Statistical and structured optimisation methods for the approximate GCD problem

by

John Allan

A thesis submitted to the
Computer Science
in conformity with the requirements for
the degree of PhD

Sheffield University
England

December 2008

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The computation of polynomial greatest common divisors (GCDs) is a fundamental problem in algebraic computing and has important widespread applications in areas such as computing theory [1], control [2], image processing [23], signal processing [25] and computer-aided design (CAD) [32].

Example 1.1. Image processing: Blind image convolution involves identifying both the true image and the blurring function from a degraded image. This process is critical since in many practical applications (such as *astronomy*, *x-ray imaging* and *real time video conferencing*) the blur function (caused by camera motion, slow shutter speed etc) is often unknown. This problem may be solved by obtaining two images of the same scene from which the true image may be recognised as the GCD of the two blurred images. More details can be found in [23]. □

As the name suggests, the GCD of two polynomials, $p(x)$ and $q(x)$, is a polynomial

$d(x)$ of greatest possible degree that divides both $p(x)$ and $q(x)$ exactly, such that,

$$p(x) = u(x)d(x)$$

$$q(x) = v(x)d(x),$$

where $u(x)$ and $v(x)$ are co-prime.

The Euclidean algorithm [34] is an effective method for solving this problem when the coefficients of the polynomials are known exactly. However, the introduction of even an arbitrarily small perturbation can cause $p(x)$ and $q(x)$ to become co-prime and reduce $d(x)$ to a scalar.

Example 1.2. Consider the polynomials,

$$p(x) = (x - \alpha)(x - \beta)(x - \gamma)$$

$$q(x) = (x - \alpha)(x - \gamma)(x - \lambda),$$

where $\beta \neq \lambda$, whose GCD is,

$$d(x) = GCD(p(x), q(x)) = (x - \alpha)(x - \gamma).$$

If $p(x)$ is perturbed such that,

$$p(x) \rightarrow \tilde{p}(x) = (x - (\alpha + \delta\alpha))(x - \beta)(x - (\gamma + \delta\gamma)),$$

where $\alpha + \delta\alpha \neq \gamma + \delta\gamma \neq \lambda$ and $\delta\alpha, \delta\gamma \neq 0$, then $\deg(\mathrm{GCD}(\tilde{p}(x), q(x))) = 1$. □

This is a major problem in practical applications where it is common for the coefficients to become corrupted by noise. This may be as a result of floating point accuracy or the involvement of laboratory measurements, which allow only a limited number of significant figures to be obtained. This occurs in many applications such as those involved in the field of computer aided design. In this area GCD operations are used for computing intersections of curves and surfaces. The problems associated

with this are highlighted in the next section

1.1 CAD systems

CAD systems have fundamentally changed the way that product design is done and they are used for everything from garden design to aerospace engineering in order to allow companies to obtain a competitive advantage. Additionally, CAD-powered systems such as CAD/CAM (Computer aided manufacturing), allow manufacturers to achieve greater productivity in less time. Such systems are increasingly being used to automate and manage the manufacturing process.

1.1.1 Benefits of CAD systems

CAD systems provide many benefits and services, some of which are highlighted below.

- **Creating precise product designs:** CAD systems have the ability to produce very precise product designs, because of the successful integration of the design, analysis (e.g. finite element analysis) and manufacturing processes.

- **Solid 3D modelling:** It is possible for CAD systems to create photo-realistic models allowing manufacturers the ability to manipulate them as if they were tangible objects. This allows the model to be experimented with, without the need for expensive solid prototypes.

- **Reverse engineering:** CAD systems allow the virtual disassembling of a physical product to study all of its parts and how they fit together which allows analysis time to be reduced.

- **Finite element analysis:** This allows the responses of structures and materials to environmental issues to be predicted. Such factors may be force, heat and vibration and this allows the ability to create a more viable and high quality product.

1.1.2 Problems in CAD

Many CAD users are unaware that in order to reduce the computations to more manageable levels, the edge curves of the solid models may be merely approximations to the actual intersection of two surfaces. Stephen Wolfe [40] reports that, according to a paper from Sandia National Laboratories, the precise intersection of two NURBS[1] surfaces must be described by a polynomial of degree 54. The approximating polynomial, which is generally of lower degree than the exact one, therefore usually lies close to, but not precisely on, the two surfaces it bounds. Because of these approximations, there may be gaps between edges and the faces of boundary-representation solids. A consequence of these gaps becoming too large is that the CAD program may be unable to distinguish between the inside of the model and the universe around it, causing the model to be reported as "corrupted". Most CAD systems employ a strategy to calculate the maximum allowable gap between an edge and the two surfaces it bounds. Unfortunately, different systems employ different strategies for calculating this tolerance. Problems such as this have led to one maker of aircraft engines spending over $24 million per year identifying and resolving them.

[1] Nonuniform rational B-spline. See `http://en.wikipedia.org/wiki/NURBS`

1.1.3 Polynomials in CAD

If two objects intersect the surface that defines them will have a non-constant divisor. As a consequence of this a significant number of important CAD/CAM problems reduce directly to both the solving of polynomial equations and computing common factors through GCD operations. Many of the difficulties associated with this are increased significantly when dealing with multiple roots, which occur when two or more surfaces are tangential. These arise, usually by design, in fairing and blending operations, and curvature continuous surface intersections are not uncommon in ship-hull design. In these situations one approach is to attempt to increase the accuracy of any numerical procedures through the use of interval-arithmetic enclosure methods [28]. However, a common problem caused by multiple roots is that the enclosure cannot exclude the presence of a root in many intervals near the true root as the polynomial evaluation is not accurate enough [15].

Many of the problems caused by complex geometry with tangential and curvature-continuous contacts, based on user specifications and proprietary algorithms, can be controlled whilst the geometry remains under the control of the CAD system. However, when the geometry is exported to other CAD systems or analysis programs for additional manipulation, further information which simplifies difficult intersections for the original CAD system, such as the the rail curves of blending surfaces, is removed.

In addition to these approximations, inaccuracies in the input data, caused by roundoff error due to floating point arithmetic and any experimental equipment limitations, can lead to contradictory information about the input object. For example, the object description may require n adjacent faces to meet at a common vertex,

while the numerical plane coefficients may specify n planes that intersect in n closely spaced points rather than a single one.

Clearly procedures that would enable CAD systems to effectively perform GCD operations on inexact polynomials, which may be of high degree and contain multiple roots, would be highly desirable. Consequently, the methods developed in this thesis, while being presented in the power basis, are equally applicable in the Bernstein basis and should be of interest to researchers in geometric modelling as well as those involved in scientific computing. The layout of the thesis is now detailed.

1.2 Thesis layout

Many of the problems that arise when attempting to carry out computations upon polynomials are as a result of inaccuracies in the coefficients and the ill-conditioned nature of the problem. Therefore, in Chapter 2 the concept of ill-conditioning is introduced along with illustrations of how multiple roots can behave in a pathologically ill-conditioned manner. It is then shown, in Chapter 3, that repeated GCD operations can be used, in certain circumstances, to stabilise the process of solving polynomials with multiple roots along with an example of the need for an *approximate GCD*, which is defined in Chapter 4. An overview of the major approaches taken by other works when solving the approximate GCD problem, is also discussed in this chapter.

In Chapter 5 a method for calculating the rank of a resultant matrix using maximum likelihood is developed. This aims to calculate the rank of a matrix without prior knowledge of the noise in the data. Chapter 6 then presents a method of calculating the approximate GCD of two polynomials using structured matrix methods to enforce a number of relationships between the elements in the system. The methods

from these two chapters are then combined in Chapter 7 and tested on a random set of polynomials. The viability of using the GCD algorithm to calculate the *multiplicity structure* of two polynomials is then discussed in Chapter 8 and possible future extensions to the work are detailed in Chapter 9 along with a summary of the results and methods.

1.3 Thesis contribution

Some of the contributions to the problem of approximate GCD computations covered in this thesis are:

- The use of the principle of maximum likelihood for the estimation of the rank of an inexact resultant matrix.
- The use of structured matrix methods in order to obtain an approximate GCD of two inexact polynomials.
- An investigation into the use of bounded least squares in conjunction with structured matrix methods to limit the perturbations that may be applied when calculating an approximate GCD.
- An investigation into the use of a sequence of nested structured matrix methods in order to calculate the theoretically exact multiplicities of the roots of an inexact polynomial.

1.3.1 Papers published

The following papers have been published as a consequence of the work described in this thesis:

1. Structured total least norm and approximate GCDs of inexact polynomials, *Journal of Computational and Applied Mathematics*, vol. 215, 2008, pp. 1-13.

2. Structured low rank approximations of the Sylvester resultant matrix for approximate GCDs of Bernstein polynomials. To appear in *Electronic Transactions on Numerical Analysis.*

Chapter 2

Ill-conditioning

In [36] Wilkinson remarked that:

> "A tendency to underestimate the difficulties in working with general polynomials is perhaps a consequence of one's experience in classical analysis. There it is natural to regard a polynomial as a very desirable function since it is bounded in any finite region and has derivatives of all orders. In numerical work, however, polynomials having coefficients which are more or less arbitrary are tiresome to deal with by entirely automatic procedures."

In order to appreciate why algorithms involving polynomials can fail, it is necessary to understand the concept of *forward error*, *backward error* and how the *condition* of the system and its measure, called the *condition number*, describes the effect of errors on a computed solution. In this chapter these concepts are discussed along with the conditioning of the roots of a polynomial with particular emphasis on the effect of the roots' multiplicity.

2.1 Forward and backward errors and condition numbers

Suppose that we compute an approximation $\tilde{y}$ to a polynomial function $y = f(x)$ where $f, x \in \mathbb{R}$. How can the quality of $\tilde{y}$ be determined? In general a vanishingly small relative error,

$$E_{\text{rel}}(\tilde{y}) = \frac{|y - \tilde{y}|}{|y|},$$

is most desirable. However, this is not always possible, since the exact answer may not be known, so instead the set of data, $\{x + \delta x\}$, for which the problem was actually solved, is examined. The values of $|\delta x|$ and $|\delta x|/|x|$, are the absolute and relative *backward errors* whilst the values of $|y - \tilde{y}|$ and $E_{\text{rel}}(\tilde{y})$ are the *forward errors*. The relationship between them is illustrated in Figure 2.1, which is reproduced from [13].

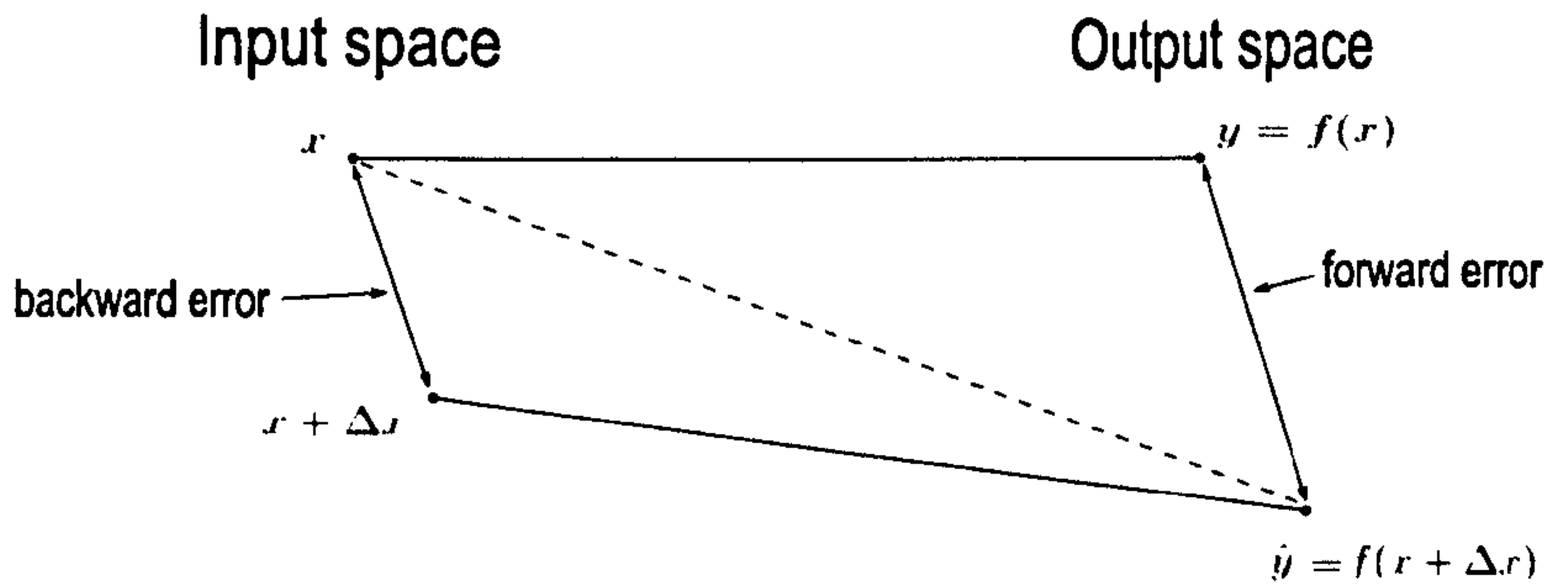

Figure 2.1: Backward and forward errors for $y = f(x)$. Solid line = exact; dotted line = computed.

Backward error analysis, the process of bounding the backward error of a computed solution, provides an important concept relating to the uncertainties in the

data:

> "If the backward error is no larger than these uncertainties then the computed solution can hardly be criticised - it may be the solution that we are seeking, for all we know." [13]

In other words, given a random perturbation and its associated backward error, it is possible to define a solution space in which exists a whole family of solutions, all of which are equally valid.

The forward and backward errors are directly related to the *condition* a system, which describes its sensitivity to perturbations in its parameters. In the context of this thesis perturbations may be defined as random variables drawn from a proposed distribution, while the parameters are the polynomial coefficients. If tiny perturbations in the input space, corresponding to a small backward error, result in a *comparatively* large change in the solution space, i.e. a large forward error, then the system is said to be *ill-conditioned.* The term 'ill-conditioned' is regularly misused by applying the term to a matrix rather than to the class of perturbations to which that matrix is exposed. Wilkinson [36] emphasises that:

> "...it is the susceptibility of *the required solution* to changes in the parameters which determines the condition of the problem. Thus it is meaningless to state that a matrix A is ill-conditioned. For example, it can be ill-conditioned with respect to the calculation of its inverse or the calculation of its eigenvalues or eigenvectors, but in general, ill-condition with respect to one of these will not imply ill-condition with respect to any other."

A measure of the sensitivity of a function to evaluation with respect to a class of perturbations applied to the data (input parameters) is called its *condition number.*

Condition number: A condition number is a measure of a problems' sensitivity to perturbations with a small condition number indicating a stable problem and a large condition number indicating a highly sensitive problem.
Condition numbers can be considered in both the normwise, κ_n, or componentwise sense, κ_c. A normwise condition number considers the norm of the elements, i.e. the condition of the system as a whole, whilst componentwise condition number considers the condition of the individual components of the system.

The magnitude of the calculated condition number gives a direct representation of the problem's sensitivity. A very large condition number, $\kappa_c \gg 1$, indicates that the problem is *ill-conditioned* and hence highly sensitive to perturbations, whilst a small condition number means that the problem is robust with respect to the class of perturbations for which κ_c is defined.

The relationship of the condition number to the forward and backward errors, in the case of a simple root, is given by,

$$\text{forward error} \approx \text{backward error} \times \text{condition number}.$$

Now that terms have been defined, the conditioning of roots is explained.

2.2 Ill-conditioned roots

Typically, computing the roots of a polynomial of high degree with multiple roots is regarded as being a highly ill-conditioned problem with an arbitrarily small backward error resulting in a forward error of much greater magnitude. Any multiple roots will generally, on the introduction of an arbitrary perturbation, split into a cluster, as

demonstrated in the following example.

Example 2.1. Consider the polynomial,

$$\begin{aligned} p(x) &= x^{10} - 10x^9 + 45x^8 - 120x^7 + 210x^6 - 252x^5 + 210x^4 - 120x^3 + 45x^2 - 10x + 1 \\ &= (x-1)^{10} \end{aligned} \tag{2.1}$$

and perturb the constant term by ε where $|\varepsilon| \ll 1$ such that (2.1) becomes,

$$\tilde{p}(x) = (x-1)^{10} - \varepsilon,$$

if $\varepsilon = 2^{-10} = 1/1024$ then

$$\tilde{p}(x) = (x-1)^{10} - 2^{-10} \tag{2.2}$$

Euler's formula states,

$$\exp(i\theta) = \cos\theta + i\sin\theta.$$

From which it follows that,

$$\exp(i2\pi k) = \cos 2\pi k + i\sin 2\pi k = 1. \tag{2.3}$$

Then from (2.2) and (2.3) the solution of $\tilde{p}(x) = 0$ is,

$$\begin{aligned} (x-1)^{10} &= 2^{-10}\exp(2i\pi k), \\ (x-1) &= 2^{-1}\exp\left(\frac{2i\pi k}{10}\right), \qquad k = 0,\ldots,9 \\ x &= 1 + \frac{\exp\left(\frac{2i\pi k}{10}\right)}{2}. \end{aligned}$$

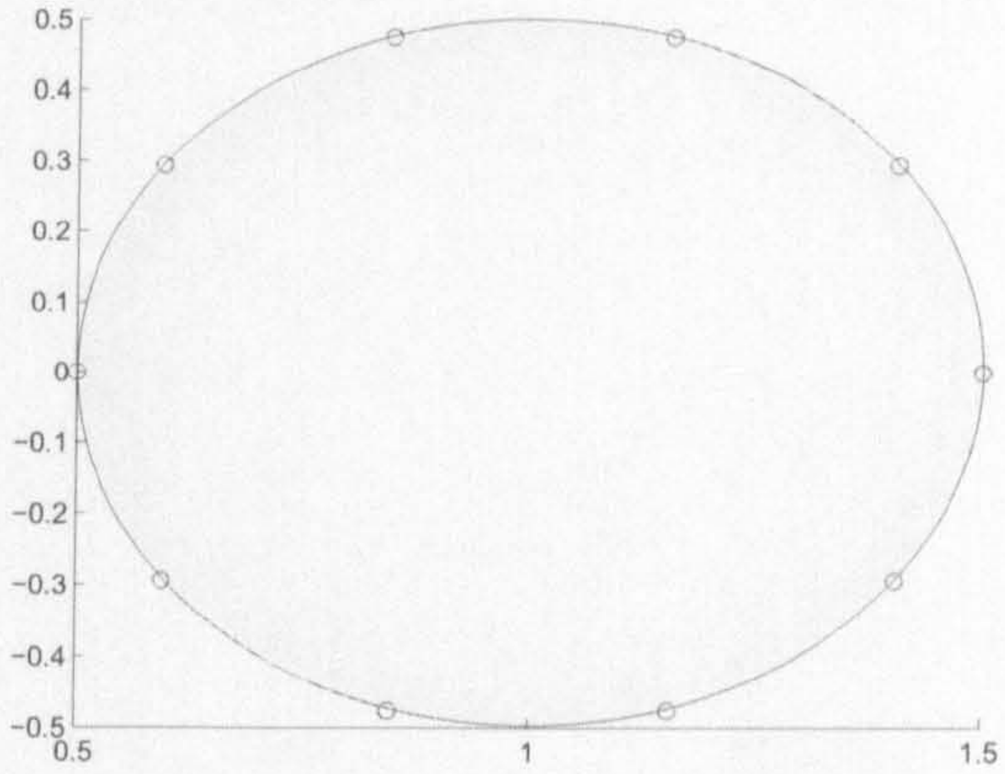

Figure 2.2: Perturbation region, in the complex plane, of the roots of $p(x) = (x-1)^{10}$ after the constant term has been perturbed by -2^{-10}.

As can be seen from Figure 2.2, introducing a perturbation as small as 2^{-10} to the constant coefficient can cause a relative error of over 50%. This error magnification increases with the multiplicity of the root.

However, Figure 2.3 illustrates how the difference between the size of each successive perturbation region decreases as the multiplicity increases. This indicates that there is an upper bound for the radius of the region which is achieved as the multiplicity of the root tends towards infinity. By inspection, this upper limit is seen to be 1 and this can easily be shown to be the case if the equation $\tilde{p}(x) = (x-1)^n - 2^{-10} = 0$ is considered, where n is large such that $n \to \infty$,

$$\begin{aligned} (x-1)^n &= 2^{-10} \\ (x-1) &= 2^{-10/n} \exp\left(\frac{2\pi i k}{n}\right), \qquad k = 0, \ldots, (n-1). \end{aligned}$$

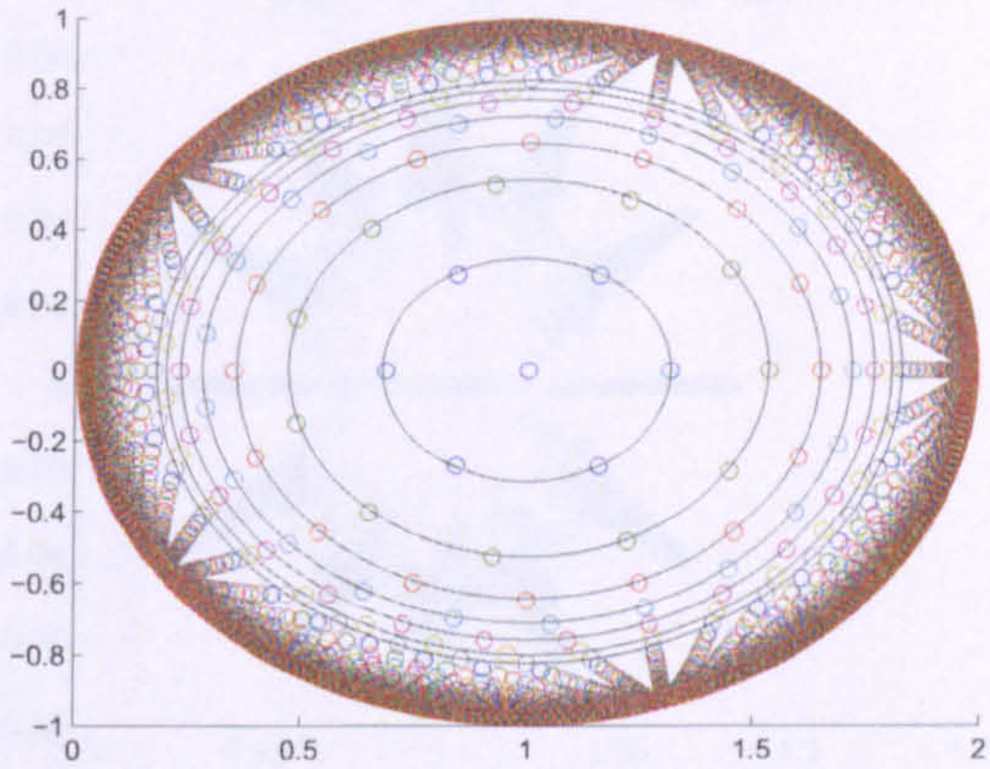

Figure 2.3: Perturbation regions of $p(x) = (x-1)^n, \quad n = 1, 6, 11, \ldots$ after the constant term has been perturbed by -2^{-10}.

Hence, as $n \to \infty$, $|x-1| \to 1$. $\square$

This natural clustering of roots would seem to suggest that the original root could be reconstructed by methods such as defining the value of the multiple root as being equal to the arithmetic mean of the cluster. An initial examination of Figure 2.4 which shows the root distribution of the polynomial $p(x) = (x-1)^6$, whose coefficients have been randomly perturbed and roots calculated 1000 times, suggests that this approach is acceptable provided there is prior information that the polynomial contains an isolated multiple root.

