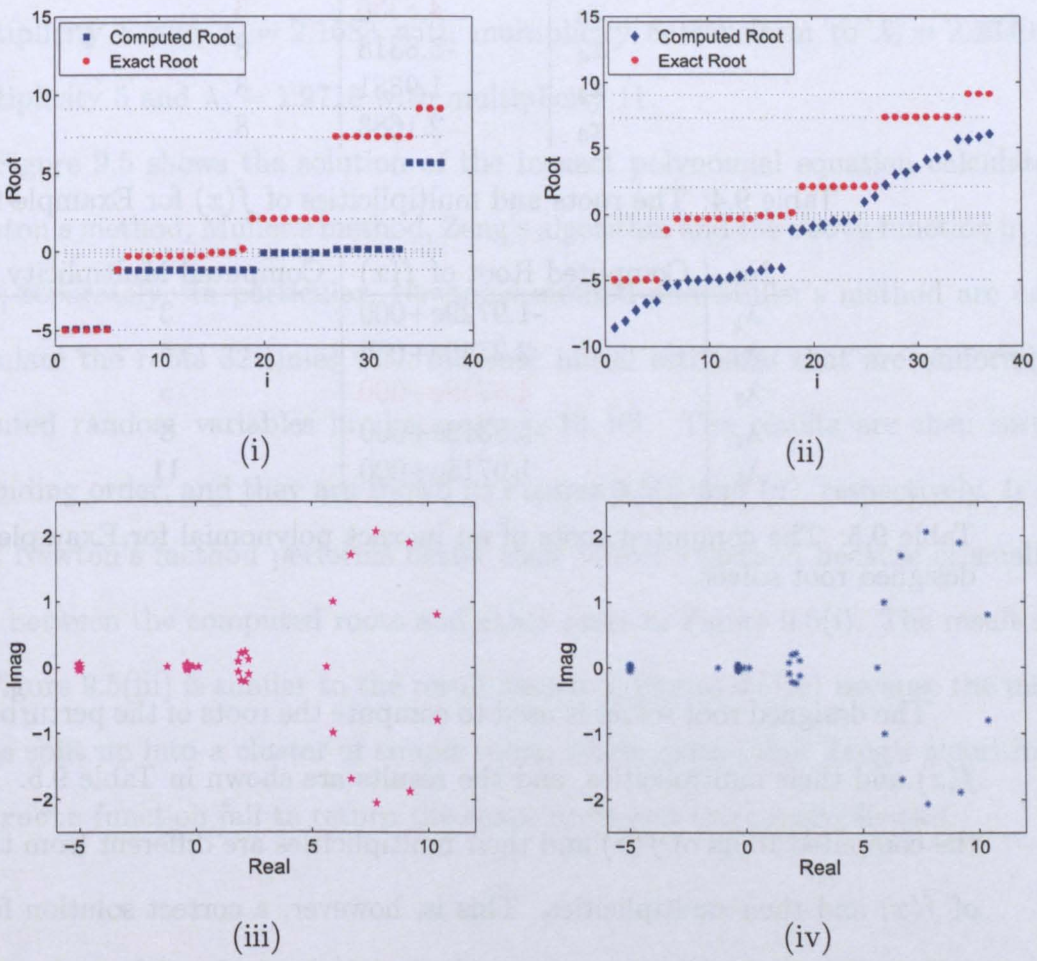


Figure 9.4: The computed roots of an inexact polynomial for Example 9.3 using (i) Newton's method, (ii) Müller's method, (iii) Zeng's algorithm and (iv) the `roots` function.

Example 9.4. Consider an exact polynomial $\hat{f}(x)$ of degree $m = 32$, whose roots and multiplicities are specified in Table 9.4. Noise with componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^8$ was applied to $\hat{f}(x)$, thereby yielding $f(x)$.

No.	Root of $\hat{f}(x)$	Multiplicity
x_1	4.9429	2
x_2	-1.9729	3
x_3	4.8336	3
x_4	-5.8318	8
x_5	1.9381	8
x_6	2.1683	8

Table 9.4: The roots and multiplicities of $\hat{f}(x)$ for Example 9.4.

No.	Computed Root of $f(x)$	Computed Multiplicity
λ_1	-1.9729e+000	3
λ_2	2.2340e+000	5
λ_3	4.8748e+000	5
λ_4	-5.8319e+000	8
λ_5	1.9718e+000	11

Table 9.5: The computed roots of an inexact polynomial for Example 9.4 using the designed root solver.