This is, however, not a viable solution. The experiment is repeated in Figure 2.5, with two multiple roots whose separation is reduced. It can be seen that the clusters begin to interfere with one another the closer they become, until the two are indistinguishable. A further example of the unreliability of clustering multiple roots

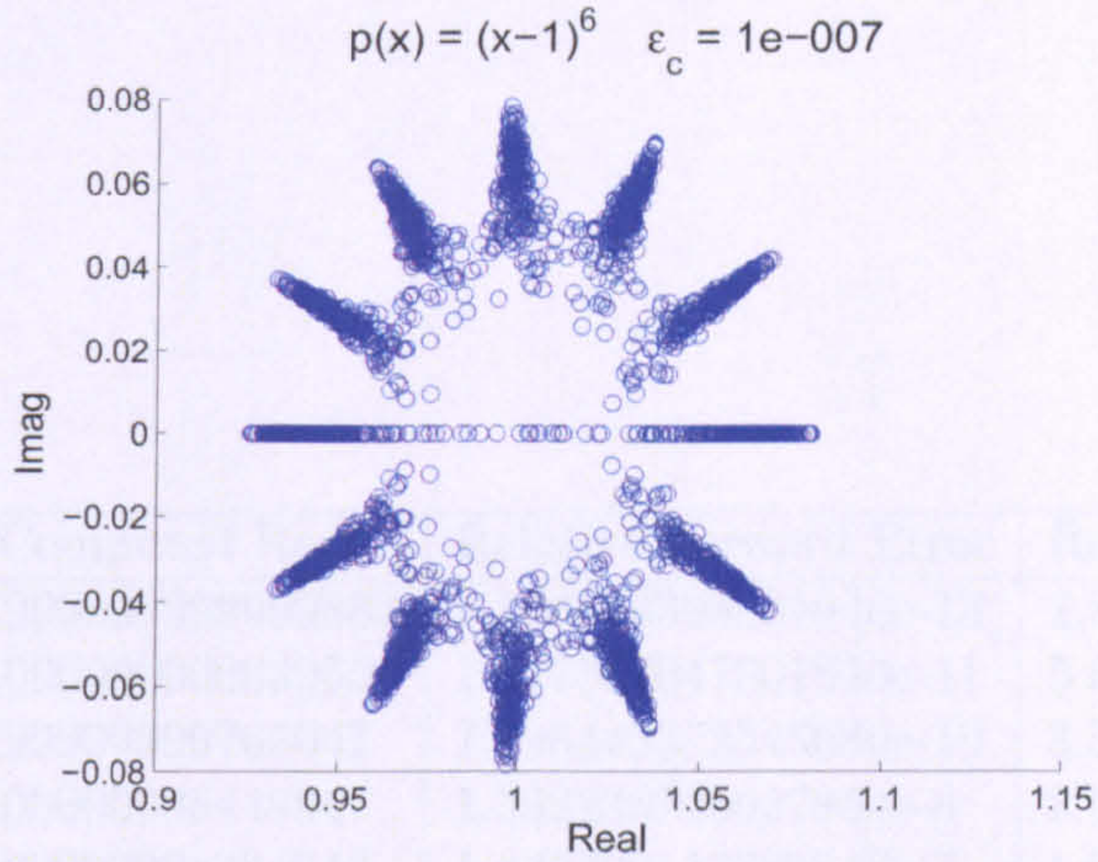

Figure 2.4: The root distribution of $p(x) = (x-1)^6$ after the coefficients have been perturbed and roots calculated 1000 times. ε_c = maximum relative error in the coefficients.

can be found in section 4.1 of [24].

However, we must not be overly preoccupied by the sensitivity of multiple zeros as this may blind us to the fact that polynomials with zeros that cannot be described as pathologically close may be extremely ill-conditioned [36].

Example 2.2. Consider the polynomial,

$$p(x) = \prod_{i=1}^{20}(x-i), \tag{2.4}$$

which is known as Wilkinson's polynomial. The roots, which were computed using a Newton-Raphson solver [34] *p.174-179*, along with their forward and backward errors are shown in Table 2.1.

It can be clearly seen from the Table 2.1 that a relatively small error in the input

Exact Root	Computed Root	Relative Forward Error	Relative Backward Error
1	0.99999999999883	1.166844398881040e-13	2.805976830848637e-16
2	2.00000000003955	1.977507047001836e-11	5.065334798104484e-16
3	2.99999999766047	7.798443372545686e-10	3.555399230054017e-16
4	4.00000005448947	1.362236745627854e-8	2.642783129408560e-16
5	4.99999928384543	1.432309147730848e-7	1.508986599057591e-16
6	6.00000611580954	1.019301589651652e-6	1.172204028875028e-16
7	6.99996498100788	5.002713159666225e-6	7.669506888764093e-17
8	8.00013019258004	1.627407250470725e-5	2.901410194285699e-17
9	8.99972563147745	3.048539139428720e-5	9.679882673970336e-18
10	10.0000976596732	9.765967315367164e-6	1.680754190412488e-17
11	11.0012104299577	1.100390870678390e-4	6.145029453654695e-18
12	11.9963438017751	3.046831854118063e-4	4.839313005234271e-17
13	13.0046321163424	3.563166417256488e-4	6.512737349296931e-18
14	13.999577286005	3.019385678561447e-5	1.791278019801270e-18
15	14.9920296810649	5.313545956755187e-4	1.368611956684288e-17
16	16.0139410552817	8.713159551059224e-4	7.731005708759904e-17
17	16.9875352381577	7.332212848430017e-4	3.438849867457723e-17
18	18.0064991667242	3.610648180085718e-4	1.162029471036370e-16
19	18.9980571410393	1.022557347732875e-4	8.250228276947081e-17
20	20.0002501670688	1.250835343959977e-5	1.076569462240355e-16

Table 2.1: Analysis of the computed roots of (2.4).

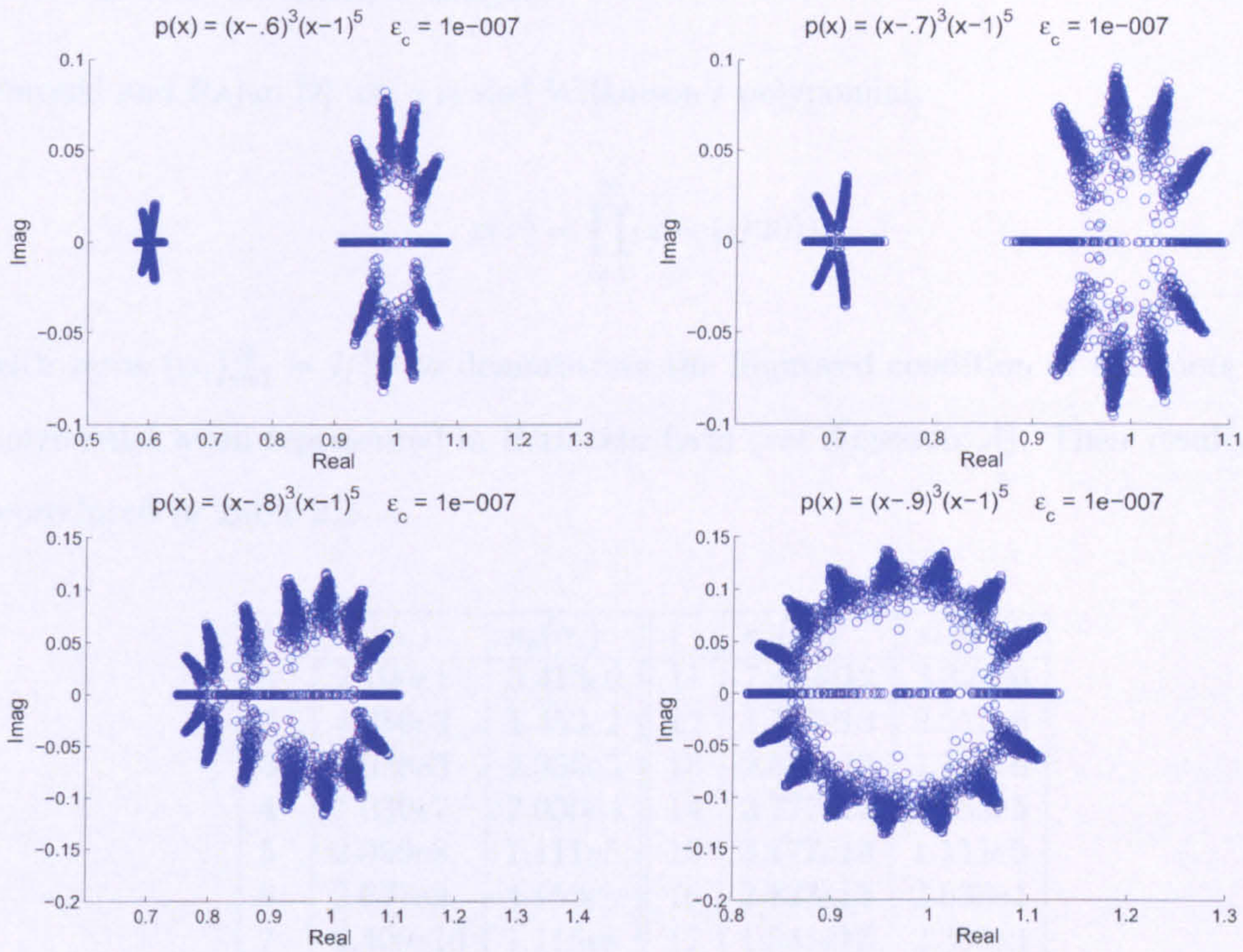

Figure 2.5: The root distribution of $p(x)$ after the coefficients have been perturbed and roots calculated 1000 times. ε_c = maximum error coefficient.

space (the backward error), which is the result of finite precision arithmetic, causes a significantly larger error in the solution space (the forward error).

The ill-conditioning of this polynomial, whose roots are both simple and well separated, is discussed in detail in [36] and clearly illustrates that the roots of a polynomial can be ill-conditioned regardless of multiplicity or proximity. In 1984 Wilkinson wrote of the discovery of the polynomial,

> "Speaking for myself I regard it as the most traumatic experience in my

career as a numerical analyst."

Farouki and Rajan [9] use a scaled Wilkinson's polynomial,

20

$$p(x) = \prod_{i=1}(x - (i/20)). \tag{2.5}$$

with zeros $\{\alpha_i\}_{i=1}^{20} = i/20$ to demonstrate the improved condition of the roots of a polynomial when represented in Bernstein form (*see Appendix A*). Their results are reproduced in Table 2.2.

i	$\kappa_p(\alpha_i)$	$\kappa_b(\alpha_i)$	i	$\kappa_p(\alpha_i)$	$\kappa_b(\alpha_i)$
1	2.100e1	3.413e0	11	7.822e12	3.321e6
2	4.389e3	1.453e2	12	1.707e13	2.215e6
3	3.028e5	2.335e3	13	2.888e13	1.115e6
4	1.030e7	2.030e4	14	3.777e13	4.153e5
5	2.059e8	1.111e5	15	3.777e13	1.111e5
6	2.677e9	4.153e5	16	2.833e13	2.030e4
7	2.409e10	1.115e6	17	1.541e13	2.335e3
8	1.566e11	2.215e6	18	5.742e12	1.453e2
9	7.570e11	3.321e6	19	1.310e12	3.413
10	2.775e12	3.797e6	20	1.378e11	0

Table 2.2: Componentwise condition numbers for (2.5). $\kappa_p(x_0)$ - Power basis condition numbers, $\kappa_b(x_0)$ - Bernstein basis condition numbers.

It can be clearly seen that the Bernstein basis has dramatically improved the condition of the roots over their power basis representations. Indeed, this is one of the primary reasons for its use in geometric modelling. However, while this improvement in stability is welcome, performing analysis on inexact polynomials, especially those with multiple roots, is still a highly non-trivial problem which cannot be dealt

with by merely changing the basis in which the polynomial is described. Instead, robust methods must be developed that can gracefully handle polynomials of the type detailed in this chapter.

2.3 Summary

In this chapter the concept of ill-conditioning has been introduced in addition to its relationship to the forward and backward errors. In addition, both theoretical and practical examples of ill-conditioned roots have been presented with evidence of increasing instability of a root as its multiplicity increases and it moves out of isolation. The Wilkinson polynomial was also introduced to show that root separation and multiplicity, whilst being contributing factors in the stability of a polynomial, are not the only causes of ill-conditioning.

In the next chapter the role of a GCD calculations in helping to deal with with the evaluation of polynomials with multiple roots is discussed. This is done in order to provide further motivation as to the need for methods developed in this thesis.

Chapter 3

Solving polynomials with multiple roots

In [34] Uspensky discusses a process which was known as early as 1863 by Gauss, which involves '*removing*' the multiplicities of the roots by reducing the problem to that of solving a sequence of polynomial equations that contain only simple roots. This can be achieved through repeated GCD and polynomial division operations as discussed below.

3.1 Factorisation via GCD operations

Consider the polynomial,

$$p(x) = (x - \alpha_1)^{m_1}(x - \alpha_2)^{m_2} \dots (x - \alpha_n)^{m_n}$$

with m_i indicating the multiplicity of the root, α_i and $m_{\max} = \max(\{m_i\}_{i=1}^{n})$. Since a root of multiplicity m_i in $p(x)$ is a root of multiplicity $m_i - 1$ in $p^{(1)}(x)$, where $p^{(j)}$ is the jth derivative of $p(x)$, it follows that,

$$GCD(p(x), p^{(1)}(x)) = (x - \alpha_1)^{m_1 - 1}(x - \alpha_2)^{m_2 - 1} \dots (x - \alpha_n)^{m_n - 1} r(x).$$

Clearly, $r(x)$ must be a constant otherwise it would have to be a polynomial containing the term, $x - \alpha_i$, meaning that $p^{(1)}(x)$ is divisible by $(x - \alpha_i)^{m_i}$ which is not possible. Therefore, (3.1) can be written as,

$$GCD(p(x), p^{(1)}(x)) = (x - \alpha_1)^{m_1 - 1}(x - \alpha_2)^{m_2 - 1} \dots (x - \alpha_n)^{m_n - 1}$$

Let $\lambda_1 = \lambda_1(x)$ be the product of all linear factors corresponding to simple roots; $\lambda_2 = \lambda_2(x)$ be the the product of all those factors corresponding to double roots, etc. Note: $\lambda_i = \lambda_i(x)$ will be set to a constant if no roots of multiplicity i exist. Therefore,

$$\lambda_1 \lambda_2^2 \lambda_3^3 \dots \lambda_{m_{\max}}^{m_{\max}},$$

only differ from $p(x)$ by a constant term. Thus,

$$d_0(x) = \lambda_2 \lambda_3^2 \lambda_4^3 \dots \lambda_{m_{\max}}^{m_{\max} - 1} = GCD(p(x), p^{(1)}(x)).$$

Similarly,

$$\begin{aligned} d_1(x) &= \lambda_3\lambda_4^2\lambda_5^3\ldots\lambda_{m_{\max}}^{m_{\max}-2} = GCD(d_0(x), d_0^{(1)}(x)) \\ d_2(x) &= \lambda_4\lambda_5^2\lambda_6^3\ldots\lambda_{m_{\max}}^{m_{\max}-3} = GCD(d_1(x), d_1^{(1)}(x)) \\ &\vdots \end{aligned}$$

This sequence ends at $d_{m_{\max}}(x)$ which is a constant term. A sequence of polynomials, $\{f_i(x)\}_{i=0}^{m_{\max}}$, can now be defined such that,

$$\begin{aligned} f_0(x) &= \tfrac{p(x)}{d_0(x)} = \lambda_1\lambda_2\lambda_3\ldots \\ f_1(x) &= \tfrac{d_0(x)}{d_1(x)} = \lambda_2\lambda_3\ldots \\ f_2(x) &= \tfrac{d_1(x)}{d_2(x)} = \lambda_3\ldots \\ &\vdots \\ f_{m_{\max}}(x) &= \tfrac{d_{m_{\max}-1}}{d_{m_{\max}}} = \lambda_{m_{\max}}, \end{aligned}$$

from which all the functions, $\lambda_1, \lambda_2, \ldots$, can be found,

$$\lambda_1 = \frac{f_0}{f_1}, \quad \lambda_2 = \frac{f_1}{f_2}, \quad \lambda_{m_{\max}-1} = \frac{d_{m_{\max}-1}}{d_{m_{\max}}}, \quad \lambda_{m_{\max}} = d_{m_{\max}}.$$

This leads to the polynomial equations,

$$\lambda_1 = 0, \quad \lambda_2 = 0, \ldots,$$

which contain only simple roots. These roots give at once the simple, double, triple, etc, roots of $p(x)$, where the index, i of m_i indicates the multiplicity. Note: If m_i is

constant then there is no root of multiplicity i. The advantages of this approach are illustrated by a simple numerical example.

Example 3.1. Consider the polynomial,

$$p(x) = (x - 0.5)^{10}(x - 1)^{30} = x^{40} - 35x^{39} + \cdots - 0.048828125x + 0.0009765625.$$

If the roots are now calculated directly from the coefficients using MatLab's `roots()` function the results illustrated in Figure 3.1 clearly demonstrate that a totally unacceptable result has been obtained. There is nothing to suggest that there are two distinct roots, let alone their actual values.

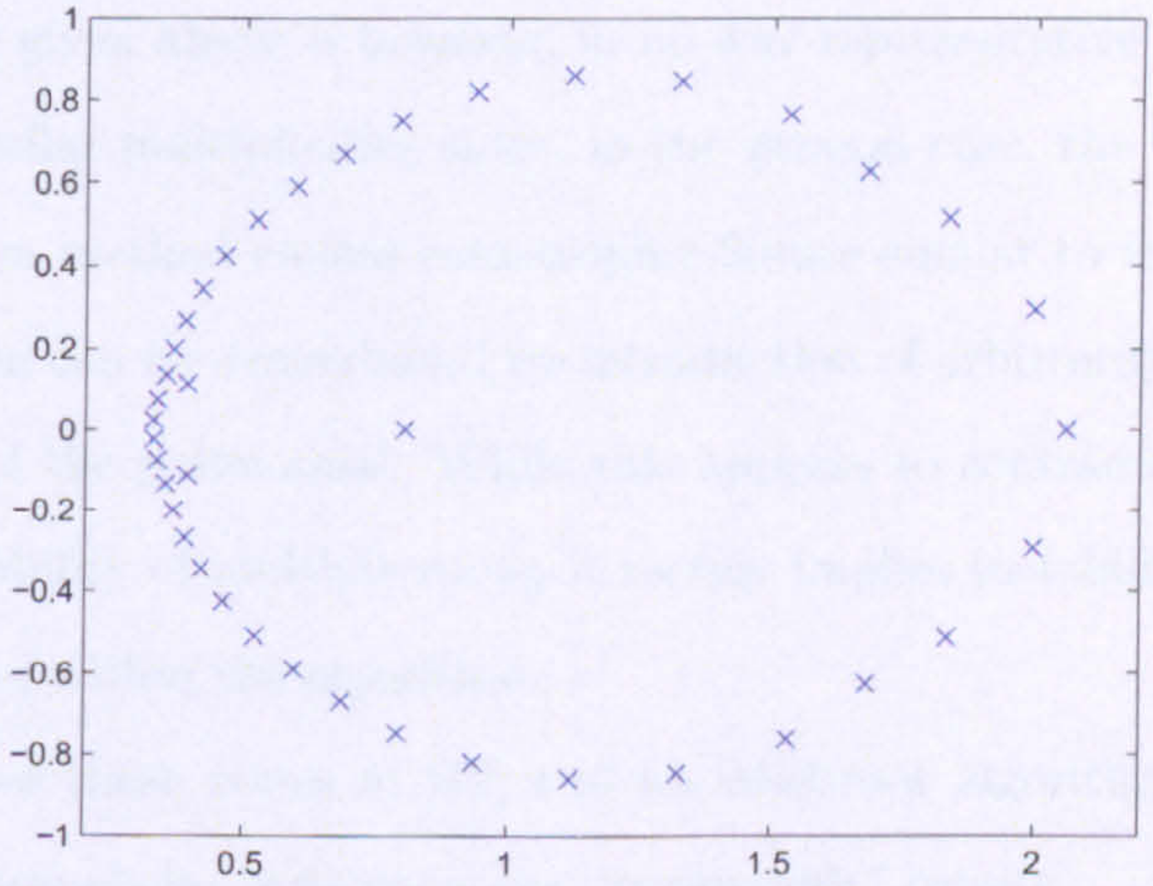

Figure 3.1: The calculated roots of $p(x) = (x - 0.5)^{10}(x - 1)^{30}$

If however, the described factorisation method is used, using the built in MatLab

functions, and then the roots calculated from the resulting polynomials, the correct answer is obtained.

Computed Root	Multiplicity
0.5	10
1	30

It would therefore seem that in spite of theoretical error analysis carried out in [35] and [36], multiple zeros do not necessarily imply ill-conditioning. Rather, it is the method by which they are calculated that accounts for their sensitivity. This idea is supported by Kahan [17] who proved that if the multiplicities are preserved, the roots can actually be well behaved. □

The example given above is however, in no way representative of all polynomials with roots of similar multiplicities since, in the general case, the application of the basic factorisation method causes catastrophic failure similar to that seen in Figure 3.1. This problem can be exacerbated by introduction of arbitrarily small errors into the coefficients of the polynomial. While this appears to contradict the conclusions drawn on the stability of multiple roots, it merely implies instability in one or more of the subroutines within the algorithm.

Zeng addresses these issues in [42] and his Multroot algorithm, which is based upon the procedure above, achieves some *"remarkable"* results.

3.2 Multroot

Multroot is a combination of two algorithms that uses the factorisation method described above and the insight by Kahan to produce a stable general purpose polynomial root-finder. The first stage employs an *approximate GCD* finder to calculate the multiplicity structure and initial root approximations.

Approximate-GCD: An approximate GCD differs from an exact GCD in that instead of containing factors that *exactly* divide two polynomials, it instead contains factors that *nearly* divide the two polynomials. For instance consider the polynomials

$$p(x) = (x-1)^2(x-2)(x-3+\delta) \quad \text{and} \quad q(x) = (x-1)(x-3)^2(x-4).$$

The exact GCD is clearly,

$$d(x) = (x-1),$$

for $\delta \neq 0$. However, if δ is below a pre-defined tolerance then the approximate-GCD is said to be,

$$d(x) = (x-1)(x-3).$$

The approximate GCD is discussed in more detail in the next chapter.

The second stage of the algorithm then transforms the root-finding problem into a regular nonlinear least squares routine which calculates the multiple roots simultaneously on a pejorative manifold.

Pejorative Manifold: Polynomials with one or more multiple zeros form a subset of the space of all polynomials. These subsets are called *pejorative manifolds* [17].

The use of an approximate GCD algorithm in Multroot, is largely responsible for the relative stabilisation of the factorisation method described in the previous section. This is due to its ability to handle errors by searching for a solution to a nearby problem as opposed to the exact GCD implementation used in the above methodology which searches only for *exact* divisors.

Zeng's algorithm for revealing the multiplicity structure consists of four steps.

1. Find the degree k of the approximate GCD, $d(x) = GCD(p(x), p'(x))$.

2. Set up the system,

$$\begin{aligned} p(x) &= d(x)v(x) \\ p^{(1)}(x) &= d(x)w(x), \end{aligned}$$

   where $d(x)$ is monic and $v(x)$ and $w(x)$ are coprime, in accordance with the degree k.

3. Find an initial approximation to $d(x)$, $v(x)$ and $w(x)$ for the GCD system above.

4. Use the Gauss-Newton iteration [3] to refine the GCD triplet $(d(x), v(x), w(x))$.

Steps one and four are of particular interest and essentially encapsulate the research presented in this thesis, albeit for the more general case of calculating $\mathrm{GCD}(p(x), q(x))$.

In step one it is necessary to find the rank of a resultant matrix R (*see Appendix B*). The method used in the algorithm above involves applying a threshold, θ, to the singular values of R such that the computed rank is the number of singular values that are greater than θ. This clearly requires some prior knowledge of the signal to noise ratio[1], Δe, in order to set the threshold. If Δe is not known, or is not *accurately* known, then the returned rank is likely to be inaccurate. Consequently, a data driven rank estimator that does not require manual threshold setting, such as the one described in Chapter 5 would appear to be desirable.

[1]The componentwise signal to noise ratio is used in this thesis, and this is defined as $\max_{i=0,\ \ldots m} \frac{|a_i|}{|\delta a_i|}$, where $f(x) = \sum_{i=0}^{m} a_i x^i$.

The iteration described in step four involves the evaluation of,

$$\begin{bmatrix} d_{j+1} \\ v_{j+1} \\ w_{j+1} \end{bmatrix} = \begin{bmatrix} d_j \\ v_j \\ w_j \end{bmatrix} - J(d_j, v_j, w_j)^{\dagger}_{W} \begin{bmatrix} (e_1^T d_j & -1) \\ (\mathrm{conv}(d_j, v_j) & -p) \\ (\mathrm{conv}(d_j, w_j) & -p^{(1)}) \end{bmatrix}, \tag{3.1}$$

where $\mathrm{conv}(s,t)$ denotes the convolution of the vectors s and t, and $J(d_j, v_j, w_j)^{\dagger}_{W}$ is the weighted pseudo-inverse of the Jacobian,

$$J(d,v,w) = \begin{bmatrix} e_1^T & & \\ C_{m+1}(v) & C_{k+1}(d) & \\ C_{m+1}(w) & & C_k(d) \end{bmatrix},$$

where $e_1 = [\ 1 \quad 0 \quad \cdots \quad 0\ ]^T \in \mathbb{R}^{m+1}$ and C_i is the ith combinatorial matrix. For example $C_k(d)$ is,

$$C_k(d) = \begin{bmatrix} d_0 & & & \\ d_1 & d_0 & & \\ \vdots & d_1 & \ddots & \\ d_k & \vdots & \ddots & d_0 \\ & d_k & \ddots & d_1 \\ & & \ddots & \vdots \\ & & & d_k \end{bmatrix}$$

$k+1$

This system performs remarkably well, displaying both good stability and fast convergence [42].

Alternatives to these steps for the general case of $\mathbf{GCD}(p(x), q(x))$ are proposed in Chapters 5 and 6. In Chapter 5 a novel method for determining the rank of a matrix using maximum likelihood is developed. Whilst Chapter 6 replaces the system in (3.1) with a constrained least squares problem that uses structured matrix methods to refine the GCD triplet obtained from the previous steps, whilst enforcing any relationship between the elements of the constituent matrices.

3.3 Summary

In this chapter it has been shown that repeated GCD factorisations can be used to reduce the problem of calculating the roots of a polynomial with multiple roots to one of solving a sequence of polynomials, all of which have only simple roots.

Two possible refinements to the approximate GCD algorithm have been suggested, the first being a way of calculating the rank of the matrix without prior knowledge of signal to noise ratio in the coefficients, and the second being a novel way of calculating an approximate GCD using structured matrix methods.

The next chapter examines the concept of the approximate GCD and illustrates the three major approaches used in other work.

Chapter 4

The approximate GCD

As was shown in the previous chapters, computing the exact GCD of two polynomials is an ill-posed problem since an arbitrarily small perturbation can cause a dramatic decrease in its degree. Consequently, asking the question:

"Do two polynomials have a non-trivial GCD?",

is of limited use in real world applications and one should instead consider the question:

> *"Are the given polynomials near a pair of polynomials that have a non-constant GCD?"*

Which is quite different from asking "*how near* are the polynomials". The solution to this problem is known as an *approximate GCD* and its solution is the objective of this thesis. In this chapter an overview of the major approaches used in other works is given and the reasoning behind the methods developed in this work explained.

4.1 An overview of the approximate GCD problem

In the literature there are three major approaches to solving the approximate GCD problem. The first involves modifying the classical Euclidean algorithm which, according to Knuth [20]:

> "...is the oldest non-trivial algorithm which has survived to the present day."

Hribernig and Stetter [16] demonstrate how to change the termination criteria in Euclid's algorithm in order to compute a ε-GCD which can be defined as being the *exact* GCD of nearby polynomials within distance ε of the original polynomials. In addition to the successive remainders, computed as part of Euclid's algorithm, they compute multiplier polynomials which express the original two polynomials as linear combinations of the last two remainders. The algorithm terminates when the norm of the product of these polynomials is less than the tolerance, ε. This algorithm is very efficient since its computational cost is almost the same as that of computing the classical generalised Euclidean algorithm i.e. the extension of the classical Euclidean algorithm from integers to polynomials. It does not, however, guarantee that the calculated GCD will be of maximum degree within a given error tolerance. This is the goal of the next approach.

Using the singular value decomposition (SVD) [11] it is possible to determine the degree of the GCD of two polynomials from the rank of the corresponding resultant matrix. More precisely, the SVD allows the determination of the rank of a *nearby* resultant matrix and consequently the degree of a nearby or approximate GCD. In fact the ith singular value, σ_i, gives the 2-norm distance to the nearest matrix with rank strictly less than i [11].

Typically the Sylvester resultant matrix is used for this, due to its simple structure which makes it easy to manipulate compared to the Bezoutian resultant matrix (*see Appendix B*). However, it is possible to use any of the other resultant matrices[1].

The Sylvester resultant matrix: The Sylvester matrix $S(p(x), q(x)) \in \mathbb{R}^{(m+n)\times(m+n)}$ is equal to,

$$S(p(x), q(x)) = \begin{bmatrix} a_0 & & & & b_0 & & & \\ a_1 & a_0 & & & b_1 & b_0 & & \\ \vdots & a_1 & \ddots & & \vdots & b_1 & \ddots & \\ a_{m-1} & \vdots & \ddots & a_0 & b_{n-1} & \vdots & \ddots & b_0 \\ a_m & a_{m-1} & \ddots & a_1 & b_n & b_{n-1} & \ddots & b_1 \\ & a_m & \ddots & \vdots & & b_n & \ddots & \vdots \\ & & \ddots & a_{m-1} & & & \ddots & b_{n-1} \\ & & & a_m & & & & b_n \end{bmatrix},$$

where the coefficients $\{a_i\}_{i=0}^{m}$ of $p(x)$ occupy the first n columns, and the coefficients $\{b_i\}_{i=0}^{n}$ of $q(x)$ occupy the last m columns.

It is shown in [7] that if $\sigma_{r+1} \leq \varepsilon \leq \sigma_r$, where ε is the noise threshold, are a pair of singular values bracketing ε and there is a sufficiently large gap between σ_r and σ_{r+1}, then the maximum degree of any GCD within the space defined by ε is guaranteed to be r. If the gap is not big enough then only an *upper-bound* on the GCD can be obtained.

Once r has been computed, the coefficients of the GCD may be calculated via a triangular decomposition of the resultant matrix, such as QR or LU [11]. The simplest implementation is given in the following example.

[1] Resultant matrices are discussed in more detail in Chapter 5.

Example 4.1. Consider the polynomials,

$$\begin{aligned} p(x) &= (x-1)(x-2)(x-3)(x-4) = x^4 - 10x^3 + 35x^2 - 50x + 24 \\ q(x) &= (x-2)(x-4)(x-5) = x^3 - 11x^2 + 38x - 40, \end{aligned}$$

and the corresponding Sylvester resultant matrix,

$$S(p(x), q(x)) = \begin{bmatrix} 1 & & & 1 & & & \\ -10 & 1 & & -11 & 1 & & \\ -35 & -10 & 1 & 38 & -11 & 1 & \\ -50 & 35 & -10 & -40 & 38 & -11 & 1 \\ 24 & -50 & 35 & & -40 & 38 & -11 \\ & 24 & -50 & & & -40 & 38 \\ & & 24 & & & & -40 \end{bmatrix},$$

which is of rank 5. If a QR decomposition is performed on $S(p(x), q(x))^T$ then the coefficients of the GCD are obtained from the 5th row of the upper triangular matrix R,

$$R = \begin{bmatrix} -0.2681 & 2.8150 & -9.7856 & 12.0644 & -3.2172 & 0 & 0 \\ & -0.2997 & 2.6977 & -8.1530 & 9.3520 & -2.8775 & 0 \\ & & -0.9479 & 6.1422 & -10.4266 & 4.3223 & -0.9100 \\ & & & 1.2233 & -7.7969 & 12.5291 & -3.6570 \\ & & & & 1 & -6 & 8 \\ & & & & & 0 & 0 \\ & & & & & & 0 \end{bmatrix},$$

and hence, as shown in [2], [4],

$$d(x) = \mathrm{GCD}(p(x), q(x)) = x^2 - 6x + 8.$$

□

The final collection of approaches involves posing the problem as an optimisation problem with the aim of finding a GCD of specified degree having minimum distance from the original polynomials. The distance between two polynomials, unless defined otherwise, is equal to the Euclidian distance (2-norm) between the vectors of their coefficients. Kamarkar and Lakshman [19] provide a method for calculating the minimum perturbation of any pair of polynomials which would have a non-trivial GCD. They then show that the problem of computing any minimum norm maximum degree GCD can be reduced to known algorithms for quantifier elimination. This gives a guaranteed optimal solution to the original problem but computing time is exponential in the degree of the GCD and is consequently impractical for pairs of polynomials containing many common factors.

Chin and Corless [5] present an alternative optimisation method which assumes that the degree of the GCD along with initial approximations of its coefficients are available as a starting point for the algorithm. The problem formulation in this method is used as the basis for the methods developed in Chapter 6.

Other examples of approximate GCD algorithms include works by Noda and Sasaki [26], Corless et al [6], Pan [27], Karcanias and Mitrouli [18] and Zarowski [4].

To summarise, there exist three major approaches to solving the approximate GCD problem which are defined by their requirements of the solution. These are:

1. **Nearness:** The approximate GCD is the exact GCD of nearby pair of polynomials.

2. **Max-degree:** The approximate GCD has the highest degree among those polynomials satisfying nearness.

3. **Min-distance:** The approximate GCD minimises the 2-norm distance between the chosen nearby polynomials and the given polynomials.

Whilst all of these criteria are valid, their use in defining an approximate GCD should be treated with caution. Indeed, the "correct" definition of an approximate GCD may vary from application to application. There is even the possibility that, without prior information as to the magnitude of the signal to noise ratio (ratio of signal power to noise power) or the structure of the desired solution, a reasonable data driven criteria cannot be decided upon.

Consider, for example, two theoretically exact polynomials. If they contain no errors, the solution to the GCD problem is unique. If, however, the polynomials are perturbed such that the signal to noise ratio is Δe, a region in which the exact data lies can be defined. This region can be considered a hypersphere whose centre is at the given inexact data with radius proportional to the signal to noise ratio. Without precise knowledge of the noise there is no way of knowing the degree of the theoretically exact GCD, let alone the exact values of the coefficients, since a solution with a degree greater than that of the theoretically exact GCD may lie in the space defined by Δe. Hence, there now exists a whole family of GCDs, of differing degrees, all of which are exact solutions to nearby polynomials and equally valid. Consequently, unless there is a prior requirement on the solution to have either

maximum degree or be a minimum distance from the given data, these criteria alone should not define the solution. Indeed, if a reasonable estimate of Δc is unknown then the nearness requirement is also hard to justify. In this work therefore, an approximate GCD is developed such that different constraints may be added and removed, depending on the application requirements, and attempts have been made to make this a truly data driven black box technique. The methods developed in the following two chapters can be used either together, or in conjunction with existing works allowing the user the flexibility to obtain an approximate GCD whose definition is best suited to their problem.

4.2 Summary

In this chapter the concept of an *approximate GCD* has been discussed along with the major approaches to this problem. It has also been stated that caution should be exercised when choosing the criteria by which an approximate GCD is to be defined.

In the next chapter a statistically based matrix rank estimator is developed which computes the *most likely* rank of a matrix, based upon an underlying probability distribution.

Chapter 5

Calculating the rank of a resultant matrix

In this chapter a novel method for calculating the rank of a resultant matrix using maximum likelihood is developed.

Resultant: The resultant of two monic polynomials $p(x)$ and $q(y)$ is defined as,

$$R(p(x), q(y)) = \prod_{x_i} \prod_{y_j} (x_i - y_j), \quad p(x_i) = 0 \quad \text{and} \quad q(y_j) = 0,$$

and is also given by the determinant of a corresponding resultant matrix. If the determinant of the resultant matrix is 0 then the degree of $\mathrm{GCD}(p(x), q(x)) \geq 1$ and $p(x)$ and $q(x)$ consequently have common roots.

There are several types of resultant matrices, including the Sylvester, Bezoutian and companion matrices[1], and being able to accurately calculate their rank is highly desirable since any rank deficiency indicates the presence of common factors between its constituent polynomials. More significantly, its rank deficiency, *order of matrix* − *rank*, gives the *number* of common factors and hence the degree of the GCD.

[1]See Appendix B for details on the construction and structure of the Bezoutian matrix.

Example 5.1. Consider the polynomial,

$$p(x) = (x-1)^2(x-2)^2(x-3), \tag{5.1}$$

and its derivative $p^{(1)}(x)$. Clearly they share 2 common factors, $(x-1)$ and $(x-2)$, and therefore the corresponding resultant matrix has rank *order of matrix* $-$ 2 and $\deg(\mathrm{GCD}(p(x), p^{(1)}(x))) = 2$. □

This property, therefore, not only provides a means for testing for the presence of common factors, but also provides the degree of the GCD of the two polynomials.

The problem of rank determination appears in virtually every field of numerical and scientific analysis and has been tackled by [21], [22] and many others in a variety of different ways and contexts.

One problem with the vast majority of methods previously considered is that they require a threshold to be manually set in order to determine the index of the smallest singular value. If the magnitude of the noise is unknown, then setting this value may become problematic.

Example 5.2. Consider the polynomial $p(x)$, from Example 5.1, and introduce a signal to noise ratio of the order 10^8 to the coefficients $\{a_i\}_{i=0}^n$, such that,

$$\tilde{a}_i = a_i + \left(\frac{a_i r_i}{10^8}\right). \tag{5.2}$$

where r_i is a random variable uniformly distributed in the interval $[-1, 1]$, resulting in $p(x) \to \tilde{p}(x)$. The Sylvester matrix, $S(\tilde{p}(x), \tilde{p^{(1)}}(x)) \in \mathbb{R}^{(9\times 9)}$, is now constructed from the perturbed polynomial $\tilde{p}(x)$ and its derivative, and Matlab's `rank()` function is called. If the default tolerance is used, then the calculated rank is 9. If the tolerance

is manually set to 10^{-9} a rank of 8 is obtained. The correct rank of 7 is only obtained when the tolerance is set to 10^{-8}. Clearly, if inaccurate information on the signal to noise ratio is provided, setting this threshold may be problematic. □

Zarowski [41] suggests the use of the minimum description length (MDL) criterion to determine the rank of a matrix. This automatically sets a threshold and does not relate the characteristics of the data that gives rise to the singular values to the statistical distribution of the values themselves. This enables the automatic threshold setting to be performed on data from an arbitrary source.

MDL: The *Minimum Description Length* (MDL) criterion was first proposed by Rissnanen [29] and is a formalisation of Occam's Razor in which the "best" model among a given collection of models is the one that yields the shortest description of the data *and* the model itself. For each model in the collection, the length of description of the data is counted as the codelength of encoding the data using that model in binary digits (bits). The length of description of a model is the codelength of specifying that model, e.g. the number of parameters and their values if it is a parametric model. MDL therefore attempts to reconcile conflicting demands between signal compression and noise suppression. For a tutorial on MDL, see [12] and [31].

The use of MDL to calculate of the rank of a noisy matrix is a novel and insightful advance in an area of mathematics that has primarily relied on linear algebra to solve this problem, leaving statistical methods such as [21] in the minority. The concept appears to be drawn from Saito [31] where an algorithm for simultaneous signal compression and noise suppression is developed for applications such as image reconstruction and signal transmission where there is a requirement for both good model fidelity *and* low bandwidth usage. The problem of calculating the rank of a noisy matrix however, requires only noise suppression and therefore the term regarding signal compression is redundant. Consequently it is possible to use the initial model proposed by Zarowski and obtain a solution using the principle of *Maximum*

Likelihood Estimation (MLE).

MLE: Loosely defined, for a fixed set of data and underlying probability model, maximum likelihood picks the values of the model parameters that make the data "more likely" than any other values of the parameters would make them.

See `http://statgen.iop.kcl.ac.uk/bgim/mle/sslike_1.html` for an excellent tutorial on MLE.

This is due to the fact that the principle of MLE requires the maximisation of a function of the form $\ln p(x|\theta)$, whereas the principle of MDL requires the minimisation of an expression that includes the function $-\log p(x|\theta)$. Although these two operations are identical (apart from the changes in the sign and base of the logarithm), the methods have fundamentally different interpretations because optimality in the principle of MLE is attained when the parameter values maximise the probability of the observed data, but optimality in the principle of MDL is attained when the conflicting requirements of data compression and model fidelity are satisfied, where optimality is quantified by code length.

In the following section a method of calculating rank using the principle of maximum likelihood is defined. The initial likelihood expression that is developed differs by only one term from the MDL criterion function in [41] but it will then be shown that the expression can be simplified further removing a large proportion of the computational intensity.

5.1 Calculating the rank of a noisy matrix using MLE

Consider a matrix $A \in \mathbb{R}^{m \times n}$, with rank $r \leq p = \min(m, n)$ and singular values,

$$\{\sigma_j\}_{j=1}^{p} = \begin{cases} \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0 \\ \sigma_{r+1} = \sigma_{r+2} \cdots = \sigma_p \end{cases}.$$

In practice, only estimates of the singular values,

$$\tilde{\sigma}_j = \sigma_j + e_j, \tag{5.3}$$

are known, due to perturbations in the elements of A and roundoff introduced while calculating σ_j.

If the errors, $\{e_j\}_{j=1}^{p}$, are treated as statistically independent random variables, then it is reasonable to regard them as having Gaussian and exponential probability density functions (pdf's),

$$p(e_j) = \begin{cases} \frac{1}{\sqrt{2\pi s^2}} \exp\left(-\frac{e_j}{2s^2}\right), & j = 1, 2, \ldots, r \\ \alpha \exp(-\alpha e_j) & j = r+1, \ldots, p \end{cases}, \tag{5.4}$$

where $s, \alpha > 0$ are the variance and decay coefficient respectively. The pdf's, while being chosen somewhat arbitrarily, provide a compromise between a physically accurate and a mathematically simple model and are intended to merely capture the general trends and uncertainties in the singular values well enough to determine r rather than accurately model the theoretical distribution of the errors in an attempt

to reconstruct data for a specific problem. This is important as it allows the algorithm to be applicable on data drawn from an arbitrary source, whilst allowing scope to refine the pdf's for problems that contain highly correlated perturbations.

From (5.4), the joint pdf for the random variables e_j is,

$$\frac{\alpha^{p-r}}{(2\pi s^2)^{r/2}} \exp\left(-\frac{1}{2s^2}\sum_{j=1}^{r} e_j^2 - \alpha \sum_{j=r+1}^{p} \tilde{\sigma}_j\right), \tag{5.5}$$

and the substitution of $e_j = \tilde{\sigma}_j - \sigma_j$, from (5.3), into this expression yields the probability density function for the estimates $\tilde{\sigma}_j$ of the exact singular values σ_j,

$$\begin{aligned} p_\sigma &= p(\{\tilde{\sigma}\}|a, s^2, \alpha, k, r) \\ &= \frac{\alpha^{p-r}}{(2\pi s^2)^{r/2}} \exp\left(-\frac{1}{2s^2}\sum_{j=1}^{r} (\tilde{\sigma}_j - \sigma_j)^2 - \alpha \sum_{j=r+1}^{p} \tilde{\sigma}_j\right). \end{aligned} \tag{5.6}$$

The ML estimate for α, $\hat{\alpha}$, is easily obtained by setting the partial derivative of p_σ, with respect to α to zero, which yields,

$$\hat{\alpha} = \frac{p-r}{\sum_{j=r+1}^{p} \tilde{\sigma}_j}, \tag{5.7}$$

and similarly the ML estimate of s^2, $\hat{s}^2$, satisfies,

$$\sum_{j=1}^{r} (\tilde{\sigma}_j - \sigma_j)^2 = r\hat{s}^2. \tag{5.8}$$

From (5.6) the log likelihood expression of the data $\{\tilde{\sigma}_j\}$ is,

$$\begin{aligned} L_{\tilde{\sigma}_j} &= -\log p_{\tilde{\sigma}}(\{\tilde{\sigma}\}|\hat{\alpha}, \hat{s}^2, \hat{a}, k, r) \\ &= (p-r)\log\hat{\alpha} - \frac{r}{2}\log(2\pi\hat{s}^2) \\ &\quad - \log\exp\left(-\frac{1}{2\hat{s}^2}\sum_{j=1}^{r}(\tilde{\sigma}_j - \sigma_j)^2 - \hat{\alpha}\sum_{i=r+1}^{p}\tilde{\sigma}_i\right). \end{aligned} \tag{5.9}$$

Substituting (5.7) and (5.8) into (5.9) yields,

$$\begin{aligned} L_{\tilde{\sigma}} &= (p-r)\log\left(\frac{p-r}{\sum_{j=r+1}^{p}\tilde{\sigma}_j}\right) - \frac{r}{2}\log\left(\frac{2\pi}{r}\sum_{j=1}^{r}(\tilde{\sigma}_j - \sigma_j)^2\right) \\ &\quad - \left(p - \frac{r}{2}\right). \end{aligned} \tag{5.10}$$

In [41] it is assumed that the theoretically exact non-zero singular values $\{\sigma_j\}$ can be modelled with a polynomial such that,

$$\sigma_j = \sum_{i=0}^{d} a_i j^i, \tag{5.11}$$

where $p \geq r \geq d+1$, with $d \geq 0$.

However, the instability caused by the Hankel system [11], resulting from obtaining the MLE of the coefficients $\{a_i\}$ in (5.11) motivates Zarowski to replace (5.11) with a finite series of Gram polynomials [14], which are orthogonal and therefore provide numerical stability. Gram polynomials are useful for approximation over a discrete point set, for properties and recursive relationships, see [41]. Whilst this improves the stability of the algorithm, using a polynomial to model the singular values seems like a less than optimal choice since their profile contains no turning points and bears

a greater resemblance to a decaying exponential (See Figure 5.1).

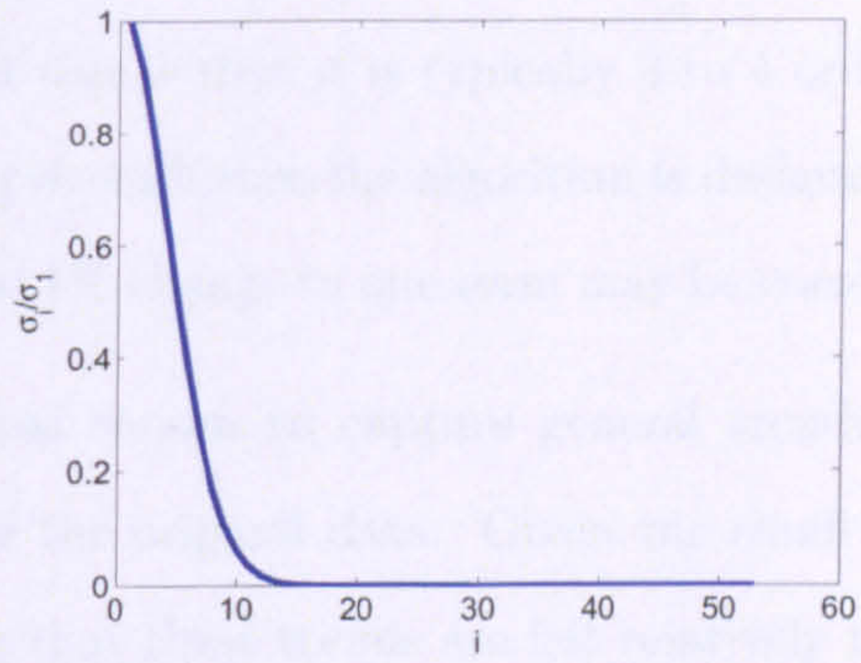

Figure 5.1: A typical profile of the singular values of a resultant matrix.