The designed root solver is used to compute the roots of the perturbed polynomial $f(x)$ and their multiplicities, and the results are shown in Table 9.5. It is seen that the computed roots of $f(x)$ and their multiplicities are different from the exact roots of $\hat{f}(x)$ and their multiplicities. This is, however, a correct solution for the inexact polynomial equation because the residual of $f(x)$ calculated from the computed roots is smaller than the residual from the exact roots [1], in which case the residuals are equal to $6.4806e + 001$ and $6.4877e + 001$, respectively. It also means that the computed

[1]Assume that the roots $x_i, i = 1, \ldots, n$, with the multiplicities m_i are the computed roots of the polynomial $f(x)$, and thus the residual of $f(x)$ is equal to $\frac{1}{\|f(x)\|} \sum_{i=1}^{n} m_i f(x_i)$.

solution is the theoretically exact solution of a neighbouring polynomial equation that is 'nearer' the noisy polynomial equation than the exact polynomial equation whose roots are desired, with reference to Figure 9.1. Furthermore, it seems that the proximity of the roots may lead to the occurrence of this situation because the pair of roots $x_1 = 4.9429$ with multiplicity 2 and $x_3 = 4.8336$ with multiplicity 3 transform to the root $\lambda_3 = 4.8748$ with multiplicity 5, and the pair of roots $x_5 = 1.9381$ with multiplicity 8 and $x_6 = 2.1683$ with multiplicity 8 transform to $\lambda_2 = 2.2340$ with multiplicity 5 and $\lambda_5 = 1.9718$ with multiplicity 11.

Figure 9.5 shows the solution of the inexact polynomial equation calculated by Newton's method, Müller's method, Zeng's algorithm and the `roots` function in MATLAB, separately. In particular, Newton's method and Müller's method are used to calculate the roots 32 times with different initial estimates that are uniformly distributed random variables in the range $[-10, 10]$. The results are then sorted in ascending order, and they are shown in Figures 9.5(i) and (ii), respectively. It seems that Newton's method performs better than Müller's method because of smaller errors between the computed roots and exact roots in Figure 9.5(i). The result shown in Figure 9.5(iii) is similar to the result shown in Figure 9.5(iv) because the multiple roots split up into a cluster of simple roots, which means that Zeng's algorithm and the `roots` function fail to return the exact roots and their multiplicities. □