The choice, therefore, appears to be due to convenience rather than it being a good model. Additionally it requires (5.10) to be calculated r times, once for every value of d, since $d = 0, \ldots, r-1$, making the algorithm slow and cumbersome for all but the most trivial examples.

However, experimental results have indicated that the use of a polynomial model is not actually necessary. Therefore, σ_j can be omitted and the likelihood function can be expressed as,

$$L_{\tilde{\sigma}} = (p-r)\log\left(\frac{p-r}{\sum_{j=r+1}^{p}\tilde{\sigma}_j}\right) - \frac{r}{2}\log\left(\frac{2\pi}{r}\sum_{j=1}^{r}\tilde{\sigma}_j^2\right) - \left(p-\frac{r}{2}\right). \tag{5.12}$$

Identical results, with regard to the calculated rank, have been observed over a population of several thousand random polynomials when using this expression instead of the one proposed in [41]. The reasons for this somewhat unexpected result are believed to be a combination of the following.

- Fitting a polynomial to the data $\{\sigma_j\}$ is a poor model.
- In order for $L_{\tilde{\sigma}}$ to be maximised it is necessary to minimise the term describing σ_j. The result of this is that it is typically 3 to 4 orders of magnitude smaller than the term for $\tilde{\sigma}_j$ and since the algorithm is designed to work in the presence of noise a 0.1% to 1% change in one term may be considered as relatively small.
- The pdf (5.5) was chosen to capture general trends rather than allowing a reconstruction of the original data. Given the small change in the term it is possible to argue that these trends are left relatively unaltered.
- The output of the algorithm is discrete, i.e. it returns an integer value indicating the calculated rank meaning that the computed likelihood can vary over a certain interval before the calculated rank changes from r to $r \pm 1$.

Results obtained from the expression (5.12) are now detailed.

5.2 Results

This section contains several examples that illustrate the method of calculating the rank of a matrix using the likelihood function (5.12). This method will be referred to as `MLrank()`.

In order to assess its performance, it is tested against 2 other procedures for calculating numerical rank. They are Matlab's built in rank function, `rank()`, and `Approxirank()`, a function from Zeng's Apalab toolbox [43]. These algorithms are both tolerance based methods, and comparisons will be made between them and `MLrank()` with their default tolerances, and a user defined tolerance, θ, which is based on the estimated signal to noise ratio, Δe. The testing procedure is now described.

A test set of 1000 pairs of inexact polynomials are generated with each pair having an almost non-constant common divisor. This is done as described in Algorithm 5.1.

Algorithm 5.1: Polynomial Test Set 1

1. Generate three polynomials,

$$\begin{aligned} d(x) &= \prod_{i=0}^{k}(x-\alpha_i), \\ u(x) &= \prod_{i=0}^{h}(x-\beta_i), \\ v(x) &= \prod_{i=0}^{l}(x-\gamma_i), \end{aligned}$$

where k, h and l are randomly generated integers on the intervals $[3, 25]$, $[3, 10]$, $[3, 10]$ respectively.

2. Form the polynomials $p(x)$ and $q(x)$ such that,

$$p(x) = d(x)u(x) \quad \text{and} \quad q(x) = d(x)v(x)$$

3. Perturb the coefficients $\{p_i\}$ and $\{q_j\}$ of $p(x)$ and $q(x)$ in the componentwise sense, such that,

$$\{p_i\} \rightarrow \{\tilde{p}_i\} = \left\{p_i + \frac{p_i r_i}{\Delta e}\right\},$$

where r_i is a uniformly distributed random variable and Δe is the signal to noise ratio. Hence $p(x) \rightarrow \tilde{p}(x)$ and $q(x) \rightarrow \tilde{q}(x)$.

For each pair of inexact polynomials the corresponding Bezoutian matrix[2] $\tilde{B}$, is constructed. The rank of this matrix is then computed by the three algorithms and the difference between this and the theoretically exact rank is calculated.

Initially, it is assumed that a good estimate of Δe is known and the tolerance θ is,

$$\theta = \frac{\|\tilde{B}\|}{\Delta e}.$$

The tests are then repeated on the same set of polynomials, first assuming that the information regarding the magnitude of the noise is inaccurate, and then that there is no knowledge of the noise level, in which case the default tolerances of `rank()` and `ApproxiRank()` are used.

The algorithm has been tested on both the Sylvester and Bezoutian matrices, although only examples for the Bezoutian are presented here since, in general, it gives superior results. In addition to this, it is unaffected by the relative scaling of its constituent polynomials, which is not the case with the Sylvester matrix. The effect of scaling is now discussed.

5.2.1 The effect of α on the rank of a Sylvester Resultant matrix

The linear structure of the Sylvester matrix, $S(p(x), q(x))$, makes it convenient for computations. In particular the partitioning of the columns allows the scaling of one of the polynomials with respect to the other, by an arbitrary scale factor α, producing the matrix $S(p(x), \alpha q(x))$.

It would seem intuitive that this scaling should have no effect on the rank of the

[2]See Appendix B for details.

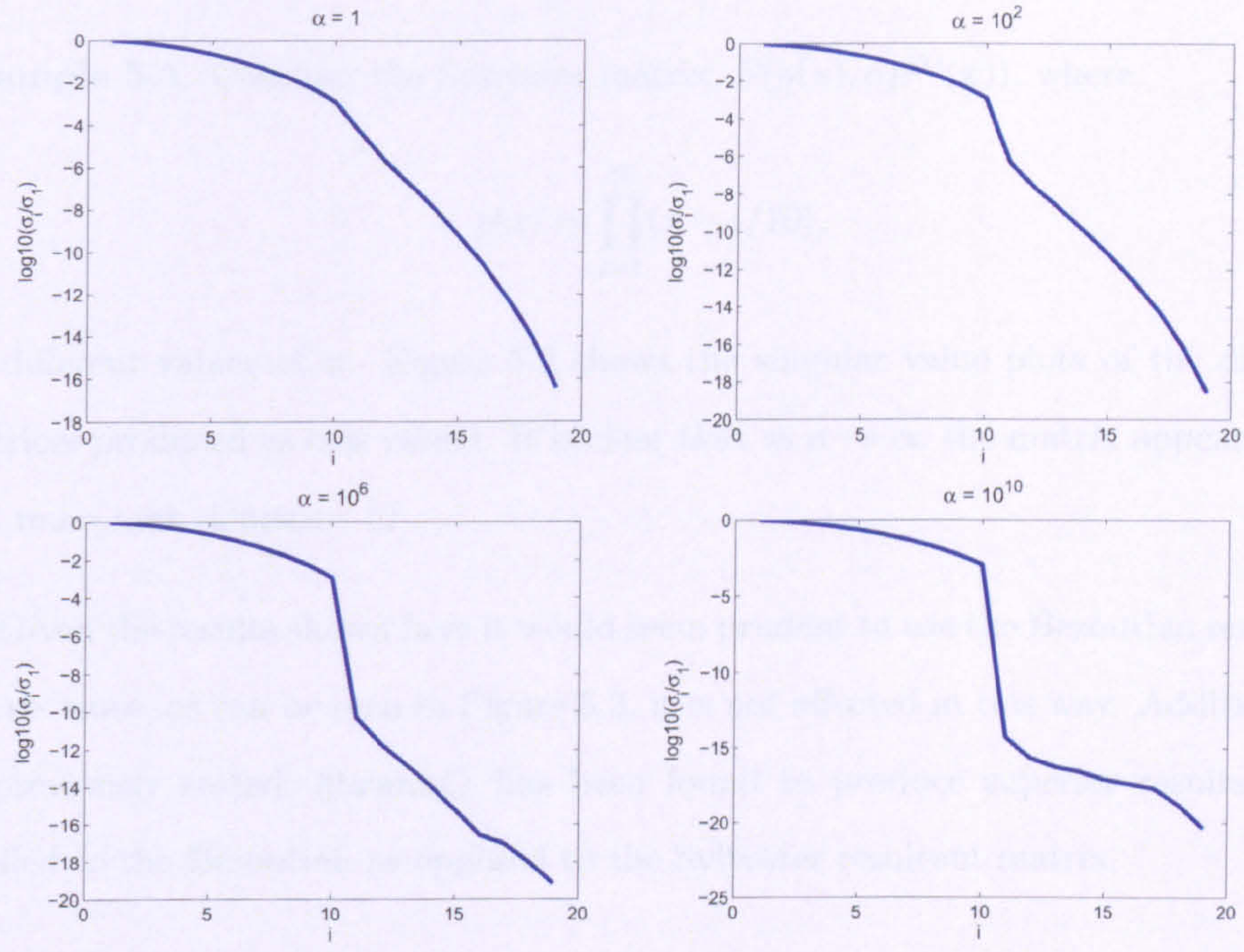

Figure 5.2: The effect of α on the singular values of $S(p(x), \alpha p'(x))$.

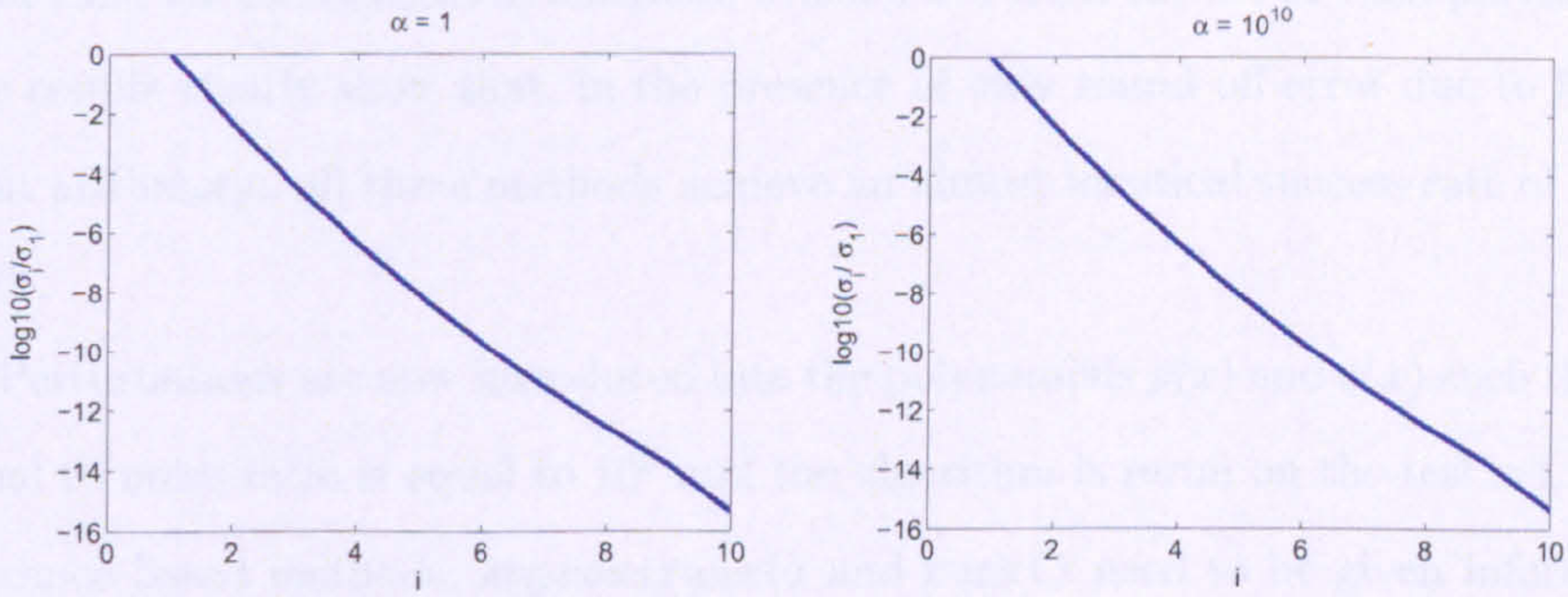

Figure 5.3: The effect of α on the singular values of $B(p(x), \alpha p'(x))$.

matrix. However, it can be clearly demonstrated that this is not the case.

Example 5.3. Consider the Sylvester matrix, $S(p(x), \alpha p^{(1)}(x))$, where,

$$p(x) = \prod_{i=1}^{10}(x - i/10), \tag{5.13}$$

for different values of α. Figure 5.2 shows the singular value plots of the different matrices produced as α is varied. It is clear that as $\alpha \to \infty$ the matrix appears more and more rank deficient. □

Given the results shown here it would seem prudent to use the Bezoutian resultant matrix since, as can be seen in Figure 5.3, it is not affected in this way. Additionally, as previously stated, `MLrank()` has been found to produce superior results when applied to the Bezoutian as opposed to the Sylvester resultant matrix.

5.2.2 Examples

Figure 5.4 shows the difference between the computed rank and the theoretically exact rank for the Bezoutian matrices, constructed from the set of 1000 polynomials. The results clearly show that, in the presence of only round off error due to floating point arithmetic, all three methods achieve an almost identical success rate of almost 100%.

Perturbations are now introduced into the polynomials $p(x)$ and $q(x)$ such that the signal to noise ratio is equal to 10^8 and the algorithm is rerun on the test set. Being tolerance based methods, `Approxirank()` and `rank()` need to be given information about the magnitude of Δe. Since in many practical applications Δe will need to be estimated, it is important to see the effect of the accuracy of the estimate on

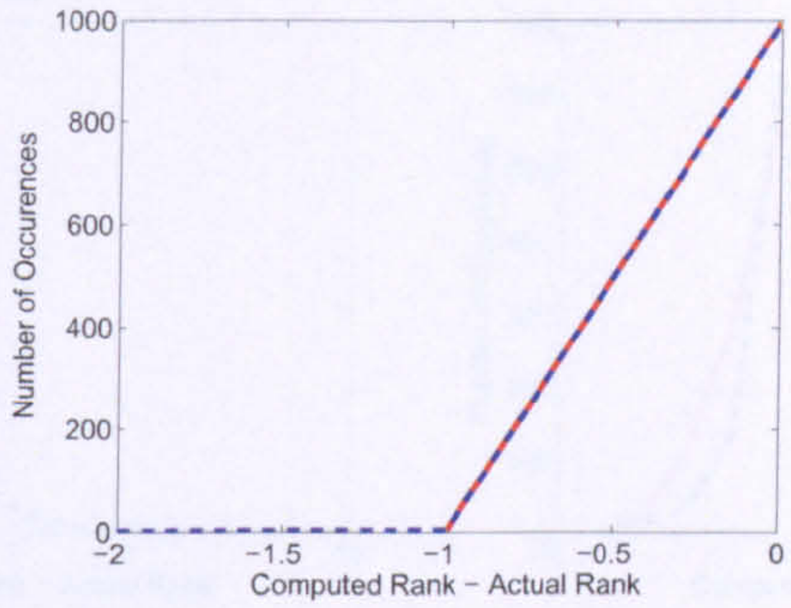

Figure 5.4: The number of occurrences of the error computed rank-actual rank for 1000 random polynomials in the absence of noise. `MLrank()` = - -; `rank()` and `ApproxiRank()` = —.

the accuracy of the calculated rank. This will also highlight the benefits of `MLrank()` since it doesn't require the user to provide any information about Δe. Figures 5.5 (a), (b) and (c) show the effect of the accuracy of the tolerance, θ, on the computed rank. In (a) the level of noise is overestimated with $\theta = \frac{\|B\|}{10^7}$; in (b) the correct magnitude of Δe is given and $\theta = \frac{\|B\|}{10^8}$; whilst in (c) the noise is underestimated and hence $\frac{\|B\|}{10^9}$.

It is clear from these three graphs that obtaining a good estimate for Δe is vitally important in order for the correct rank of a matrix to be computed. In particular, Figure 5.5 (a) shows that underestimating Δe can cause a significant decrease in the performance of both `rank()` and `ApproxiRank()`. `MLrank()` is of course unaffected by these inaccuracies as it does not rely on any threshold to be set.

In order to emphasise the performance of `MLrank()` over the other methods in the absence of any information as to the magnitude of Δe, the test set has been re-run with only the default tolerances being used for `rank()` and `ApproxiRank()`. The results are shown in Figures 5.6 (a) and (b). Note that this is the only example where

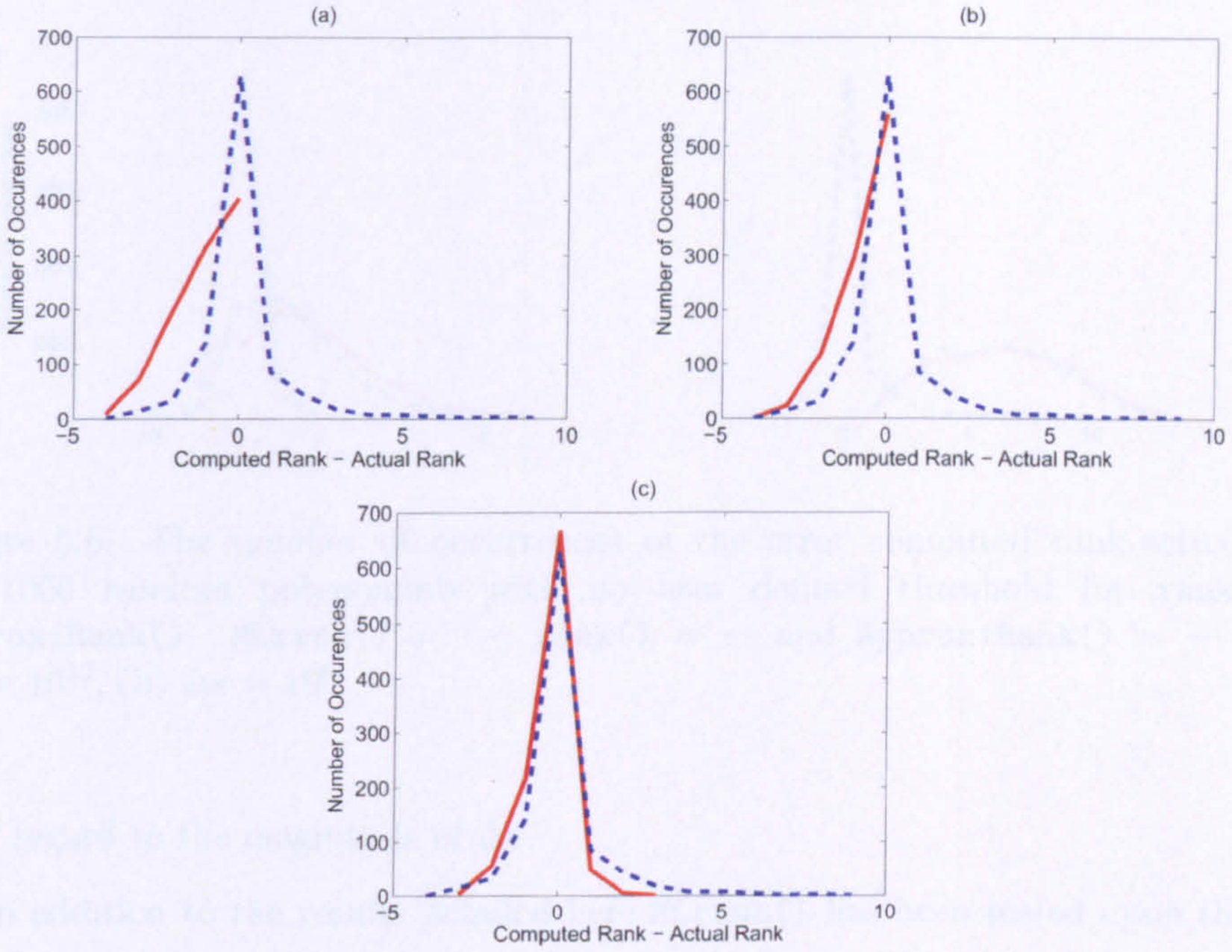

Figure 5.5: The number of occurrences of the error computed rank-actual rank for 1000 random polynomials with $\Delta e = 10^8$. `MLrank()` = - -; `rank()` and `ApproxiRank()` = —. (a) $\theta = \frac{\|\tilde{B}\|}{10^7}$, (b) $\theta = \frac{\|\tilde{B}\|}{10^8}$, (c) $\theta = \frac{\|\tilde{B}\|}{10^9}$.

the algorithms are on a "level playing field" since it is only here that no information as to the magnitude of Δe has been provided. In both (a) and (b) `MLrank()` dramatically outperforms the other two. Indeed, for $\Delta e = 10^8$ neither `rank()` or `Approxirank()` calculate a single rank in the test set correctly whereas `MLrank()` achieves a success rate of over 60%.

From the examples that have been shown it seems clear that `MLrank()` offers a good, and in many cases superior, method for calculating the rank of a resultant matrix. This is particularly true for the case where little or no information is available

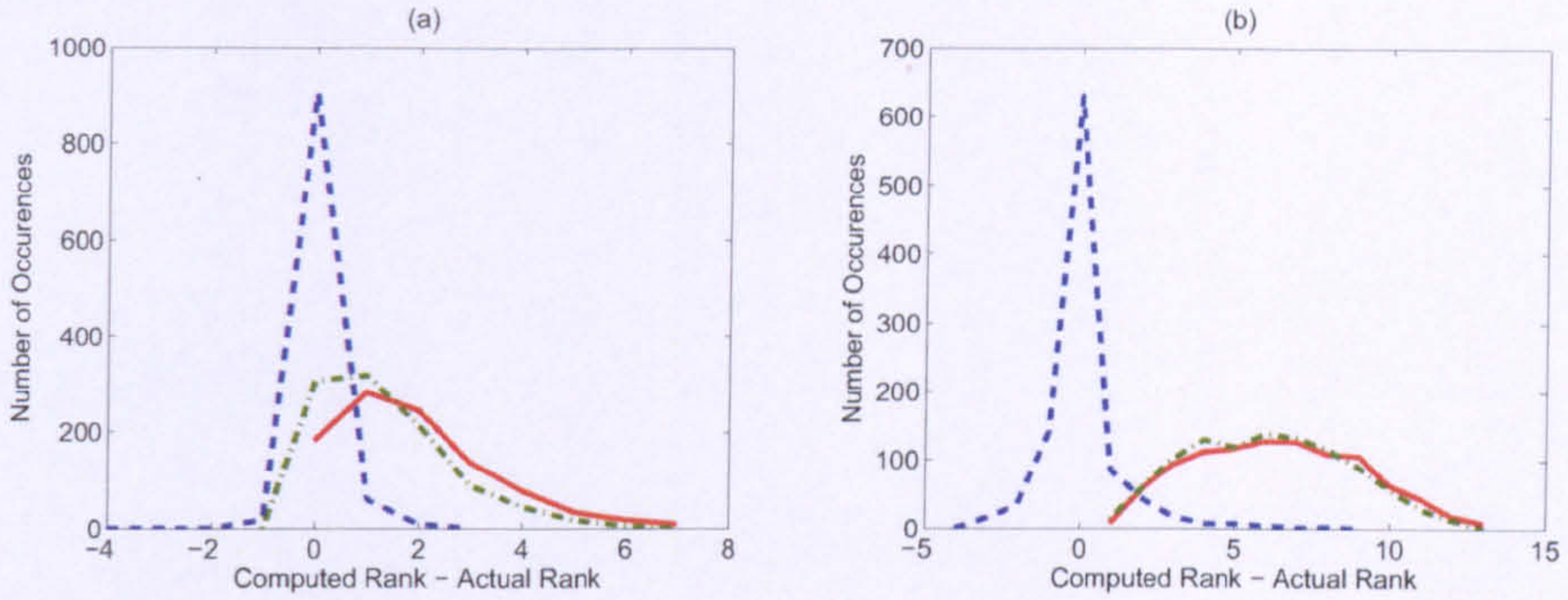

Figure 5.6: The number of occurrences of the error computed rank-actual rank for 1000 random polynomials with no user defined threshold for `rank()` or `ApproxiRank()`. `MLrank()` = - -; `rank()` = — and `ApproxiRank()` = $-\dot{-}$. (a) $\Delta e = 10^{12}$, (b) $\Delta e = 10^{8}$.

with regard to the magnitude of Δe.