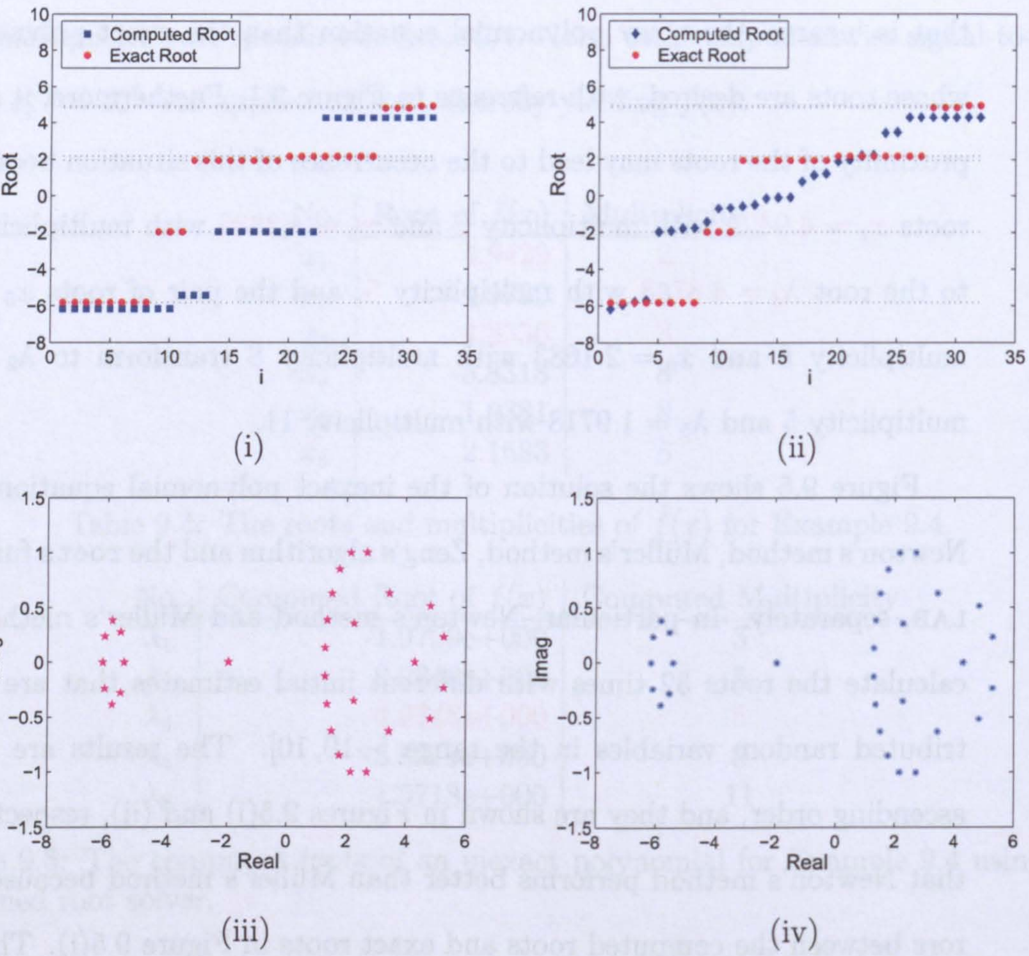


Figure 9.5: The computed roots of an inexact polynomial for Example 9.4 using (i) Newton's method, (ii) Müller's method, (iii) Zeng's algorithm and (iv) the `roots` function.

Example 9.5. Consider an exact polynomial $\hat{f}(x)$ of degree $m = 24$, whose roots and multiplicities are specified in Table 9.6.

Noise with componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^7$ was applied to $\hat{f}(x)$, thereby yielding $f(x)$. The designed root solver is used to compute the roots of the perturbed polynomial $f(x)$ and their multiplicities, and the results are shown in Table

No.	Root of $\hat{f}(x)$	Multiplicity
x_1	-7.6516	2
x_2	-4.0665	3
x_3	4.2243	5
x_4	-5.5651	6
x_5	-3.6244	8

Table 9.6: The roots and multiplicities of $\hat{f}(x)$ for Example 9.5.

No.	Computed Root of $f(x)$	Computed Multiplicity
λ_1	-7.4650e+000	2
λ_2	4.2244e+000	5
λ_3	-5.6731e+000	6
λ_4	-3.7176e+000	11

Table 9.7: The computed roots of an inexact polynomial for Example 9.5 using the designed root solver.

9.7. Although the computed roots of $f(x)$ and their multiplicities are different from the exact roots of $\hat{f}(x)$ and their multiplicities, this is a correct solution for the inexact polynomial equation because the residual of $f(x)$ calculated from the computed roots is smaller than the residual from the exact roots, in which case the residuals are equal to $3.8687e-002$ and $2.1457e-001$, respectively. It seems that the occurrence of finding all zeros of a neighbouring polynomial depends not only on the proximity of roots because the roots $x_1 = -7.6516$ and $x_4 = -5.5651$ have a small change of value with the evidence of $\lambda_1 = -7.4650$ and $\lambda_3 = -5.6731$ in the computed roots, respectively.

Figure 9.6 shows the solution of the inexact polynomial equation calculated by Newton's method, Müller's method, Zeng's algorithm and the `roots` function in MATLAB, separately. In particular, Newton's method and Müller's method are used to calculate the roots 24 times with uniformly distributed initial estimates of the roots in the range $[-10, 10]$, and the computed roots are then sorted in ascending order, as shown in Figures 9.6(i) and (ii), respectively. It is seen that both methods can obtain only the biggest root at 4.2243. Figures 9.6(iii) and (iv) show that Zeng's algorithm and the `roots` function fail to return the exact roots and their multiplicities because the multiple roots split up into a cluster of simple roots. □

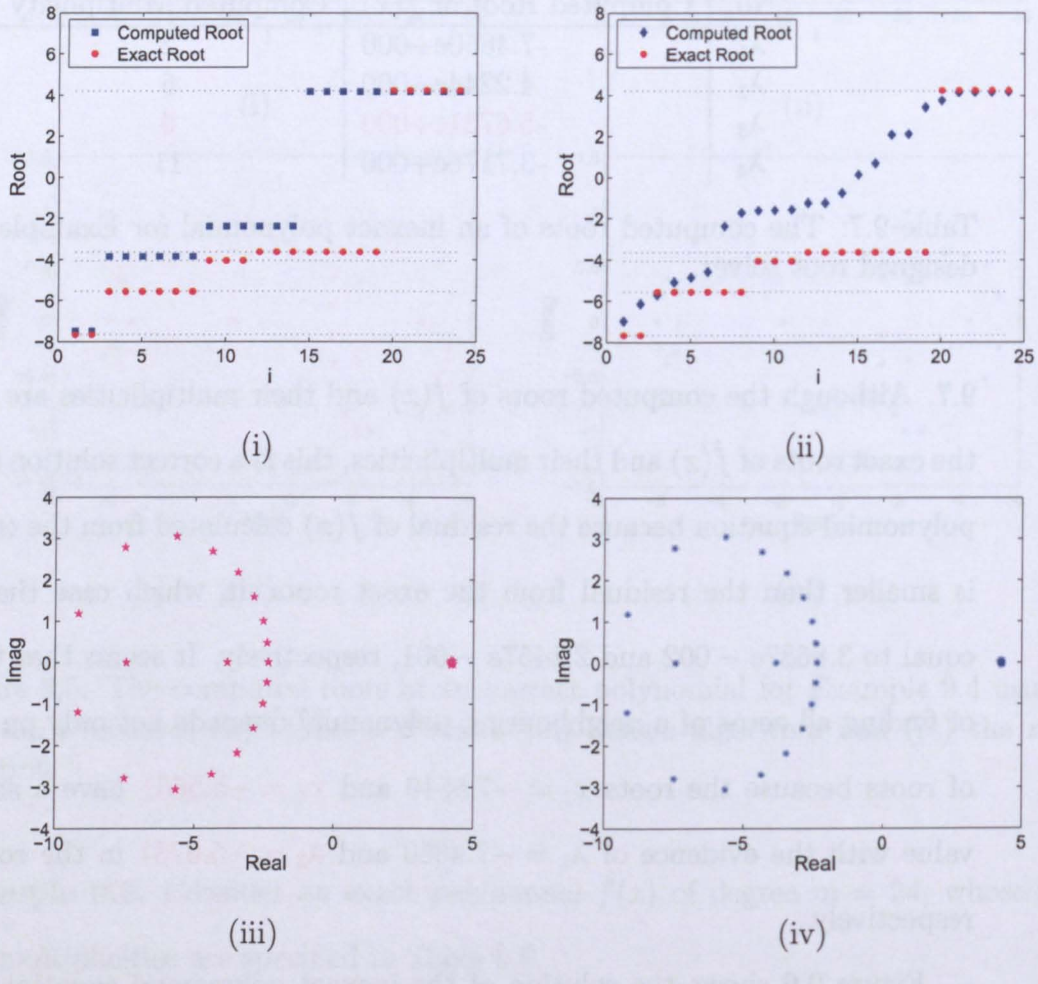


Figure 9.6: The computed roots of an inexact polynomial for Example 9.5 using (i) Newton's method, (ii) Müller's method, (iii) Zeng's algorithm and (iv) the `roots` function.

Example 9.6. Consider an exact polynomial $\hat{f}(x)$ of degree $m = 33$, whose roots and multiplicities are specified in Table 9.8. Noise with componentwise signal-to-noise ratio $\varepsilon_c^{-1} = 10^7$ was applied to $\hat{f}(x)$, thereby yielding $f(x)$.

No.	Root of $\hat{f}(x)$	Multiplicity
x_1	-0.066495	1
x_2	-6.3985	2
x_3	-6.4957	3
x_4	2.4215	4
x_5	6.8289	4
x_6	-2.3415	5
x_7	-6.2319	6
x_8	6.5445	8

Table 9.8: The roots and multiplicities of $\hat{f}(x)$ for Example 9.6.

No.	Computed Root of $f(x)$	Computed Multiplicity
λ_1	-5.9791e+000	1
λ_2	-6.6495e-002	1
λ_3	6.2191e+000	1
λ_4	2.4215e+000	4
λ_5	-2.3415e+000	5
λ_6	-6.3696e+000	10
λ_7	6.6774e+000	11

Table 9.9: The computed roots of an inexact polynomial for Example 9.6 using the designed root solver.

The designed root solver is used to compute the roots of the perturbed polynomial $f(x)$ and their multiplicities, and the results are shown in Table 9.9. It is seen that the solution of a neighbouring polynomial equation is obtained instead of the solution of the exact polynomial equation $\hat{f}(x) = 0$, such that the residuals of $f(x)$ calculated from the computed roots and exact roots are equal to $1.9766e + 002$ and $2.0476e + 002$, respectively. Similarly, this neighbouring polynomial occurs when

$\hat{f}(x)$ has a pair of close roots $x_5 = 6.8289$ with multiplicity 4 and $x_8 = 6.5445$ with multiplicity 8, and three close roots $x_2 = -6.3985$ with multiplicity 2, $x_3 = -6.4957$ with multiplicity 3 and $x_7 = -6.2319$ with multiplicity 6, in which case these close roots change to $\lambda_3 = 6.2191$ with multiplicity 1 and $\lambda_7 = 6.6774$ with multiplicity 11, and $\lambda_1 = -5.9791$ with multiplicity 1 and $\lambda_6 = -6.3696$ with multiplicity 10, respectively.