In addition to the results detailed here `MLrank()` has been tested upon the case where $q(x) = p^{(1)}(x)$ and hence $B(p(x), p^{(1)}(x))$. These can be found in [38] and a very similar level of success was observed.

5.3 Summary

In this chapter a rank determining method has been developed based upon the principle of MLE and tested against two other rank finding algorithms. In all examples `MLrank()` either performs as well as, or better than, `rank()` and `ApproxiRank()` even though it is given no information as to the magnitude of the signal-to-noise ratio. Suggestions for future work are detailed in Chapter 9.

In the next chapter an approximate GCD finder is developed using structured matrix methods in order that it might be combined with `MLrank()` which is done in

Chapter 7.

Chapter 6

STLN for the approximate GCD

6.1 Introduction

The method described in this chapter uses the problem formulation from [5] and *structured total least norm* (STLN) [30] to obtain an approximate GCD. Initially the problem is formulated in the standard total least squares (TLS) sense [11]. However, this is quickly transformed into a least squares equality problem (LSE) [3] which is then solved using STLN.

Before these issues are discussed however, it is first necessary to introduce a method of solving the LSE problem which can be solved iteratively via several different methods, including the method of weights and the QR decomposition [11]. In this case however, the method of Lagrange multipliers has been chosen since it allows quick and easy development and doesn't require a heuristic weight parameter. It is in no way suggested that this is the best or most suitable method available, and has been chosen for convenience rather than any numerical superiority it may or may not possess over other available methods.

6.2 Solving the least squares equality problem using Lagrange multipliers

The least squares equality (LSE) problem is defined as the determination of a vector $y \in \mathbb{R}^p$ such that a quadratic function is minimised subject to an equality constraint,

$$\min_{Cy=g} \|Gy - f\|, \tag{6.1}$$

where $C \in \mathbb{R}^{m_1 \times p}$, $G \in \mathbb{R}^{m_2 \times p}$, $g \in \mathbb{R}^{m_1}$, $f \in \mathbb{R}^{m_2}$ and $m_1 \leq p$. It is assumed that (i) rank $C = m_1$, which guarantees the constraint is consistent, and (ii) that,

$$N(G) \cap N(C) = \emptyset \Leftrightarrow \text{rank} \begin{bmatrix} G \\ C \end{bmatrix} = p,$$

where $N(X)$ denotes the null space of X. This guarantees that the LSE problem has a unique solution, and implies that $p \leq m_1 + m_2$, from which it follows that the constraints on m_1, m_2 and p are,

$$m_1 \leq p \leq m_3, \qquad m_3 = m_1 + m_2.$$

The constraint $Cy = g$ leaves $p - m_1$ degrees of freedom of y because $m_1 \leq p$. Since the objective function $\|Gy - f\|$ has m_2 equations, it follows that the LSE problem is over determined if $m_2 \geq p - m_1$.

If $\lambda_1 \in \mathbb{R}^{m_1}$ and $\lambda_2 \in \mathbb{R}^{m_2}$ are vectors of Lagrange multipliers, then the LSE

problem requires the minimisation of the function,

$$\min \frac{1}{2}\|r\|^2,$$

subject to,

$$Gy + r = f \quad \text{and} \quad Cy = g.$$

In order to solve this it is necessary to construct the function $h(r, y, \lambda_1, \lambda_2)$,

$$h(r, y, \lambda_1, \lambda_2) = \frac{1}{2}\|r\|^2 - \lambda_1^T(Cy - g) - \lambda_2^T(Gy + r - f).$$

Differentiation of $h(r, y, \lambda_1, \lambda_2)$ with respect to r, y, λ_1 and λ_2 yields,

$$\begin{aligned} r - \lambda_2 &= 0 \\ G^T\lambda_2 + C^T\lambda_1 &= 0 \\ Gy + r - f &= 0 \\ Cy - g &= 0, \end{aligned}$$

and the elimination of λ_2 allows these equations to be written as,

$$\begin{bmatrix} 0 & 0 & C \\ 0 & I_{m_2} & G \\ C^T & G^T & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ r \\ y \end{bmatrix} = \begin{bmatrix} g \\ f \\ 0 \end{bmatrix}. \tag{6.2}$$

6.3 Problem formulation

The problem formulation is taken from [5]. Consider the problem of finding the GCD of two polynomials, $p(x)$ and $q(x)$, which are of degrees m and n respectively. If the coefficients of these polynomials have been given inexactly then it is necessary to calculate $d(x)$, of degree k, such that it divides $p(x) + \delta p(x)$ and $q(x) + \delta q(x)$ exactly, for some polynomials $\delta p(x)$ of degree $\leq m$ and $\delta q(x)$ of degree $\leq n$. In other words, it is necessary to solve,

$$\begin{aligned} p(x) + \delta p(x) &= d(x)u(x) \\ q(x) + \delta q(x) &= d(x)v(x) \end{aligned}$$

where the polynomials $u(x)$ and $v(x)$ are co-prime and of degrees $m - k$ and $n - k$ respectively.

If p, q, d, u and v are the coefficient vectors of $p(x)$, $q(x)$, $d(x)$, $u(x)$ and $v(x)$ respectively (e.g. $p = [p_0, p_1, \ldots, p_m]^T$), then because it is a nearby solution that is being sought, the problem can be stated as the minimisation problem,

$$\min f(d, u, v) = \|\delta p\|^2 + \|\delta q\|^2, \tag{6.3}$$

where $\|\cdot\|$ is the Euclidean norm. This can now be restated to explicitly express the

vectors δp and δq in terms of the vectors d, u and v,

$$\begin{bmatrix} \delta p_0 \\ \delta p_1 \\ \\ \\ \vdots \\ \\ \\ \delta p_n \end{bmatrix} = \begin{bmatrix} d_0 & & & \\ d_1 & d_0 & & \\ \vdots & d_1 & \ddots & \\ d_k & \vdots & \ddots & d_0 \\ & d_k & \ddots & d_1 \\ & & \ddots & \vdots \\ & & & d_k \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{m-k} \end{bmatrix} - \begin{bmatrix} p_0 \\ p_1 \\ \\ \\ \vdots \\ \\ \\ p_n \end{bmatrix},$$

which can be written more compactly as,

$$\delta p = C_1(d)u - p, \tag{6.4}$$

where $C_1(d) \in \mathbb{R}^{(m+1)\times(m-k+1)}$ is the combinatorial matrix, which contains the coefficients of $d(x)$.

The vector δq can be expressed in a similar way,

$$\delta q = C_2(d)v - q, \tag{6.5}$$

where $C_2(d) \in \mathbb{R}^{(n+1)\times(n-k+1)}$ is once again a combinatorial matrix containing the coefficients of $d(x)$.

Note that, provided that d is non-zero, the combinatorial matrices are of full column rank.

It is now possible to combine equations (6.4) and (6.5) to form,

$$\begin{bmatrix} \delta p \\ \delta q \end{bmatrix} = \begin{bmatrix} C_1(d) & 0 \\ 0 & C_2(d) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} - \begin{bmatrix} p \\ q \end{bmatrix}. \tag{6.6}$$

The solution of (6.3) in the standard TLS sense is incorrect, as the structure of the matrices $C_1(d)$ and $C_2(d)$ will be lost. The method of structured total least norm (STLN) is therefore now employed as it will allow (6.6) to be solved in the least squares sense, whilst preserving any relevant relationships between the elements of the constituent vectors and matrices.

STLN: The method of STLN yields a least squares solution of,

$$\min_x \|Ax - b\|^2, \tag{6.7}$$

such that the structure of A and/or b is preserved.

6.4 The method of STLN

It is shown in this section that the method of STLN [30] can be used to solve (6.6). It is assumed that both the degree of the GCD and an initial estimate of its coefficients are available. These may be obtained from methods such as those described in the previous chapter and a triangular decomposition of a resultant matrix, as described in Chapter 4.

Initially, only errors in the matrix A are considered allowing an initial estimation of the coefficients of $d(x)$ to be refined whilst leaving the original polynomials, $p(x)$ and $q(x)$ unchanged. The theory is then extended to allow for perturbations in both A and b and thus produce both a corrected GCD $d(x)$ and the polynomials, $\hat{p}(x)$ and $\hat{q}(x)$, for which it is a GCD. A derivative constraint is introduced in Section 6.7

for the case where $q(x) = p^{(1)}(x)$ which forces the computed polynomial $\hat{q}(x)$ to be the derivative of $\hat{p}(x)$. This is especially important when using GCD factorisation to extract a multiplicity structure of a polynomial, since the degree of the GCD of a polynomial *and its derivative* allows the multiplicities of the roots to be ascertained [34]. If the relationship is not enforced, then $\hat{q}(x) \neq \hat{p}^{(1)}(x)$ and conclusions about the underlying multiplicity structure cannot be drawn. Finally the bounding of the magnitude of the perturbations is discussed in Section 6.8.

The case of errors only in A is now considered.

6.5 Errors in A

Let the matrix $A \in \mathbb{R}^{(m+n+2)\times(m+n-2k+2)}$ and vector $b \in \mathbb{R}^{m+n+2}$ be defined as,

$$A = \begin{bmatrix} C_1(d) & 0 \\ 0 & C_2(d) \end{bmatrix}, \quad \text{and} \quad b = \begin{bmatrix} p \\ q \end{bmatrix}. \tag{6.8}$$

If the elements of $E \in \mathbb{R}^{(m+n+2)\times(m+n-2k+2)}$ are the perturbations of A, then A and E will have the same structure,

$$E = \begin{bmatrix} C_1(z_A) & 0 \\ 0 & C_2(z_A) \end{bmatrix},$$

where,

$$C_1(z_A) = \begin{bmatrix} z_1 & & & \\ z_2 & z_1 & & \\ \vdots & z_2 & \ddots & \\ z_{k+1} & \vdots & \ddots & z_1 \\ & z_{k+1} & \ddots & z_2 \\ & & \ddots & \vdots \\ & & & z_{k+1} \end{bmatrix},$$

and $z_A \in \mathbb{R}^{k+1}$ is a vector containing the distinct perturbations $z_1, \ldots, z_{k+1}$ in E. The matrix $C_2(z_A)$ has a very similar structure. From this, equation (6.7) now becomes,

$$\min_{x, z_A} \|(A + E)x - b\|^2.$$

It will be necessary to represent the vector Ex in terms of z_A. This can be achieved by the introduction of the matrix $Y \in \mathbb{R}^{(m+n+2)\times(k+1)}$ which satisfies,

$$\begin{aligned} Yz_A &= Ex, \\ &= \begin{bmatrix} C_1(z_A) & 0 \\ 0 & C_2(z_A) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \\ &= \begin{bmatrix} C_3(u) \\ C_4(v) \end{bmatrix} z_A \end{aligned} \tag{6.9}$$

where Y is constructed such that,

$$Y = \begin{bmatrix} C_3(u)_{(m+1)\times(k+1)} \\ C_4(v)_{(n+1)\times(k+1)} \end{bmatrix},$$

where $C_3(u)$ and $C_4(v)$ are combinatorial matrices in the coefficients u and v respectively.

Example 6.1. If $m = 4$, $n = 3$ and $k = 2$, then from (6.9),

$$Y = \begin{bmatrix} u_0 & 0 & 0 \\ u_1 & u_0 & 0 \\ u_2 & u_1 & u_0 \\ 0 & u_2 & u_1 \\ 0 & 0 & u_2 \\ v_0 & 0 & 0 \\ v_1 & v_0 & 0 \\ 0 & v_1 & v_0 \\ 0 & 0 & v_1 \end{bmatrix}$$

□

In standard TLS problems the matrix E is unstructured, and thus there always exists a matrix $E = E_0$ such that,

$$(A + E_0)x = b, \tag{6.10}$$

is satisfied. However, in the method of STLN the matrix E is structured and thus a

solution to (6.10) is not guaranteed. The residual of (6.10) due to an approximate solution of this equation is,

$$r(z_A, x) = b - Ax - Yz_A,$$

using (6.9).

The definition of z_A implies that each of its elements may represent more than one element of E, but this information is not included in z_A. It is possible to address this deficiency by solving the LSE problem,

$$\min \|Dz_A\|, \quad \text{subject to} \quad r(z_A, x) = 0 \tag{6.11}$$

where $D = (m + n - 2k + 2)I_{k+1} \in \mathbb{R}^{(k+1)\times(k+1)}$, is a diagonal matrix that accounts for the repetition of the elements of z_A in E, and I_{k+1} is an identity matrix of order $k + 1$.

The equation $r(z_A, x) = 0$ is non-linear and it must therefore be linearised and solved iteratively. This may be done by performing a Taylor expansion of $r(z_A, x)$ and only retaining lowest order terms,

$$\begin{aligned} r(z_A + \delta z_A, x + \delta x) &= b - A(x + \delta x) - (Y + \delta Y)(z_A + \delta z_A) \\ &\approx r(z_A, x) - Y\delta z_A - A\delta x - \delta Y z_A. \end{aligned}$$

It follows from (6.9) that,

$$\delta Y z_A = E\delta x, \tag{6.12}$$

and hence,

$$r(z_A + \delta z_A, x + \delta x) \approx r(z_A, x) - Y\delta z_A - (A + E)\delta x. \tag{6.13}$$

The problem (6.11) therefore becomes,

$$\begin{aligned} \min \left\| \begin{bmatrix} D & 0 \end{bmatrix} \begin{bmatrix} \delta z_A \\ \delta x \end{bmatrix} - Dz_A \right\| \\ \text{subject to} \quad \begin{bmatrix} Y & (A+E) \end{bmatrix} \begin{bmatrix} \delta z_A \\ \delta x \end{bmatrix} = r(z_A, x). \end{aligned} \tag{6.14}$$

This minimisation can now be solved using Lagrange multipliers.

6.5.1 Solving errors in A using Lagrange multipliers

In order to solve the minimisation problem by the method of Lagrange multipliers it is necessary to replace the matrices C and G, and the vectors g, f and y in (6.2) by,

$$\begin{aligned} C &\rightarrow \begin{bmatrix} Y & (A+E) \end{bmatrix} \in \mathbb{R}^{(m+n+2)\times(m+n-k+3)} \\ G &\rightarrow \begin{bmatrix} D & 0 \end{bmatrix} \in \mathbb{R}^{(k+1)\times(m+n-k+3)} \\ g &\rightarrow r(z,x) \in \mathbb{R}^{m+n+2} \\ f &\rightarrow -Dz \in \mathbb{R}^{k+1} \\ y &\rightarrow \begin{bmatrix} \delta z \\ \delta x \end{bmatrix} \in \mathbb{R}^{m+n-k+3}, \end{aligned}$$

where, $\delta z \in \mathbb{R}^{k+1}$ and $\delta x \in \mathbb{R}^{m+n-2k+2}$.

Algorithm 6.1: Errors in A using Lagrange multipliers

Input: The coefficient vectors p, q and d and a tolerance on the residual θ_{res}.

Output: The coefficient vector of the refined approximate GCD d.

Begin

1. Construct the matrix A and vector b from (6.8) and compute the initial value of x from,

$$\min_x \|Ax - b\|.$$

Construct the residual $r = b - Ax$. Set $E = 0$, $z_A = 0$.

2. **repeat**

(a) Construct Y from x.

(b) Solve (6.2), where C, G, g, f and y are given in Section 6.5.1.

$$\begin{bmatrix} \lambda & r & y \end{bmatrix}^T.$$

(c) From the vector y of the solution, set $z_A := z_A + \delta z_A$ and $x := x + \delta x$.

(d) Construct E from z_A. Compute the residual,

$$r = b - (A + E)x.$$

until $\frac{\|r\|}{\|b\|} < \theta_{\text{res}}$

End

In order to illustrate the performance of the algorithm and provide a criteria by which a calculated approximate GCD may be assessed, an example is now provided showing the calculation of the approximate GCD of a polynomial and its derivative.

Example 6.2. Consider the polynomial,

$$p(x) = (x - 0.9)^7(x - 0.95)^8(x - 1)^9 \tag{6.15}$$

and its derivative $p^{(1)}(x)$, whose GCD, $d(x)$, is

$$d(x) = (x - 0.9)^6(x - 0.95)^7(x - 1)^8. \tag{6.16}$$

If elements, p_i, of the coefficient vector, p, of $p(x)$ are perturbed by a signal to noise ratio of $\Delta e = 10^{10}$ such that,

$$p \to \tilde{p} = p + \frac{\varrho p}{\Delta e}, \tag{6.17}$$

where $\varrho \in \mathbb{R}^{25}$ is a vector of uniformly distributed random numbers on the interval $[-1, 1]$. Consequently, $\tilde{p}(x)$ and $\tilde{p}^{(1)}(x)$ are co-prime and by definition do not possess a non-trivial GCD. Assume that $d(x)$ is now perturbed to $\tilde{d}(x)$ by noise with a signal to noise ratio, $\Delta e = 10^5$, in order to represent an initial estimate of the GCD.

The reason for computing $d(x)$ from $p(x)$ and $p^{(1)}(x)$ then perturbing it, as opposed to computing an estimate of $d(x)$ from $\tilde{p}(x)$ and $\tilde{p}^{(1)}(x)$, for example via a triangular decomposition, is because it allows greater control over the noise. It is appreciated that this is not possible in practice, since the theoretically exact polynomial

will not be known. However, in order that the algorithm's performance can be assessed independently of any external input factors, such as peculiarities of any chosen algorithm for calculating initial estimates, this method is felt to be appropriate.

The algorithm is then run and the residual of the computed system is then examined. It is reasoned that the norm of the noise in $b = [\ \tilde{p}\ \ \tilde{p}^{(1)}\ ]^T$ is,

$$\varepsilon_n = \frac{\|\varrho b\|}{\Delta e} \leq \frac{\|b\|}{\Delta e}, \tag{6.18}$$

hence it is reasonable to set the threshold $\theta = \frac{\|b\|}{\Delta e}$. If the norm of the normalised residual is given by, $r_n = \|b-(A+E)x\|/\|b\|$, then the solution is valid if $r_n \leq \theta/\|b\|$. Consequently, $\theta_{\text{res}} = 1/\Delta e$.

For this example, the initial residual, $\|b - Ax\|/\|b\|$, and the residual of the computed solution, $\|b - (A + E)x\|/\|b\|$, which was reached after 22 iteration, are,

$$\begin{aligned} \|b - Ax\|/\|b\| &= 2.42717 \times 10^{-6} \\ \|b - (A + E)x\|/\|b\| &= 3.55516 \times 10^{-11} \end{aligned}$$

Clearly, the computed solution is valid, since $\|b - (A + E)x\|/\|b\| \leq \theta_{\text{res}}$. □

When, as in Example 6.2, the given polynomials are perturbed such that they have a non-constant GCD it may be desirable to "*correct*" them in addition to the GCD estimate $\tilde{d}(x)$. This will allow $\hat{d}(x) = \text{GCD}(\hat{p}(x), \hat{q}(x))$ to be computed with a much smaller residual. This is addressed in the next section.

6.6 Errors in both A and b

In the previous section, only errors in A were considered. This section extends the analysis by considering the case when both A and b have structured errors. That is,

$$(A + E)x = b + h,$$

where the matrix E represents the errors in A, as in the previous section, and a vector $h \in \mathbb{R}^{m+n+2}$ that represents the errors in b is given by,

$$h = \begin{bmatrix} z_{k+2} & z_{k+3} & \cdots & z_{m+n+k+3} \end{bmatrix}^T,$$

where k is the degree of the GCD. The vector of distinct perturbations is now represented by $z = [z_A, z_b]^T \in \mathbb{R}^{m+n+k+3}$, where z_b are the distinct perturbations in b.

It follows from the definitions of h and z that there exists a matrix $P \in \mathbb{R}^{(m+n+2)\times(m+n+k+3)}$ such that,

$$h = Pz = \begin{bmatrix} 0_{(m+n+2)\times(k+1)} & I_{m+n+2} \end{bmatrix} z, \tag{6.19}$$

where I_{m+n+2} is the identity matrix of order $m+n+2$ and the subscripts on the zero matrix indicate its order.

Example 6.3. If $m = 4$, $n = 3$ and $k = 2$, then

$$P = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad h = \begin{bmatrix} z_4 \\ z_5 \\ z_6 \\ z_7 \\ z_8 \\ z_9 \\ z_{10} \\ z_{11} \\ z_{12} \end{bmatrix}.$$

□

The residual $r(z, x)$ that is associated with an approximate solution of (6.6) due to the perturbations in $E(z)$ and $h(z)$ is given by,

$$r(z, x) = (b + h) - (A + E)x, \quad h = Pz. \tag{6.20}$$

The matrix Y, from the previous section, is now replaced by,

$$Y = \begin{bmatrix} C_3(u)_{(m+1)\times(k+1)} & 0_{(m+1)\times(m+n+2)} \\ C_4(v)_{(n+1)\times(k+1)} & 0_{(n+1)\times(m+n+2)} \end{bmatrix} \in \mathbb{R}^{(m+n+2)\times(m+n+k+3)}. \tag{6.21}$$

Example 6.4. If $m = 4$, $n = 3$ and $k = 2$, then from (6.9),

$$Y = \begin{bmatrix} u_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ u_1 & u_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ u_2 & u_1 & u_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & u_2 & u_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & u_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_1 & v_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & v_1 & v_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & v_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{6.22}$$

□

By ignoring higher order terms it is possible to linearise the minimisation of the residual $r(z, x)$, and hence it can be expressed as,

$$\begin{aligned} r(z + \delta z, x + \delta x) &= b + h - A(x + \delta x) - E(x + \delta x) \\ &\approx b + Pz + P\delta z - Ax - A\delta x - E(z)x - E(z)\delta x \\ &\quad - \left(\sum_{i=0}^{m} \frac{\partial E}{\partial z_i} \delta z_i \right) x \\ &= r(z, x) + P\delta z - A\delta x - E\delta x - \delta E x \\ &= r(z, x) + P\delta z - A\delta x - E\delta x - Y\delta z \quad \text{since} \quad Y\delta z = \delta E x \\ &= r(z, x) - (Y - P)\delta z - (A + E)\delta x. \end{aligned} \tag{6.23}$$

The minimisation problem can therefore be approximated by,

$$\min_{\tilde{r}=0;\delta z,\delta x} \left\| \begin{array}{c} \tilde{r} \\ D(z+\delta z) \end{array} \right\|, \tag{6.24}$$

where $\tilde{r} = r(z+\delta z, x+\delta x)$ and $D \in \mathbb{R}^{(m+n+k+3)\times(m+n+k+3)}$ is a diagonal matrix that accounts for the repetition of the elements of z in A and b and thus,

$$D = \begin{bmatrix} (m+n-2k+2)I_{k+1} & 0 \\ 0 & I_{m+n+2} \end{bmatrix}. \tag{6.25}$$

Hence, equation (6.14) becomes,

$$\min \left\| \begin{bmatrix} D & 0 \end{bmatrix} \begin{bmatrix} \delta z \\ \delta x \end{bmatrix} - Dz \right\|$$

$$\text{subject to} \quad \begin{bmatrix} (Y-P) & (A+E) \end{bmatrix} \begin{bmatrix} \delta z \\ \delta x \end{bmatrix} = r(z,x).$$

6.6.1 Solving errors in A and b using Lagrange multipliers

As stated previously, the minimisation problem (6.24) can be solved by the method of Lagrange multipliers. In particular, the matrices C and G, and the vectors g, f

and y in (6.2) are replaced by,

$$\begin{aligned} C &\rightarrow \begin{bmatrix} (Y-P) & (A+E) \end{bmatrix} \in \mathbb{R}^{(m+n+2)\times(2m+2n-k+5)} \\ G &\rightarrow \begin{bmatrix} D & 0 \end{bmatrix} \in \mathbb{R}^{(m+n+k+3)\times(2m+2n-k+5)} \\ g &\rightarrow r(z,x) \in \mathbb{R}^{m+n+2} \\ f &\rightarrow -Dz \in \mathbb{R}^{m+n+k+3} \\ y &\rightarrow \begin{bmatrix} \delta z \\ \delta x \end{bmatrix} \in \mathbb{R}^{2m+2n-k+5}, \end{aligned}$$

where, $\delta z \in \mathbb{R}^{m+n+k+3}$ and $\delta x \in \mathbb{R}^{m+n-2k+2}$.