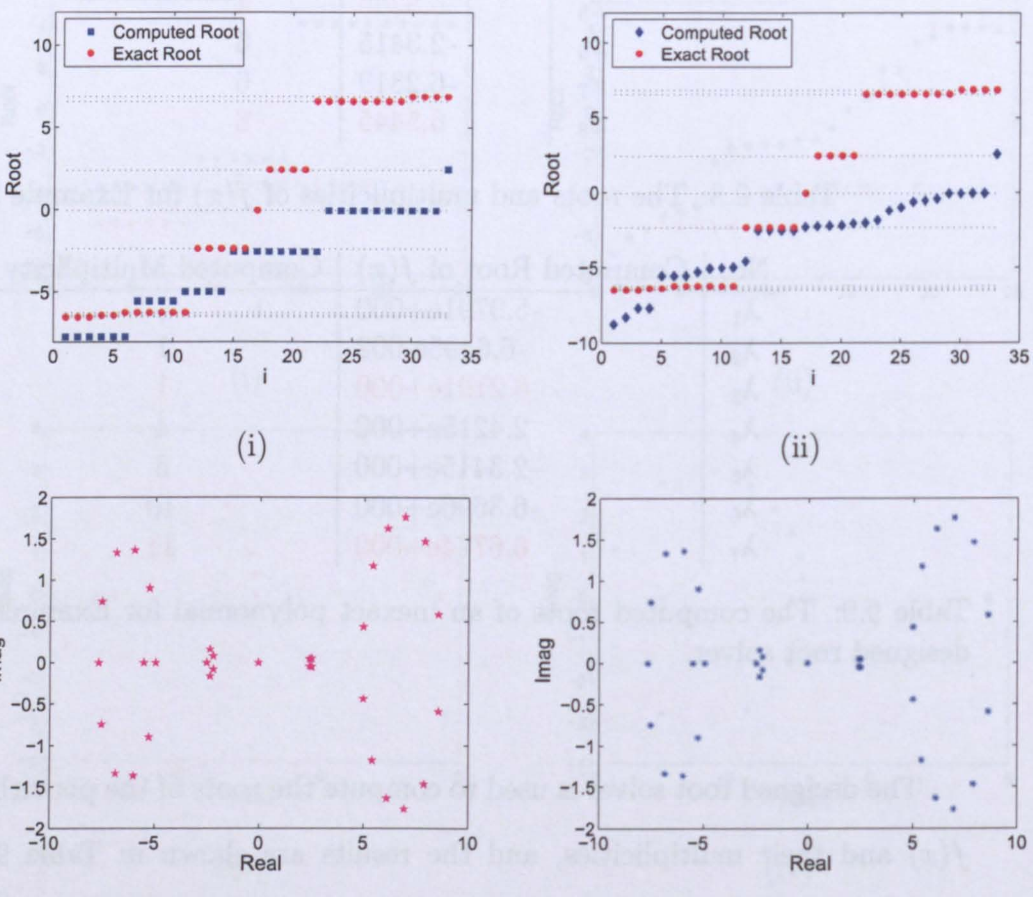


Figure 9.7: The computed roots of an inexact polynomial for Example 9.6 using (i) Newton's method, (ii) Müller's method, (iii) Zeng's algorithm and (iv) the `roots` function.

Figure 9.7 shows the solution of the inexact polynomial equation calculated by Newton's method, Müller's method, Zeng's algorithm and the `roots` function in MATLAB, separately. In particular, Newton's method and Müller's method are used to calculate the roots 33 times with uniformly distributed initial estimates in the range $[-10, 10]$, and the results are then sorted in ascending order, as shown in Figures 9.7(i) and (ii), respectively. It is seen that both methods yield the roots that equal -0.066495 and 2.4215, but are difficult to find the close distinct roots. Figures 9.7(iii) and (iv) show that Zeng's algorithm and the `roots` function fail to return the exact roots and their multiplicities because the multiple roots split up into a cluster of simple roots. $\square$

9.1 Summary

This chapter has demonstrated the success of the designed root solver for determining all zeros of an inexact polynomial compared with four algorithms, Newton's method, Müller's method, Zeng's algorithm and the `roots` function in MATLAB. It has been shown that the designed root solver may find not only the solution of the exact polynomial equation whose roots are desired, but also the theoretically exact solution of a neighbouring polynomial equation. It seems that the occurrence of the solution of a neighbouring polynomial equation depends on the proximity of the exact roots. The results of Examples 9.1 – 9.6 show that Newton's method performs better than Müller's method, but both methods fail to certify the values of all multiple roots correctly because these values vary with initial estimates of the roots. Also, these methods fail to compute the multiplicities of multiple roots, and all computed roots have unit multiplicity. It is also shown that Zeng's algorithm and the `roots` function are very sensitive to noise and roundoff errors due to floating point arithmetic with evidence of the break up of a multiple root as a cluster of simple roots, and thus they fail to return the exact roots and their multiplicities.

Chapter 10

Conclusions and future work

The main work presented in the thesis is the development of a polynomial root solver using structure preserving matrix methods. This root solver, based on a method developed by Gauss and described in Uspensky [62], involves approximate GCD computations and polynomial divisions, both of which are ill-posed computations.

The designed root solver is implemented computationally in order to calculate the multiple roots of a polynomial and their multiplicities in the presence of noise. The experiments detailed in Chapters 1 and 9 show that this root solver performs significantly better, particularly for non-trivial polynomials (high degree and many multiple roots), than the standard methods, such as Newton's method and Müller's method, as well as Zeng's algorithm and the `roots` function in MATLAB because the designed root solver retains the multiplicity structure of a polynomial and the relative errors between the exact and computed roots are approximate equal to the relative input errors.

A novel situation may occur when the designed root solver determines the multiple roots of a noisy polynomial. This is the occurrence of a neighbouring polynomial, which is nearer the given inexact polynomial than the theoretically exact polynomial whose roots are specified. This scenario is shown in Figure 9.1, and it is expected to obtain a computed solution that is the theoretically exact solution of a neighbouring polynomial equation.

It has been demonstrated in Chapter 3 that the calculation of an approximate GCD of two polynomials forms an important part of the designed root solver, and it is clear that the determination of the degree of an approximate GCD is crucial to the calculation of an approximate GCD because this is a non-trivial problem that reduces to the estimation of the rank loss of a resultant matrix of the two polynomials. The experiments detailed in Chapter 6 describe three good, and in many cases superior, methods for the determination of an approximate GCD of a noisy polynomial $f(x)$ and its derivative $f^{(1)}(x)$. It was, however, demonstrated that these three methods may return different results, and hence an attempt was made to determine the degree of an approximate GCD based on these results automatically using the method called *Majority Voting*. Also, it was found that this attempt failed for some complicated polynomials, and manual decisions were required. This therefore requires that the methods for solving the rank loss estimate problem be improved in the future.

Since the structured matrix methods can be used to solve a polynomial equation with multiple roots, future work includes the development of efficient algorithms that optimise the structure of the polynomial root solver, as mentioned in Sections 1.3 and 6.4. Moreover, future work and improvements to the designed root solver have been suggested at the end of Chapter 7. It is believed that if these changes were completed, results should be computed efficiently and improved significantly.

Bibliography

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, MA, USA, 1974.

[2] John. D. Allan. *Statistical and structured optimisation methods for the approximate GCD problem.* PhD thesis, Department of Computer Science, University of Sheffield, UK, 2008.

[3] R. Askey. *Orthogonal Polynomials and Special Functions.* SIAM, Philadelphia, USA, 1972.

[4] S. Barnett. A note on the Bezoutian matrix. *SIAM J. Appl. Math.*, 22:84–86, 1972.

[5] S. Barnett. *Polynomials and Linear Control Systems.* Marcel Dekker, New York, USA, 1983.

[6] R. C. Beach. *An Introduction to the Curves and Surfaces of Computer-Aided Design.* Van Nostrand Reinhold, New York, USA, 1991.

[7] B. Beckermann and G. Labahn. A fast and numerically stable Euclidean-like algorithm for detecting relatively prime numerical polynomials. *Journal of Symbolic Computation*, 26(6):691–714, 1998.

[8] D. Bini and P. Boito. Structured matrix-based methods for polynomial ϵ-gcd: analysis and comparisons. In *ISSAC'07 : Proc. Int. Symp. Symbolic and Algebraic Computation*, pages 9–16. ACM Press, New York, 2007.

[9] A. Björck. *Numerical Methods for Least Squares Problems.* SIAM, Philadelphia, USA, 1996.

[10] W. S. Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors. *Journal of the ACM*, 18(4):478–504, 1971.

[11] F. C. Chang. Factoring a polynomial with multiple-roots. *International Journal of Computational and Mathematical Sciences*, 2:173–176, 2008.

[12] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt. The singular value decomposition for polynomial systems. In *ISSAC'95 : Proc. Int. Symp. Symbolic and Algebraic Computation*, pages 195–207. ACM Press, New York, 1995.

[13] 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.

[14] D. K. Dunaway. Calculation of zeros of a real polynomial though factorization using Euclid's Algorithm. *SIAM J. Numer. Anal.*, 11(6):1087–1104, 1974.