Algorithm 6.2: Errors in A and b using Lagrange multipliers

Input: The coefficient vectors p, q and d and a tolerance on the residual θ_{res}.

Output: The coefficient vector of the refined approximate GCD d and the corresponding coefficients of the corrected polynomials $\hat{p}$ and $\hat{q}$.

Begin

1. Construct the matrices A and P and vector b from (6.8) and compute the initial value of x from,

$$\min_{x} \|Ax - b\|.$$

Construct the residual $r = b - Ax$. Set $E = 0$, $h = 0$, $z_b = 0$, $z_A = 0$.

2. **repeat**

   (a) Construct Y from x.

(b) Solve (6.2), where C, G, g, f and y are given in Section 6.6.1.

$$\begin{bmatrix} \lambda & r & y \end{bmatrix}^T.$$

(c) From the vector y of the solution, set $z_A := z + \delta z_A$, $z_b := z + \delta z_b$ and $x := x + \delta x$.

(d) Construct E from z_A and h from z_b. Compute the residual,

$$r = (b + h) - (A + E)x.$$

until $\frac{\|r\|}{\|b\|} < \theta_{\text{res}}$

End

It was stated that one reason for allowing structured perturbations in both A *and* b was because it would allow a GCD to be calculated with a vanishingly small residual. Indeed, if the polynomial in Example 6.2 is considered with the same values of Δc, a valid solution can be achieved in just six iterations, even with an error tolerance set to $\theta_{\text{res}} = 10^{-16}$,

$$\begin{aligned} \|b - Ax\|/\|b\| &= 2.42717 \times 10^{-6} \\ \|(b + h) - (A + E)x\|/\|b\| &= 1.93937 \times 10^{-17}. \end{aligned}$$

Clearly, in this instance θ_{res} is very much smaller than the actual error in the coefficients of $\hat{p}(x)$ and its derivative, and a residual this small would not have been

achieved if only errors in A had been considered. Consequently, by allowing structured perturbations in both A and b a much "better" solution, with respect to the norm of the residual, has been achieved. However, this is not a better solution if it is to be used to calculate a multiplicity structure since the calculated polynomial $\hat{q}(x) \neq \hat{p}^{(1)}(x)$. It is therefore necessary to enforce the derivative constraint in b and this is done in the next section.

6.7 Enforcing a derivative constraint

In the case where it is necessary to find an approximate GCD of a polynomial $p(x)$ and its derivative $p^{(1)}(x)$ it is possible to utilise the extra structure to enforce the relationship between $p(x)$ and $p^{(1)}(x)$. In particular, the vector $h \in \mathbb{R}^{2m+1}$ becomes,

$$h = \begin{bmatrix} z_{k+2} & z_{k+3} & \cdots & z_{m+k+1} & z_{m+k+2} & mz_{k+2} & (m-1)z_{k+3} & \cdots & 2z_{m+k} & z_{m+k+1} \end{bmatrix}^T,$$

for the perturbation vector $z = \begin{bmatrix} z_1 & \cdots & z_{m+k+2} \end{bmatrix}^T$. From this the matrices P, Y and D become,

$$P = \begin{bmatrix} 0_{(m+1)\times(k+1)} & I_{m+1} \\ 0_{m\times(k+1)} & \begin{bmatrix} I_m & 0_{m\times 1} \end{bmatrix} \end{bmatrix} \in \mathbb{R}^{(2m+1)\times(m+k+2)}$$

$$Y = \begin{bmatrix} C_3(u)_{(m+1)\times(k+1)} & 0_{(m+1)\times(m+1)} \\ C_1(v)_{m\times(k+1)} & 0_{m\times(m+1)} \end{bmatrix} \in \mathbb{R}^{(2m+1)\times(m+k+2)},$$

and,

$$D = \begin{bmatrix} (2m-2k+1)I_{k+1} & 0 \\ 0 & W \end{bmatrix} \in \mathbb{R}^{(m+k+2)\times(m+k+2)},$$

where $W \in \mathbb{R}^{(m+1)\times(m+1)}$ is a diagonal matrix whose elements along the diagonal are $\{w_i\}_{i=1}^{m+1} = (m+2) - i$.

Consequently, the matrices C and G, and the vectors g, f and y in (6.2) become,

$$\begin{aligned} C &\rightarrow \begin{bmatrix} (Y-P) & (A+E) \end{bmatrix} \in \mathbb{R}^{(2m+1)\times(3m-k+3)} \\ G &\rightarrow \begin{bmatrix} D & 0 \end{bmatrix} \in \mathbb{R}^{(m+k+2)\times(3m-k+3)} \\ g &\rightarrow r(z,x) \in \mathbb{R}^{2m+1} \\ f &\rightarrow -Dz \in \mathbb{R}^{m+k+2} \\ y &\rightarrow \begin{bmatrix} \delta z \\ \delta x \end{bmatrix} \in \mathbb{R}^{3m-k+3}, \end{aligned}$$

where, $\delta z \in \mathbb{R}^{m+k+2}$ and $\delta x \in \mathbb{R}^{m-2k+1}$. Solve using Algorithm 6.2.

Using the derivative constraint allows the solution space to be reduced, in the same way as enforcing the structure of A and b did in Section 6.4. This ensures that only *valid* solutions are obtained. In the case of the polynomial from Example 6.2 it is possible to achieve a good and solution in just 3 iterations. With the resulting residuals being,

$$\begin{aligned} \|b - Ax\|/\|b\| &= 2.42717 \times 10^{-6} \\ \|b - (A+E)x\|/\|b\| &= 1.56831 \times 10^{-14}. \end{aligned}$$

Clearly, the achieved residual $\|b-(A+E)x\|/\|b\|$ is well within the limits defined by (6.18) and, due to the newly introduced derivative constraints, $\hat{q}(x)$ is now equal to $\hat{p}^{(1)}(x)$.

So far, only information regarding Δe in the context of a minimum allowable residual has been considered. However, allowing perturbations in b implies that there should also be some limit, based on Δe, on the amount they may be moved. It would seem pertinent to allow the maximum allowable perturbation, $\delta\bar{p}_i$, in a coefficient p_i to be,

$$\delta\bar{p}_i = \pm\frac{p_i}{\Delta e},$$

and therefore the solution is only valid if,

$$\frac{|\hat{p}_i - p_i|}{|p_i|} \leq \frac{1}{\Delta e}.$$

In the case illustrated above, the average correction to the coefficients of $p(x)$ as defined in (6.4) and (6.5) is 6×10^{-6} which is more than 4 orders of magnitude greater than the allowable correction. An attempt must therefore be made to constrain the solution space further and some preliminary work based on constrained least squares is now detailed.

6.8 Bounded least squares and the approximate GCD

Given knowledge of the signal to noise ratio in the coefficients, it is possible to reduce the solution space still further since this information motivates the use of *bounded*

least squares (BLS) which limits the search space to within a specified region. That is,

$$\min_x \|Ax - b\| \quad \text{subject to} \quad l \leq x \leq h \tag{6.26}$$

where l and h are vectors of simple scalar bounds and $l \leq h$. It is therefore possible to ensure that the coefficients are not perturbed by more than the signal to noise ratio and hence fall within a *valid solution space*. This may be done using an active set method, p.198-203 [3], to solve the minimisation in (6.26). A brief outline of the method is now detailed.

6.8.1 An active set algorithm for BLS

Active set algorithms consist of a sequence of equality constrained problems which are solved according to the prediction of the correct active or *working set*. The working set includes those constraints which are satisfied at the current approximation, i.e. in the case of simple BLS, $x_i = l_i$ or $x_i = h_i$. It should be noted that while this set consists of only constraints that are satisfied, it is not necessary that all such constraints are included.

Initially, it is necessary to check the feasibility of the bounds i.e. $l_i \leq h_i$, $i = 1\ldots, n$, where $x \in \mathbb{R}^n$. The index set of x is then partitioned according to,

$$\{1, 2, \ldots, n\} = \mathcal{F} \cup \mathcal{B},$$

where $i \in \mathcal{F}$ if x_i is a free variable and $i \in \mathcal{B}$ if x_i is fixed at its lower or upper bound, i.e. it is in the *working set*. If, at the kth iterate, the matrix $E_{\mathcal{B}}^T$ is constructed such that it consists of the rows e_i, $i \in \mathcal{B}$ of the unit matrix I_n, and $E_{\mathcal{F}}$ is defined similarly,

then it is possible to define a permutation matrix Q_k as,

$$Q_k = \begin{bmatrix} E_{\mathcal{F}} & E_{\mathcal{B}} \end{bmatrix}, \tag{6.27}$$

from which it is easily seen that the product

$$AQ_k = \begin{bmatrix} AE_{\mathcal{F}} & AE_{\mathcal{B}} \end{bmatrix} = \begin{bmatrix} A_{\mathcal{F}} & A_{\mathcal{B}} \end{bmatrix}. \tag{6.28}$$

corresponds to the permutation of the columns of A.

If x_k is the iterate at the kth step that satisfies the working set of constraints associated with the matrix $E_{\mathcal{B}}$, then the $(k+1)$th iterate may be defined as,

$$x_{k+1} = x_k + \alpha_k p_k, \tag{6.29}$$

where p_k is the search direction and α_k is a nonnegative step length. The search direction is constructed so that the working set is satisfied for all values of α_k. This is achieved by defining p_k as,

$$p_k = E_{\mathcal{F}} q_k, \tag{6.30}$$

and determining the vector q_k so that $x_k - E_{\mathcal{F}} q_k$ minimises the objective function of the unconstrained least squares problem,

$$\min_{q_k} \|AE_{\mathcal{F}} q_k - r_k\|, \quad r_k = b - Ax_k. \tag{6.31}$$

A value for the maximum nonnegative step length, $\bar{\alpha}$, along p_k for which x_{k+1}

remains feasible with respect to the constraints not in the working set is now determined. If $\bar{\alpha} \leq 1$ then $\alpha_k = \bar{\alpha}$ and the constraints that are hit are added to the working set for the next iteration.

If $\bar{\alpha} > 1$ then $\alpha_k = 1$ and x_{k+1} will minimise the objective function when the constraints in the working set are treated as equalities. In this case the optimality of x_{k+1} is now checked by computing the Lagrange multipliers for the constraints in the working set,

$$\lambda = -E_{\mathcal{B}}^T A^T r_{k+1}. \tag{6.32}$$

The Lagrange multiplier λ_i, $i \in \mathcal{B}$, for the constraint $l_i \leq x_i \leq h_i$ is considered to be optimal if $\lambda_i \leq 0$ at an upper bound and $\lambda_i \geq 0$ at a lower bound. If all multipliers are optimal then the algorithm terminates. If one or more multipliers are not optimal, then the least optimal one is deleted from the working set for the next iteration. This may be achieved via a right circular shift, in which the columns are permuted such that,

$$k+1, \ldots, q-1, q \Rightarrow q, k+1, \ldots, q-1., \tag{6.33}$$

in the case where the bound to be dropped is x_q. This can be achieved by,

$$AQ_{k+1} = AQ_k P_R(k, q), \tag{6.34}$$

where $P_R(k, q)$, $q > k+1$, is a permutation matrix which performs the right circular shift. Similarly, if the bound corresponding to x_q becomes active then it can be added to the working set by,

$$AQ_{k+1} = AQ_k P_L(k, q), \tag{6.35}$$

where $P_R(k,q)$, $q < k+1$, is a permutation matrix which performs a left circular shift such that the columns are permuted,

$$q, q+1, \ldots, k \Rightarrow q+1, \ldots, k, q. \tag{6.36}$$

The active set method can now be used to calculate a solution to (6.2). It should be noted that, as stated previously, only preliminary testing has been carried out and some problems involving floating point accuracy can result in bounds being violated. However, it is believed that these problems could be overcome with further work, and despite this problem, results are promising and for the polynomial in Example 6.2 a normalised residual of 1.05236×10^{-10} is achieved and the average correction applied to the coefficients is 6×10^{-10} which is an improvement on the case where BLS wasn't used.

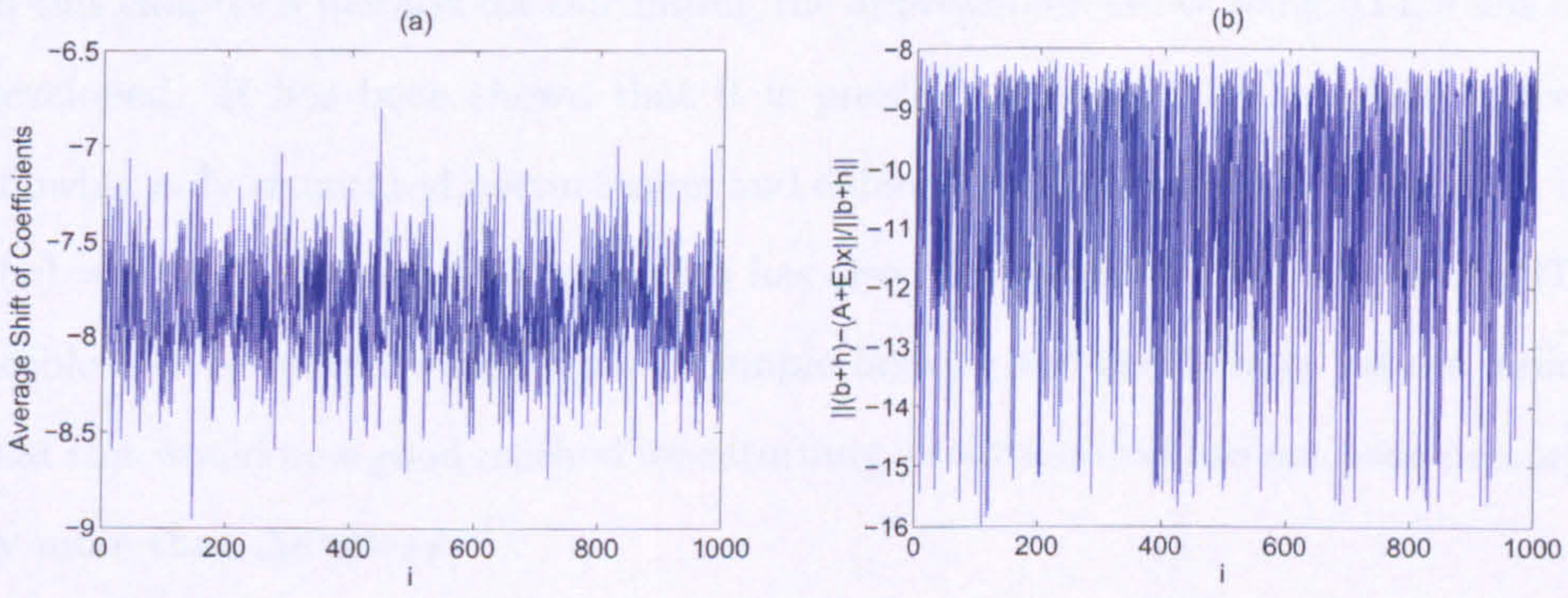

Figure 6.1: For a test set of 1000 polynomials. (a) The average shift of the coefficients; (b) The normalised residual. The y-axis uses the log scale.

Figures 6.1 (a) and 6.1 (b) display the results from running the algorithm on the set of 1000 test polynomials in the case where the signal to noise ratio in both the

polynomial and the initial estimate of the GCD is $\Delta e = 10^8$. Figure 6.1 (a) shows the average normalised distance each coefficient was moved whilst Figure 6.1 (b) shows the normalised residual. Clearly, despite the breaking of the bounds that BLS attempts to enforce, good results are achieved with respect to the normalised residuals which are bounded to within the limits defined in (6.18). As stated previously, the reason for the violation of the bounds is believed to be mainly due to underflow as a result of dealing with coefficients with tiny coefficients of vastly differing magnitudes. Dealing with this problem would be highly non-trivial, perhaps involving multi-precision arithmetic and therefore it may be prudent to regard the bounding process as limiting rather than bounding the perturbations.

6.9 Summary and future work

In this chapter a method for calculating the approximate GCD using STLN has been developed. It has been shown that it is possible to restrict the solution space by allowing only structured perturbation and enforcing a derivative constraint such that $\hat{q}(x) = \hat{p}^{(1)}(x)$. An active set algorithm has also been detailed which allows the STLN problem to be solved within a set of simple bounds and preliminary results indicate that this would be a good method for returning a solution that has not been perturbed by more than Δe allows.

Apart from obvious refinements to the algorithm, such as update procedures to increase computational efficiency, there is also scope to extend the theory to calculating GCD's for a given multiplicity structure. For example, the polynomial,

$$p(x) = (x-1)^2(x-2)^3(x-3)^4 \tag{6.37}$$

has a multiplicity structure $\mu = [2, 3, 4]$ and the successive GCD calculations therefore produce,

$$\begin{aligned} d_1(x) &= \mathrm{GCD}(p(x), p^{(1)}(x)) = (x-1)(x-2)(x-3)^3 \\ d_2(x) &= \mathrm{GCD}(d_1(x), d_1^{(1)}(x)) = (x-2)(x-3)^2 \\ d_3(x) &= \mathrm{GCD}(d_2(x), d_2^{(1)}(x)) = (x-3). \end{aligned}$$

If this structure is known in advance then the problem can be formulated such that,

$$\begin{bmatrix} \begin{bmatrix} C(d_1) & 0 \\ 0 & C(d_1) \end{bmatrix} & & \\ & \begin{bmatrix} C(d_2) & 0 \\ 0 & C(d_2) \end{bmatrix} & \\ & & \begin{bmatrix} C(d_3) & 0 \\ 0 & C(d_3) \end{bmatrix} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{bmatrix} = \begin{bmatrix} p \\ p^{(1)} \\ d_1 \\ d_1^{(1)} \\ d_2 \\ d_2^{(1)} \end{bmatrix}.$$

If, as is more likely, the multiplicity structure is not known in advance then this structure can be built up over several iterations. For example, the first iteration would have the structure,

$$\begin{bmatrix} C(d_1) & 0 \\ 0 & C(d_1) \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \begin{bmatrix} p \\ p^{(1)} \end{bmatrix},$$

the second would have the structure,

$$\begin{bmatrix} \begin{bmatrix} C(d_1) & 0 \\ 0 & C(d_1) \end{bmatrix} & \\ & \begin{bmatrix} C(d_2) & 0 \\ 0 & C(d_2) \end{bmatrix} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} p \\ p^{(1)} \\ d_1 \\ d_1^{(1)} \end{bmatrix},$$

and so on. This would force each iteration of the factorisation procedure to bare the necessary relationships to each other and hence reduce the solution space still further. It is believed that this would aid in the case where the GCD algorithm is used as subroutine of a root-finding algorithm.

An alternative way of addressing the approximate GCD problem is by calculating a low rank approximation of a Sylvester matrix using STLN and sub-resultants. This method can be found in [39] which contains not only the theory but also non-trivial examples using polynomials of up to degree 50. The equivalent Bernstein basis formulation is presented in [37]. These methods are not discussed in this thesis because they are already in the public domain.

In the next chapter the STLN GCD and the method of rank determination via ML are combined into one algorithm.

Chapter 7

Calculating the GCD of two polynomials

In the previous two chapters methods have been developed to calculate the rank of a resultant matrix and refine an estimate of an approximate GCD for two polynomials. Until now, these methods have been considered in isolation with the degree and initial estimates being given to the refinement algorithm described in Chapter 6. In practice, however, it is more likely that the GCD will have to be computed from the polynomials alone. This can be achieved by calculating the rank of the resultant matrix corresponding to the given polynomials and then performing a triangular decomposition, such as QR or LU [11]. From this the coefficients may be extracted as shown in Chapter 4.

A generalised algorithm for computing the approximate GCD of two polynomials can therefore be formulated as in Algorithm 7.1.

Algorithm 7.1: Computing the approximate GCD of two polynomials

Input: The coefficient vectors $p \in \mathbb{R}^{m+1}$ and $q \in \mathbb{R}^{n+1}$ of the polynomials $p(x)$ and $q(x)$ respectively.

1. Construct the resultant matrix $R = R(p(x), q(x)) \in \mathbb{R}^{t \times t}$.

2. Calculate the rank, r, of R.

3. Perform an upper triangular decomposition on R. The last $t - r + 1$ elements on the rth row are the initial estimates of the coefficients of the GCD.

4. Refine the estimate using an optimisation method.

Output: The coefficient vector d of the approximate GCD.

In this chapter the methods developed in Chapters 5 and 6 are used in steps two and four of Algorithm 7.1, respectively, and the algorithm is tested on a set of 100 pairs of random, inexact polynomials. Each polynomial pair has an almost common divisor and are generated as described in Algorithm 7.2.

Algorithm 7.2: Polynomial Test Set 2

1. Generate three polynomials,

$$\begin{aligned} d(x) &= \prod_{i=0}^{k}(x-\alpha_i), \\ u(x) &= \prod_{i=0}^{h}(x-\beta_i), \\ v(x) &= \prod_{i=0}^{l}(x-\gamma_i), \end{aligned}$$

where k, h and l are randomly generated integers on the intervals $[3,5]$, $[3,10]$, $[3,7]$ respectively.

2. Form the polynomials $p(x)$ and $q(x)$ such that,

$$p(x) = d(x)u(x) \quad \text{and} \quad q(x) = d(x)v(x)$$

3. Perturb the coefficients $\{p_i\}$ and $\{q_j\}$ of $p(x)$ and $q(x)$ in the componentwise sense, such that,

$$\{p_i\} \rightarrow \{\tilde{p}_i\} = \left\{p_i + \frac{p_i r_i}{\Delta e}\right\},$$

and hence $p(x) \rightarrow \tilde{p}(x)$ and $q(x) \rightarrow \tilde{q}(x)$.

7.1 Results and Analysis

In this section results from Algorithm 7.1 are given for various different scenarios involving different implementations of steps two and four. These are,

- Options for step two in Algorithm 7.1,
    1. Theoretically exact rank is given.
    2. Rank is calculated using `MLrank()`.
- Options for step four in Algorithm 7.1,
    1. Only errors in A are considered.
    2. Errors in A and b are considered.
    3. Errors in A and b are considered with bounds.

Results are given for the theoretically exact rank in addition to the calculated rank because while this algorithm is presented as a black box solution, step four in Algorithm 7.1 is independent of the method used to calculate the rank of the resultant matrix. Consequently it is desirable to observe the performance of the algorithm on the test set, independent of any error in step two.

All results, unless otherwise stated, are obtained from polynomials whose componentwise signal to noise ratio, Δe, is equal to 10^8. The presented results are however representative of those for different values of Δe.

Figure 7.1 shows the error in the calculated rank for the test set of 100 polynomials and it can be clearly seen that `MLrank()` accurately calculated the rank for approximately 80% of the set. Consequently approximately 20% of the computed GCDs are of the wrong degree. From Figure 7.1 it can be seen that in the majority of these

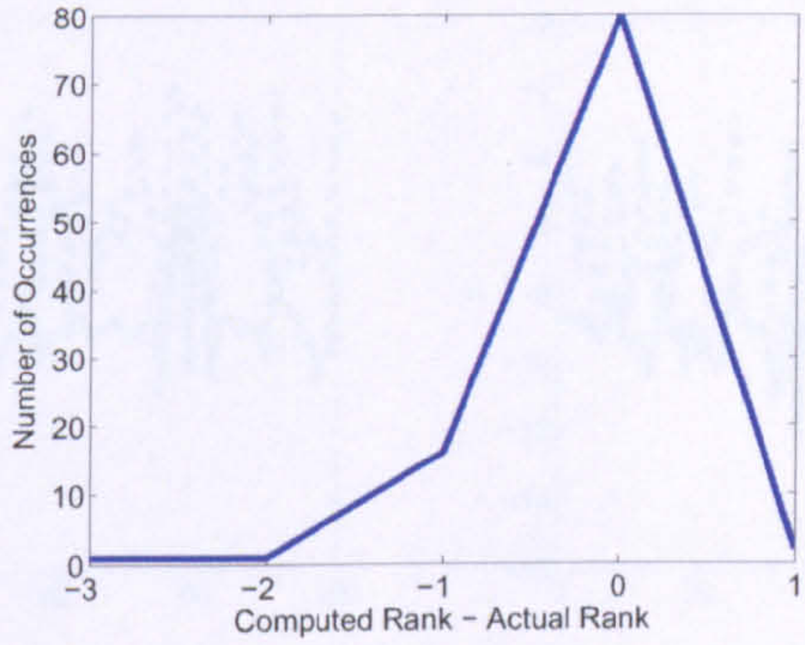

Figure 7.1: The error between the computed rank and the theoretically exact rank when calculated using `MLrank()`.

cases the estimated degree of the GCD is greater than the theoretically exact degree. This will clearly lead to increased residuals with respect to the final refinements of the GCDs.

This is seen in Figures 7.2 (a) and (b) which shows the normalised residual,

$$r = \frac{\|Ax - b\|}{\|b\|},$$

where A, x and b are the matrix and vectors associated with the system (6.6), for the initial estimate and final refinement. Figure 7.2 (a) shows the results when the theoretically exact rank is given and Figure 7.2 (b) shows the results for when `MLrank()` is used to calculate the rank.

It can be clearly seen that, while being generally very similar, there are several cases where the refined residuals in (b) are greater than those in (a). This is due to the overestimates made by `MLrank()` with regard to the degree of the GCD.

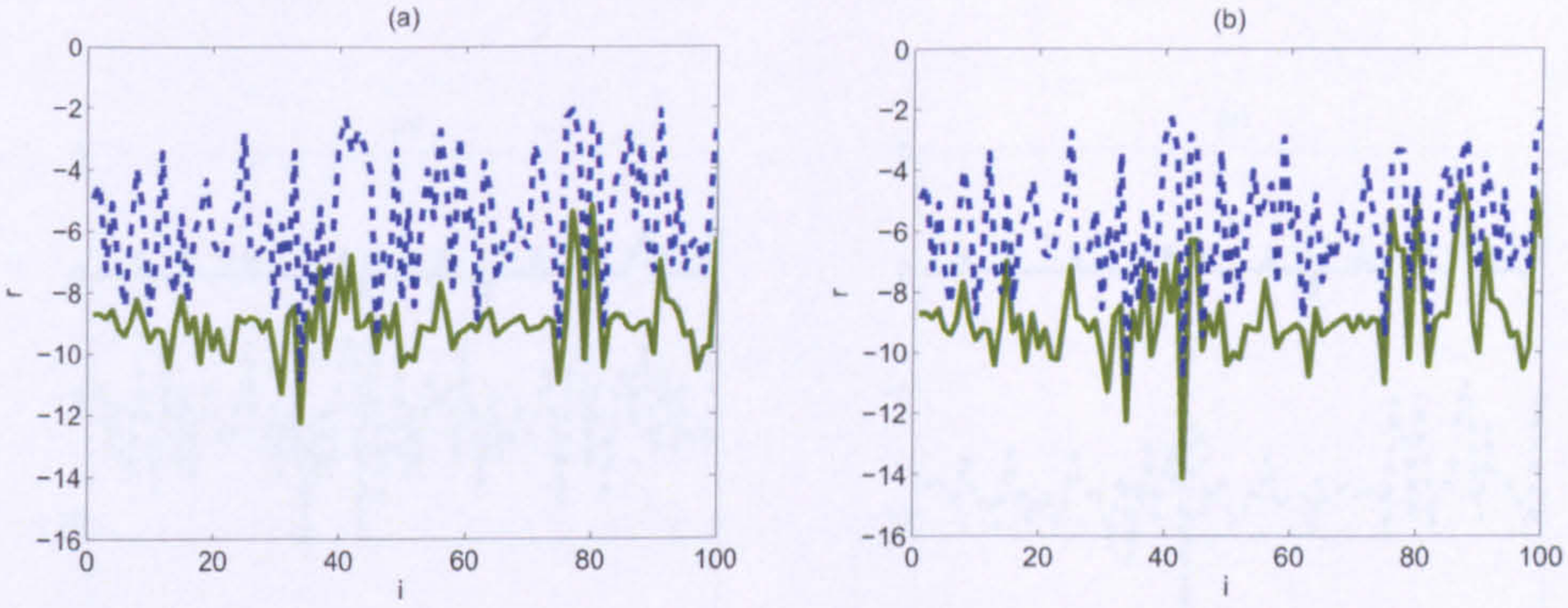

Figure 7.2: The normalised residual for the initial estimate = - - and after the refinement = — when considering only errors in A. (a) Given theoretically exact rank; (b) Rank calculated using `MLrank()`. The y-axis uses the log scale.

The link between the error in the degree of the GCD and the residual can be seen in Figures 7.3 (a) and (b) which show the errors in the rank and corresponding initial and refined residuals respectively. In Figure 7.3 (a) the initial estimates are all fairly inaccurate, with residuals significantly larger than Δe. In contrast, the residuals in Figure 7.3 (b) are, in general, only greater than Δe when the rank of the resultant matrix is underestimated/the degree of the GCD overestimated. This is as expected because while two polynomials with a GCD of degree k also have common divisors of degrees $k-1,\ldots,1$ they do not have common divisors of degree greater than k and therefore, when considering only errors in A, it is unlikely that a small residual can be achieved when the degree is overestimated. This is however not a problem when considering errors in both A and b where it is possible to produce vanishingly small residuals for all polynomials in the test set.

Figures 7.4 (a) and (b) show the normalised residuals for the initial and refined GCDs where in (a) the theoretically exact degree has been given and in (b) the degree

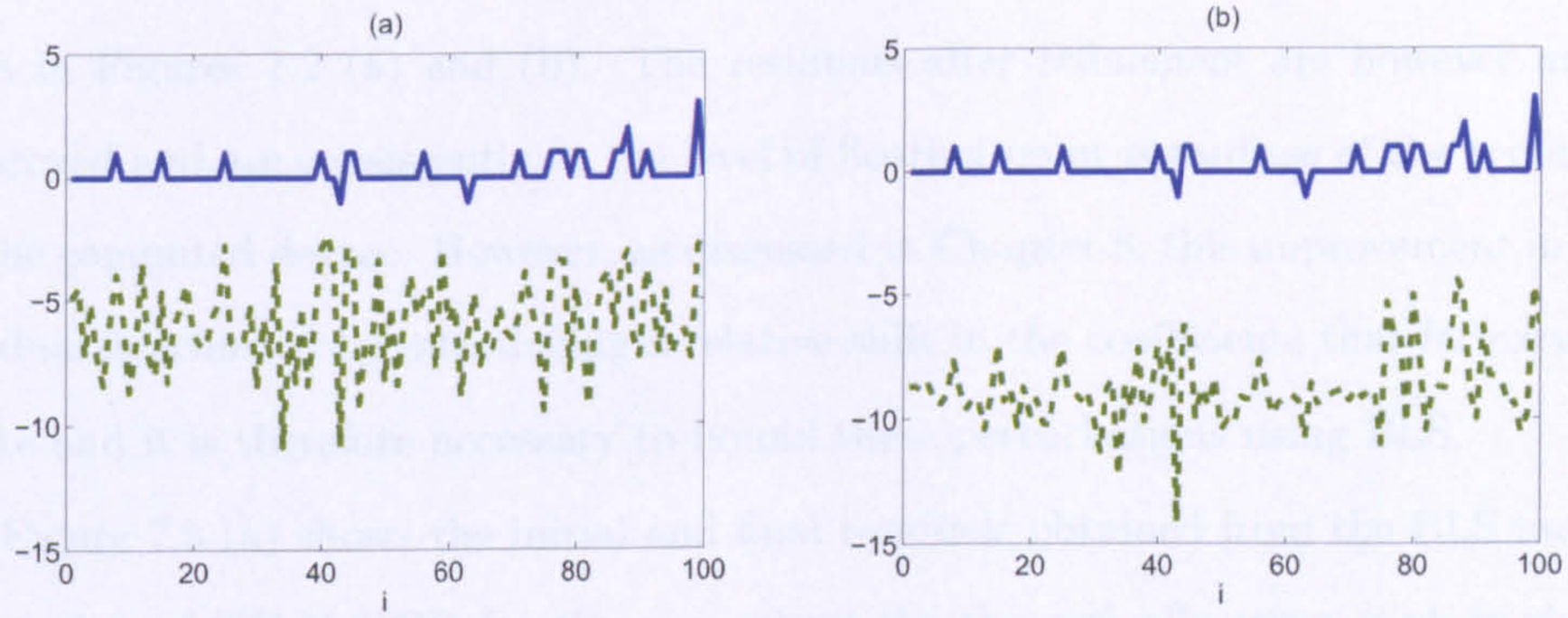

Figure 7.3: *Calculated degree of GCD − The theoretically exact degree* = — and its effect on (a) The initial residual = - -; (b) The refined residual = - -.

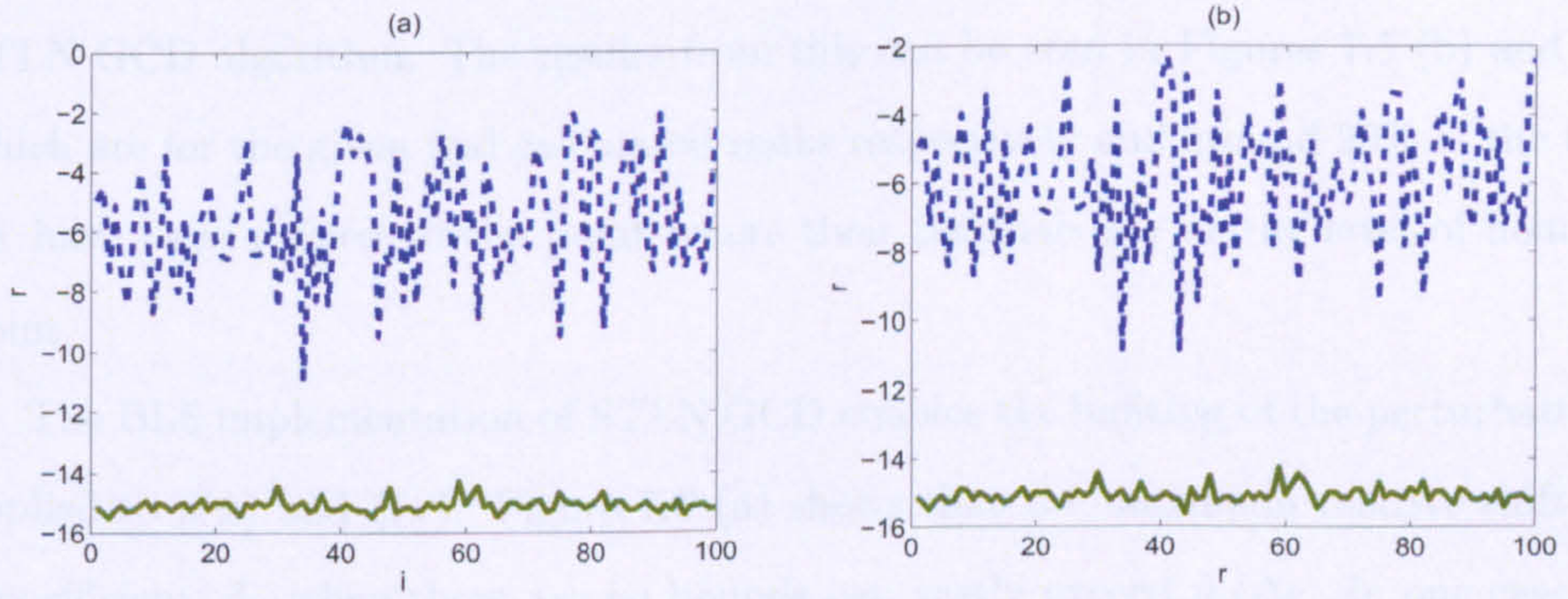

Figure 7.4: The normalised residual of the coefficients of the initial estimate = - - and after the refinement = — when considering errors in A and b. (a) Given theoretically exact rank; (b) Rank calculated using `MLrank()`.

has been calculated using `MLrank()`. Since the initial approximations are obtained in the same way, regardless of the refinement method, they are identical to those seen in Figures 7.2 (a) and (b). The residuals after refinement are however much improved and are consistently at the level of floating point regardless of the accuracy of the computed degree. However, as discussed in Chapter 6, this improvement in the residual is achieved by introducing a relative shift in the coefficients that far exceeds $1/\Delta e$ and it is therefore necessary to bound these perturbations using BLS.

Figure 7.5 (a) shows the initial and final residuals obtained from the BLS implementation of STLN GCD for the case where the theoretically exact rank is given. As can be seen the algorithm only provides a significant improvement for two of the polynomial pairs and in most cases provides no improvement at all. This is due to the initial estimate being too inaccurate causing the algorithm to fail to converge. Hence, in order to obtain acceptable results the initial estimate must first be refined by considering errors only in A which can be used as the starting point for the BLS STLN GCD algorithm. The results from this can be seen in Figures 7.5 (b) and (c) which are for the given and calculated ranks respectively and around 20% of the test set have been refined to the point where their residuals are at the level of floating point.

The BLS implementation of STLN GCD enables the limiting of the perturbations applied to $\tilde{p}(x)$ and $\tilde{q}(x)$. Figure 7.6 (a) shows that the maximum relative shift on a coefficient, $\tilde{p}_i$, when there are no bounds can vastly exceed $\tilde{p}_i/\Delta e$. In one case by as much as 12 orders of magnitude. This problem is not as extreme in Figure 7.6 (b) due to the bound constraints. However, it is also apparent that the bounds, whilst constraining the perturbations to some extent, are not keeping them within the region

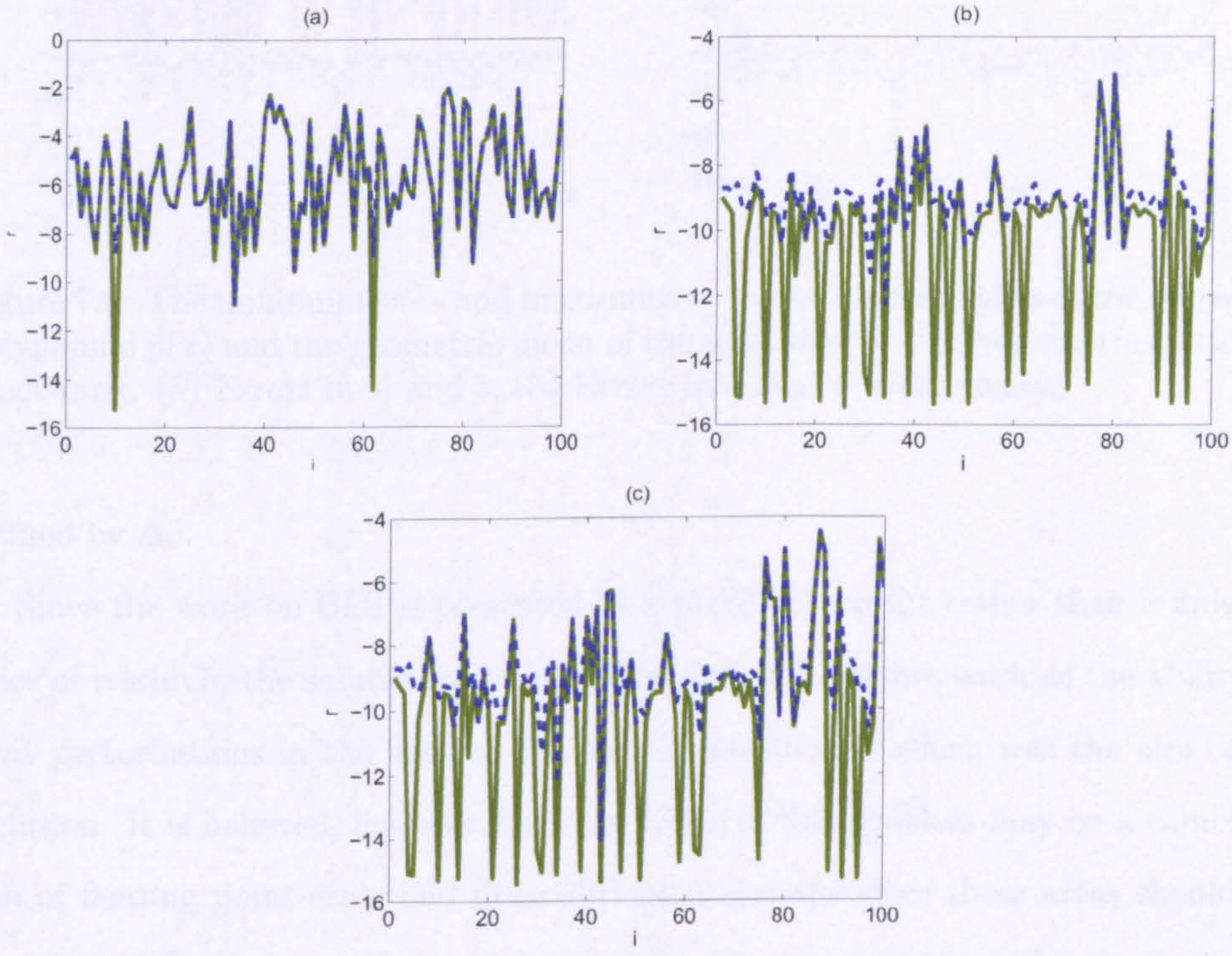

Figure 7.5: The normalised residual of the coefficients of the initial estimate = - - and after the refinement = — when considering errors in A and b with bounds. (a) Given theoretically exact rank; (b) Given theoretically exact rank and using the output from errors only in A as the initial estimate; (c) Rank calculated using `MLrank()` and using the output from errors only in A as the initial estimate.

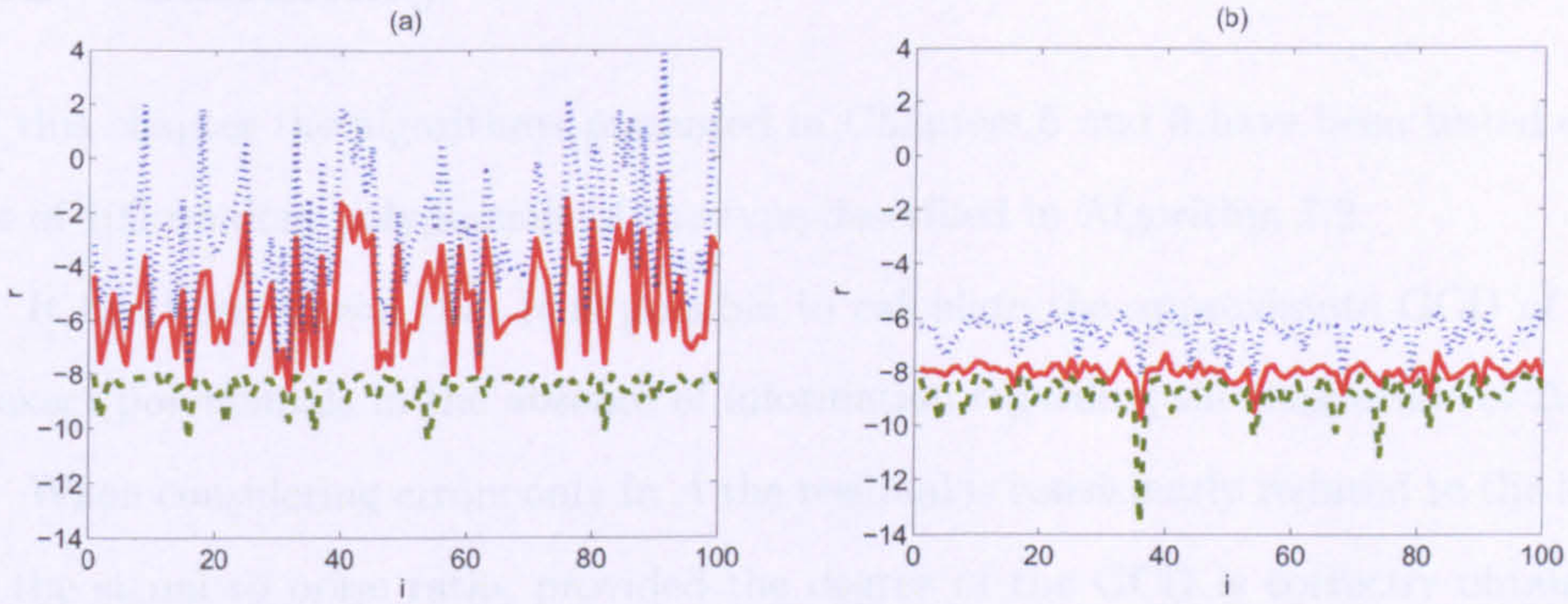

Figure 7.6: The minimum = - - and maximum = $\cdots$ normalised shifts of the corrected polynomial $\hat{p}(x)$ and the geometric mean of the total shift = — given the theoretically exact rank. (a) Errors in A and b; (b) Errors in A and b with bounds.

defined by Δe.

Since the work on BLS is presented as a proof of concept rather than a finished piece of research, the solution to this problem is left for future work as the ability to limit perturbations in the system has been demonstrated which was the aim of its inclusion. It is believed, however, that the cause of this problem may be a combination of floating point error and ill-conditioning and therefore these areas should be investigated first. Any refinement is hoped to also improve the residuals such that they are consistently smaller than those in Figures 7.2 (a) and (b) with the possibility of vanishingly small residuals such as those shown in Figures 7.4 (a) and (b).

In addition to showing the maximum and minimum relative shifts of a coefficient Figures 7.6 (a) and (b) also show the geometric mean of the relative shift of in the coefficients and it is interesting to note that with BLS this mean is within the level of the signal to noise ratio and consequently, when considered in the normwise sense, the bounding on the coefficients is effective.