[15] D. K. Dunaway and B. L. Turlington. Some major modifications to a new method for solving ill-conditioned polynomial equations. In *Proceedings of the ACM annual conference*, pages 636–643. ACM Press, New York, 1972.

[16] I. Emiris, A. Galligo, and H. Lombardi. Certified approximate univariate GCDs. *J. Pure and Applied Algebra*, 117,118:229–251, 1997.

[17] L. Foster. Generalizations of Laguerre's method. *SIAM J. Numer. Anal.*, 18:1004–1018, 1981.

[18] P. A. Fuhrmann. *A Polynomial Approach to Linear Algebra.* Universitext, Springer Verlag, New York, USA, 1996.

[19] G. R. Garside, P. Jarratt, and C. Mack. A new method for solving polynomial equations. *The Computer Journal*, 11:87–90, 1968.

[20] L. Gemignani. Structured matrix methods for polynomial root-finding. In *ISSAC'07 : Proc. Int. Symp. Symbolic and Algebraic Computation*, pages 175–180. ACM Press, New York, 2007.

[21] C. F. Gerald and P. O. Wheatley. *Applied Numerical Analysis.* Addison-Wesley, USA, 1994.

[22] S. Ghaderpanah and S. Klasa. Polynomial scaling. *SIAM J. Numer. Anal.*, 27(1):117–135, 1990.

[23] S. Goedecker. Remark on algorithms to find roots of polynomials. *SIAM J. Sci. Comput.*, 15(5):1059–1063, 1994.

[24] G. H. Golub and C. F. Van Loan. *Matrix Computations.* John Hopkins University Press, Baltimore, USA, 1996.

[25] J. Hadamard. *Lectures on the Cauchy Problem in Linear Partial Differential Equations.* Yale University Press, New Haven, USA, 1923.

[26] E. Hansen, M. Patrick, and J. Rusnack. Some modificiations of Laguerre's method. *BIT*, 17:409–417, 1977.

[27] U. Helmke and P. A. Fuhrmann. Bezoutians. *Linear Algebra and Its Applications*, 124:1039–1097, 1989.

[28] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, USA, 2002.

[29] F. B. Hildebrand. *Introduction to Numerical Analysis.* Tata McGraw-Hill, New Delhi, India, 1974.

[30] R. A. Horn and C. R. Johnson. *Matrix Analysis.* Cambridge University Press, Cambridge , UK, 1985.

[31] V. Hribernig and H. J. Stetter. Detection and validation of clusters of polynomial zeros. *Journal of Symbolic Computation*, 24:667–681, 1997.

[32] M. A. Jenkins and J. F. Traub. A three-stage variable-shift iteration for polynomial zeros and its relation to generalized Raleigh iteration. *Numerische Mathematik*, 14(3):252–263, 1970.

[33] M. A. Jenkins and J. F. Traub. Algorithm 419: Zeros of a complex polynomial. *Comm. ACM*, 15:97–99, 1972.

[34] M. A. Jenkins and J. F. Traub. Principles for testing polynomial zerofinding programs. *ACM Trans. Mathematical Software*, 1(1):26–34, 1975.

[35] G. F. Jónsson and S. Vavasis. Solving polynomials with small leading coefficients. *SIAM J. Matrix Anal. Appl.*, 26(2):400–414, 2005.

[36] B. Kågström and A. Ruhe. An algorithm for numerical computation of the jordan normal form of a complex matrix. *ACM Trans. Mathematical Software*, 6(3):398–419, 1980.

[37] W. Kahan. Conserving confluence curbs ill-condition. Technical report, Department of Computer Science, University of California, Berkeley, USA, 1972.

[38] N. Karcanias and M. Mitrouli. Normal factorisation of polynomials and computational issues. *Comput. Math. Appl.*, 45:229–245, 2003.

[39] N. K. Karmarkar and Y. N. Lakshman. On approximate GCDs of univariate polynomials. *Journal of Symbolic Computation*, 26(6):653–666, 1998.

[40] A. Kirsch. *An Introduction to the Mathematical Theory of Inverse Problems.* Springer, New York, USA, 1996.

[41] T. L. Lee, T. Y. Li, and Z. Zeng. A rank-revealing method with updating, downdating, and applications. part II. *SIAM J. Matrix Anal. Appl.*, 31:503–525, 2009.

[42] B. Li, Z. Liu, and L. Zhi. A structrued rank-revealing method for Sylvester matrix. *Journal of Computational and Applied Mathematics*, 213:212–223, 2008.

[43] T. Y. Li and Z. Zeng. A rank-revealing method with updating, downdating, and applications. *SIAM J. Matrix Anal. Appl.*, 26:918–946, 2005.

[44] B. Liang and S. U. Pillai. Blind image deconvolution using a robust 2-D GCD approach. *IEEE Int. Symp. Circuits and Systems*, pages 1185–1188, June 9-12, 1999.

[45] K. Madsen. A root-finding algorithm based on Newton's method. *BIT*, 13:71–75, 1973.

[46] D. Manocha. Numerical methods for solving polynomial equations. In D. Cox and B. Sturmfels, editors, *Proceedings of Symposia in Applied Mathematics, volume 53*, Applications of Computational Algebraic Geometry, pages 41–66. AMS, Rhode Island, USA, 1998.

[47] 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.

[48] L. Miranian and M. Gu. Strong rank revealing LU factorization. *Linear Algebra Appl.*, 367:1–16, 2003.

[49] M.Lang and B. C. FrenZel. Polynomial root finding. *IEEE Signal Processing Letters*, 1(10):141–143, 1994.

[50] S. G. Nash and A. Sofer. *Linear and Nonlinear Programming.* McGraw-Hill, 1996.

[51] M. T. Noda and T. Sasaki. Approximate GCD and its application to ill-conditioned algebraic equations. *Journal of Computational and Applied Mathematics*, 38:335–351, 1991.

[52] V. Y. Pan. Solving a polynomial equation: Some history and recent progress. *SIAM Review*, 39(2):187–220, 1997.

[53] V. Y. Pan. Computation of approximate polynomial GCDs and an extension. *Information and Computation*, 167:71–85, 2001.

[54] S. U. Pillai and B. Liang. Blind image deconvolution using a robust GCD approach. *IEEE Trans. Image Processing*, 8(2):295–301, 1999.

[55] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling. *Numerical Recipes: The Art of Scientific Computing.* Cambridge U. Press, Cambridge, UK, 1990.

[56] J. Ben Rosen, H. Park, and J. Glick. Total least norm formulation and solution for structured problems. *SIAM J. Matrix Anal. Appl.*, 17(1):110–128, 1996.

[57] J. Ben Rosen, H. Park, and J. Glick. Structured total least norm for nonlinear problems. *SIAM J. Matrix Anal. Appl.*, 20(1):14–30, 1998.

[58] A. Schönhage. Quasi-gcd computations. *J.Complexity*, 1(1):118–137, 1985.

[59] G. A. Sitton, C. S. Burrus, J. W. Fox, and S. Treitel. Factoring very high degree polynomials. *IEEE Signal Processing Magazine.*, 20(6):27–42, 2003.

[60] P. Stoica and T. Söderström. Common factor detection and estimation. *Automatica*, 33(5):985–989, 1997.

[61] D. Triantafyllou and M. Mitrouli. On rank and null space computation of the generalized Sylvester matrix. *Numerical Algorithms*, 54(3):297–324, 2010.

[62] J. V. Uspensky. *Theory of Equations.* McGraw-Hill, New York, USA, 1948.

[63] D. S. Watkins. *Fundamentals of Matrix Computations.* John Wiley and Sons, New York, USA, 1991.

[64] J. Wilkinson. The evaluation of zeros of ill-conditioned polynomials. *Numerische Mathematik*, 1:150–166, 1959.

[65] J. H. Wilkinson. *Rounding Errors in Algebraic Processes.* Prentice Hall, Englewood Cliffs, New Jersey, 1963.

[66] J. R. Winkler. Polynomial roots and approximate greatest common divisors. Technical report, Department of Computer Science, The University of Sheffield, United Kingdom, 2007.

[67] J. R. Winkler and J. D. Allan. Structured total least norm and approximate GCDs of inexact polynomials. *Journal of Computational and Applied Mathematics*, 215:1–13, 2008.

[68] C. J. Zarowski. The MDL criterion for rank determination via effective singular values. *IEEE Trans. Signal Processing*, 46(8):1741–1744, 1998.

[69] 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.

[70] Z. Zeng. A method for computing multiple roots of inexact polynomials. In *ISSAC'03 : Proc. Int. Symp. Symbolic and Algebraic Computation*, pages 266–272. ACM Press, New York, 2003.

[71] Z. Zeng. Multroot - a Matlab package computing polynomial roots and multiplicities. *ACM Trans. Mathematical Software*, 30(2):218–236, 2004.

[72] Z. Zeng. Computing multiple roots of inexact polynomials. *Mathematics of Computation*, 74(250):869–903, 2005.

[73] L. Zhi and Z. Yang. Computing approximate GCD of univariate polynomials by structured total least norm. Technical Report 24, MMRC, AMSS, Academia Sinica, MM Research Preprints, 2004.