7.2 Summary

In this chapter the algorithms presented in Chapters 5 and 6 have been tested on a set of 100 random polynomials of the type described in Algorithm 7.2.

It has been shown that it is possible to calculate the approximate GCD of two inexact polynomials in the absence of information regarding the magnitude of Δe.

When considering errors only in A the residual is consistently reduced to the level of the signal to noise ratio, provided the degree of the GCD is correctly obtained. For errors in both A and b the refined residual is within the level of floating point error but the shift on the coefficients is unacceptably large. This is addressed to some extent by the introduction of a basic BLS algorithm which restricts the perturbations that can be applied to the given polynomials. This is however ineffective if inaccurate initial estimates are given. Additionally the bounds may be violated by up to around two orders of magnitude. This has been highlighted as an area for further research.

In the next chapter the use of repeated GCD factorisations to reveal the multiplicity structure of a polynomial and its derivative is examined.

Chapter 8

Revealing a multiplicity structure

In the previous chapter an approximate GCD algorithm for calculating $\mathrm{GCD}(\tilde{p}(x), \tilde{q}(x))$, using the methods developed in Chapters 5 and 6, has been presented and tested. In this chapter the more specific case of calculating the GCD of a polynomial and its derivative is considered. The viability of the use of repeated iterations of this algorithm to determine the multiplicity structure of a polynomial is then explored.

8.1 Calculating the GCD of a polynomial and its derivative

Given a polynomial $p(x)$ it is possible to calculate $d(x) = \mathrm{GCD}(p(x), p^{(1)}(x))$ using the rank of a resultant matrix, a triangular matrix decomposition and the method of STLN. This may be achieved as follows.

Algorithm 8.1: Computing the approximate GCD of a polynomial and its derivative

Input: The coefficient vector $p \in \mathbb{R}^{m+1}$ of the polynomial $p(x)$.

1. Construct the resultant matrix $R = R(p(x), p^{(1)}(x))$.

2. Compute the singular values of R and calculate its rank, r, using ML, as detailed in Chapter 5.

3. Perform an upper triangular decomposition on R and obtain the coefficients of the initial estimate of the approximate GCD.

4. Refine the estimate using the method of STLN as seen in Chapter 6.

Output: The coefficient vector d of the approximate GCD.

The GCD algorithm, as described, has been tested on a large number of polynomials and has been found to produce excellent results on very noisy polynomials. Increasing the degree of the polynomials makes the algorithm slightly less stable. This is as expected, since increasing the degree of the polynomial generally causes an increase in the condition number of the corresponding resultant matrix. This is significant since calculating the rank of the matrix causes the most problems. In order to illustrate this and to further assess the algorithms performance, a representative test set of 6 polynomials has been chosen.

Polynomial test set

$$
\begin{aligned}
p_1(x) &= (x-0.1)^4(x-0.4)^3(x-0.7)^2(x-1) \\
p_2(x) &= (x-0.1)^8(x-0.4)^6(x-0.7)^4(x-1)^2 \\
p_3(x) &= (x-0.1)^{16}(x-0.4)^{12}(x-0.7)^8(x-1)^4 \\
p_4(x) &= (x-0.1)^7(x-0.9)^7(x-1)^7 \\
p_5(x) &= (x-0.1)^7(x-0.99)^7(x-1)^7 \\
p_6(x) &= (x-0.1)^7(x-0.999)^7(x-1)^7
\end{aligned}
$$

As previously stated, the calculating of the rank of the resultant matrix is crucial to obtaining the approximate GCD. This is due to the fact that its output is discrete and consequently is either correct or incorrect as opposed to the other stages whose performance may be assessed by a residual. An incorrect rank results in the degree of the GCD being incorrect and consequently every other stage is incorrect. The effect of increasing the degree of a polynomial, reducing its root separation and increasing the noise in the coefficients on the calculated degree of the GCD can be seen in Table 8.1.

It can be seen that the correct degree of $\mathrm{GCD}(p_i, p_i^{(1)})$, $p_i = p_i(x)$ has been computed for all the polynomials in the absence of noise and only $\mathrm{GCD}(p_3, p_3^{(1)}(x))$ is incorrectly computed when $\Delta e = 10^{10}$. Increasing the noise so that $\Delta e = 10^8$ results in only GCDs corresponding to p_1, p_2, p_4 being calculated correctly which is expected

(a)	(b)	(c)	(d)	(e)	(f)
p_1	6	6	6	6	9.3×10^{22}
p_2	16	16	16	16	1.4×10^{33}
p_3	36	36	32	28	2.4×10^{50}
p_4	18	18	18	18	7.1×10^{30}
p_5	18	18	18	15	8.8×10^{29}
p_6	18	18	18	19	1.6×10^{31}

Table 8.1: (a) Polynomial $p_i = p_i(x)$; (b) The theoretically exact degree of $\mathrm{GCD}(p_i, p_i^{(1)})$; (c) The calculated degree of $\mathrm{GCD}(p_i, p_i^{(1)})$ with no noise; (d) The calculated degree of $\mathrm{GCD}(p_i, p_i^{(1)})$ with $\Delta e = 10^{10}$; (e) The calculated degree of $\mathrm{GCD}(p_i, p_i^{(1)})$ with $\Delta e = 10^{8}$; (f) The condition number of $B(p_i, p_i^{(1)})$.

since two of the root clusters in p_5 and p_6 are extremely close, causing instability as shown in Chapter 2, whilst the Bezoutian associated with p_3 has a condition number of 2.4×10^{50}. However, in spite of these failings the algorithm's performance is still good given that it is given no information about Δe. In contrast, under the same conditions MatLabs `rank()` correctly computes the degree of all the GCDs in the absence of noise, but fails for all polynomials when $\Delta e = 10^{10}$ and $\Delta e = 10^{8}$.

Clearly, testing the performance of the rest of the algorithm when the calculated degree of the GCD is incorrect is of limited value. Consequently, in order to provide better analysis, the correct degrees of the theoretically exact GCDs are input into stages three and four of Algorithm 8.1 and the obtained normalised residuals examined.

Table 8.2 shows the obtained residuals for the different methods described in Chapter 6 when the coefficients are unperturbed by noise whilst Tables 8.3 and 8.4 show the residuals when the coefficients have been perturbed by a signal to noise ratio of 10^{10} and 10^{8} respectively. Column (c) shows a clear increase in the residuals of the

(a)	(b)	(c)	(d)	(e)	(f)	(g)
p_1	6	5.842e-015	1.3183e-016	5.8683e-018	2.5542e-016	3.1157e-016
p_2	16	2.6523e-012	1.5738e-016	6.5015e-019	2.9661e-016	6.2173e-016
p_3	36	9.6958e-006	4.2684e-009	1.8341e-019	1.4756e-008	4.5144e-007
p_4	18	8.6633e-007	2.6221e-016	1.3562e-016	6.8382e-015	1.2792e-011
p_5	18	1.8235e-005	2.5119e-012	5.0633e-016	0.00041598	3.9462e-006
p_6	18	5.608e-005	2.4487e-007	2.5566e-016	0.00045333	4.3716e-006

Table 8.2: No noise; (a) Polynomial p_i; (b) The theoretically exact degree of $\mathrm{GCD}(p_i, p_i^{(1)})$; (c) The normalised residual of the initial estimate from stage three; (d) The normalised residual considering errors in A in stage four; (e) The normalised residual considering errors in A and b; (f) The normalised residual using an enforced derivative constraint; (d) The normalised residual using an enforced derivative constraint and BLS.

(a)	(b)	(c)	(d)	(e)	(f)	(g)
p_1	6	5.4817e-010	6.2438e-012	1.3304e-017	1.3475e-016	1.934e-015
p_2	16	5.0985e-007	5.494e-011	1.463e-018	3.0557e-013	5.6192e-010
p_3	36	1.8704e-006	5.0477e-007	1.8716e-018	3.4786e-007	5.9894e-007
p_4	18	0.00089324	1.9968e-011	3.4513e-016	2.1067e-007	0.00065703
p_5	18	0.00092698	2.8452e-007	2.5117e-016	8.933e-009	0.00070851
p_6	18	0.00078617	1.6641e-007	3.0379e-016	1.2286e-009	0.00068757

Table 8.3: $\Delta e = 10^{10}$; (a) Polynomial p_i; (b) The theoretically exact degree of $\mathrm{GCD}(p_i, p_i^{(1)})$; (c) The normalised residual of the initial estimate from stage three; (d) The normalised residual considering errors in A in stage four; (e) The normalised residual considering errors in A and b; (f) The normalised residual using an enforced derivative constraint; (d) The normalised residual using an enforced derivative constraint and BLS.

(a)	(b)	(c)	(d)	(e)	(f)	(g)
p_1	6	4.5768e-008	2.672e-010	7.1983e-018	4.7193e-016	6.2154e-015
p_2	16	1.3814e-005	3.5642e-008	2.386e-018	5.9328e-007	2.1952e-006
p_3	36	2.9243e-005	6.2166e-006	1.2349e-016	1.0981e-005	2.5402e-005
p_4	18	3.9737e-005	8.3943e-008	3.5008e-016	1.4639e-009	3.8595e-005
p_5	18	4.171e-005	1.9985e-007	3.9778e-016	1.1112e-009	4.1488e-005
p_6	18	3.9215e-005	2.1018e-007	3.0977e-016	7.3508e-005	3.8579e-005

Table 8.4: $\Delta e = 10^8$; (a) Polynomial p_i; (b) The theoretically exact degree of $\text{GCD}(p_i, p_i^{(1)})$; (c) The normalised residual of the initial estimate from stage three; (d) The normalised residual considering errors in A in stage four; (e) The normalised residual considering errors in A and b; (f) The normalised residual using an enforced derivative constraint; (d) The normalised residual using an enforced derivative constraint and BLS.

initial estimate as the amount of noise increases as does column (d) which shows the residuals for errors only in A. This is as expected since the equality constraint in the LS problem, coupled with the inability to perturb b, results in there not being an exact solution to $Ax = b$. Consequently, a residual of approximately the same size as $\frac{1}{\Delta e}$ is to be expected and the fact that p_3, p_5 and p_6 have residuals that are considerably larger than $\frac{1}{\Delta e}$ is largely attributable to the fact that they are more unstable than p_1, p_2 and p_4. Indeed, it should be noted that these polynomials resulted in the worst performance of the rank algorithm. This trend of larger residuals is observed throughout the columns of Tables 8.2, 8.3 and 8.4 with the exception of column (e) which performs exceptionally well for all polynomials due to it being the least constrained of all the methods. This does not however mean that STLN considering errors in A and b provides the best solution since it does not require that $b = [p_i, q_i]^T \equiv [p_i, p_i^{(1)}]$ or that the solution lies with in a set space defined by Δe. This is especially important in the case where it is required to perform repeated GCD operations in

order to calculate the multiplicity structure of a polynomial. This is due to needing to maintain the relationship between each successive GCD calculation. Since perturbing the entries in b at each stage, even within a set solution space, will destroy the relationship between successive stages, only errors in A will be considered in the next section which presents an algorithm for calculating the multiplicity structure of a polynomial. This is then tested on the test set of polynomials.

8.2 Calculating the multiplicity structure of a polynomial

As discussed in Chapter 3, the multiplicity structure of a polynomial can be achieved through repeated GCD calculations as seen in Algorithm 8.2.

Algorithm 8.2: Obtaining a multiplicity structure

Input: The coefficient vector $p \in \mathbb{R}^{m+1}$ of the polynomial $p(x)$.

1. Set $d_0(x) = p(x)$, $i = 1$

   REPEAT

   - Construct the resultant matrix $R = R(d_{i-1}(x), d_{i-1}^{(1)}(x))$.
   - Compute the singular values of R and calculate its rank, r, using ML, as detailed in Chapter 5.
   - Perform an upper triangular decomposition on R and obtain the coefficients of the initial estimate of the approximate GCD.

- Refine the estimate using the method of STLN and set it equal to $d_i(x)$.
- Set $i = i + 1$.

UNTIL $d_i(x)$ is a constant term.

2. Extract multiplicity structure, as detailed in Chapter 4.

Whilst being theoretically correct, the ill-conditioned nature of the polynomials, coupled with the introduction of noise into the coefficients means that certain adaptations need to be made to Algorithm 8.2 in order for it to provide good results. In particular, the calculation of the rank of the resultant matrix, which caused problems in the previous section, is particularly crucial since an incorrect[1] output in any one of the iterations will cause an incorrect multiplicity structure to be obtained. This problem is addressed by using two control parameters in order reduce the chance of an inaccurate rank being computed.

- **Parameter 1:** In a sequence of GCD calculations, $i = 1, \ldots$ the rank of the resultant matrix, r_i, at the ith iteration must be less than or equal to the rank of the resultant matrix at the $(i-1)$th iteration. This is because the rank denotes the number of distinct roots in the polynomial which clearly cannot increase from one iteration to another. If $r_i > r_{i-1}$ set $r_i = 1$.

Proceed to parameter 2.

- **Parameter 2:** Given a computed rank r_i compute the initial approximation of the GCD and from this compute the normalised residual, res_{norm}, of (6.6).

[1] An *incorrect* multiplicity structure is taken to be one that differs from the theoretically exact one even though it may be a valid solution within the error space.

If res_{norm} is less than some tolerance θ then continue to refine the GCD. If $\text{res}_{\text{norm}} > \theta$ then $r_i = r_i + 1$ and the process is repeated until $r_i = r_{i-1}$. At this point if $\text{res}_{\text{norm}} > \theta$ then report failure. This is because if the normalised residual is too large then the degree of the GCD is likely to have been overestimated, and thus the rank of the resultant matrix has been underestimated. Hence, the rank is increased in order to reduce the normalised residual.

Clearly the setting of the tolerance θ should be data driven. However, it has been assumed that there is no knowledge of Δe and consequently, other than basing it upon the condition number of the resultant matrix, there is no obvious parameter on which to choose its value. In addition to this, it can be assumed that Δe increases with each successive GCD calculation and therefore some growth factor should be attached to θ, as is done in [42].

How to correctly set θ, if there is no knowledge of Δe, is an important problem that should certainly be the subject of further work. One possibility would be to use (5.10) to calculate the coefficients of a model of the theoretically exact singular values of the resultant matrix similar to (5.11). From this it is possible to calculate the values of the theoretically exact singular values and consequently the level of noise. This would however, require an extremely good model, the selection of which is a non-trivial task.

Not being able to accurately set θ severely limits the capabilities of the algorithm since with each successive GCD calculation the rank algorithm becomes more likely to fail due to increased errors in the computed GCD. Indeed, it is found that the algorithm generally computes the rank in the initial stages accurately. However, as discussed previously, the rank only needs to be calculated incorrectly at one stage to

(a)	(b)	(c)	(d)
p_1	✓	✓	✓
p_2	✓	✓	Fail on 4th iteration
p_3	Fail on 3rd iteration	Fail on 2nd iteration	Fail on 1st iteration
p_4	✓	Fail on 3rd iteration	Fail on 3rd iteration
p_5	Fail on 3rd iteration	Fail on 3rd iteration	Fail on 2nd iteration
p_6	Fail on 3rd iteration	Fail on 2nd iteration	Fail on 2nd iteration

Table 8.5: Performance of Algorithm 8.2. ✓= multiplicity structure calculated correctly; (a) Polynomial $p_i = p_i(x)$; (b) No noise; (c) $\Delta e = 10^{10}$; (d) $\Delta e = 10^8$.

cause a failure. The polynomial p_3, for example, requires 15 GCD iterations if the correct multiplicity structure is determined. This is a clear problem as even in the absence of noise in the coefficients of the given polynomial, the resultant matrices are very ill conditioned and the introduction of errors at each stage only serves to exacerbate this problem.

Consequently, the results from testing Algorithm 8.2 on the test set of polynomials, shown in Table 8.5, are not as good as might be hoped. However, a proof of concept is certainly made and it still exhibits good results on polynomials of around degree 20 which are still considered to be non-trivial.

The value of θ chosen for the experiments was taken as 10^{-5} and the growth factor φ, which allows for an increase in the errors in the coefficients at each stage of the process, was set equal to 0.3. This value was chosen somewhat arbitrarily as the one that gave the best results. Consequently, the tolerance θ_i at the ith iteration is $\theta_i = 10^{-(5\times 0.3i)}$. These values have been chosen somewhat arbitrarily although they seem to be the most generally applicable out of those tried. Clearly it is possible to obtain correct multiplicity structures for all the polynomials in the test set for all levels of noise through manual adjustment of θ_i. This however is unrealistic and

therefore was not attempted.

It is apparent that there is certainly scope for improvements to Algorithm 8.2 since the inaccuracies introduced in the successive iterations make calculating the rank of the resultant matrix increasingly problematic.

8.3 Summary

In this chapter experimental results have been shown for both a single GCD problem and the combination of multiple GCD operations in order to discover the multiplicity structure of a polynomial.

While the single GCD problem provides solutions of the correct degree with low residuals, even in the presence of noise, it has been shown that calculating the degrees of successive GCDs becomes increasingly difficult. Additionally, a method for selecting a value for the control parameter θ, in the absence of information regarding the magnitude of Δe, is unclear.

One method, as previously discussed, would be to calculate an estimate of Δe from the ML expression (5.10). This would however require a suitable model for the theoretically exact singular values to be developed. Two possibilities would be to base the model on either splines, or an exponential curve as both of these would provide a much better approximation to the singular value profile than the polynomial model suggested by Zarowski [41].

Another improvement would be to combine the successive GCD operations using STLN as suggested at the end of Chapter 6. This would allow the dependencies between each successive stage to be enforced and maintained which could stabilise the underlying multiplicity structure and consequently allow the degrees of the GCDs

to be better calculated.

In the next chapter the methods developed in this thesis and suggested future work are summarised.

Chapter 9

Conclusions and future work

The work presented in this thesis has involved the development of two algorithms which can be used in the solving of the approximate GCD problem.

The first algorithm uses the principle of MLE to calculate the rank of a noisy matrix which is a novel development in an area of mathematics that has primarily relied on linear algebra. The experiments detailed in Chapter 5 show that it offers a good, and in many cases superior, method for calculating the rank of a resultant matrix. This is particulary true for the case where inaccurate or no information is available regarding the magnitude of the signal to noise ratio.

The second algorithm takes an initial estimate of the coefficients of an approximate GCD and refines them using the method of STLN. By preserving any inherent relationships between the elements of the system, STLN allows only "valid" solutions to be obtained i.e. in standard TLS a solution may be obtained that doesn't have the required relationships between its elements. This has been done for errors in A; errors in A and b; errors in A and b with a derivative constraint and errors in A and b with bounds. It has been shown that the different implementations of this algorithm are

successful at calculating the approximate GCD of two noisy polynomials. However, the method involving BLS still requires further work in order to prevent the bounds from being violated.

In Chapter 8 the rank and GCD algorithms are tested on the case where it is necessary to calculate $\text{GCD}(p(x), p^{(1)}(x))$, where $p(x)$ contains multiple roots. Both algorithms performed successfully on the test set. However, when an attempt was made to combine successive GCD operations it was found that the algorithm failed after a few iterations due to errors. This was due to the rank of the resultant matrix being incorrectly computed, and consequently a multiplicity structure that was different to the theoretically exact one being calculated.

Future work and improvements to the algorithms developed in Chapters 5 and 6 have been suggested at the end each chapter. It is believed that if these changes were made, results could be improved significantly.

Bibliography

[1] J. E. Hopcraft A. V. Aho and J.D Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, USA, 1974.

[2] S. Barnett. *Polynomials and Linear Control Systems.* Marcel Dekker, New York, USA, 1983.

[3] A. Björck. *Numerical Methods for Least Squares Problems.* SIAM, Philadelphia, USA, 1996.

[4] X. Ma C. J. Zarowski and F. W. Fairman. QR-factorization method for computing the greatest common divisor of polynomials with inexact coefficients. *IEEE Trans. on Signal Processing*, 48(11):3042 3051, 2000.

[5] P. Chin, R. M. Corless, and G. F. Corliss. Optimization strategies for the approximate GCD problem. In *ISSAC '98: Proceedings of the 1998 international symposium on Symbolic and algebraic computation*, pages 228 235, New York, NY, USA, 1998. ACM.

[6] R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt. The singular value decomposition for polynomial systems. In *ISSAC '95: Proceedings of the 1995*

international symposium on Symbolic and algebraic computation, pages 195 207, New York. NY, USA, 1995. ACM.

[7] I. Z. Emiris, A. Galligo, and H. Lombardi. Certified approximate univariate GCDs. *J. Pure and Applied Algebra*, 117 and 118:229-251, 1997.

[8] R. T. Farouki and T. N. T. Goodman. On the optimal stability of the Bernstein basis. *Mathematics of Computation*, 65(216):1553-1566, 1996.

[9] R. T. Farouki and V. T. Rajan. On the numerical condition of polynomials in Bernstein form. *Computer Aided Geometric Design*, 4:191-216, 1987.

[10] R. T. Farouki and V. T. Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5:1-26, 1988.

[11] G. H. Golub and C. F. Van Loan. *Matrix Computations.* John Hopkins University Press, Baltimore, USA, 1996.

[12] P. Grünwald. A tutorial introduction to the minimum description length principle. `http://www.grunwald.nl`, 2005. [Accessed: May 2008].

[13] N. J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, Philadelphia, USA, 2002.

[14] B. F. Hildebrand. *Introduction to numerical analysis: 2nd edition.* Dover Publications, Inc., New York, NY, USA, 1987.

[15] C. M. Hoffmann, G. Park, J.-R. Simard, and N. F. Stewart. Residual iteration and accurate polynomial evaluation for shape-interrogation applications. In *SM*

'04: Proceedings of the ninth ACM symposium on Solid modeling and applications, pages 9–14, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[16] V. Hribernig and H. J. Stetter. Detection and validation of clusters of polynomial zeros. *Journal of Symbolic Computation*, 24:667–681, 1997.

[17] W. Kahan. Conserving confluence curbs ill-condition. Technical report, Department of Computer Science, University of California, Berkeley, USA, 1972.

[18] N. Karcanias and M. Mitrouli. A matrix pencil based numerical method for the computation of the GCD of polynomials. *IEEE Trans. Automatic Control*, (39):977–981, 1994.

[19] N. Karmarkar and Y. N. Lakshman. Approximate polynomial greatest common divisors and nearest singular polynomials. In *ISSAC '96: Proceedings of the 1996 international symposium on Symbolic and algebraic computation*, pages 35–39, New York, NY, USA, 1996. ACM.

[20] D. E. Knuth. *The Art of Computer Programming, Vol 2.* Addison-Wesley, Reading, USA, 1969.

[21] K. Konstantinides and K. Yao. Statistical analysis of effective singular values in matrix and rank determination. *IEEE Trans. Acoust., Speech, Signal Processing*, 36:757–763, 1988.

[22] T. Y. Li and Zhonggang Zeng. A rank-revealing method with updating, downdating, and applications. *SIAM J. Matrix Anal. Appl.*, 26(4):918–946, 2005.

[23] B. Liang and S. Unnikrishna Pillai. Blind image deconvolution using a robust GCD approach. *IEEE Transactions on Image Processing*, 8(2):295 301, 1999.

[24] 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.

[25] M.Lang and B. C. FrenZel. Polynomial root finding. *IEEE Signal Processing Letters*, 1(10):141–143, 1994.

[26] M. T. Noda and T. Sasaki. Approximate GCD and its application to ill-conditioned linear algebraic equations. *Journal of Computational and Applied Mathematics*, 38:335 -351, 1991.

[27] Victor Y. Pan. Approximate polynomial GCDs, pade approximation, polynomial zeros and bipartite graphs. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 68 77, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[28] N. M. Patrikalakis. Keynote lecture: Shape Interrogation. In *Proceedings of the Eighth ACM Symposium on Solid Modeling and Applications 2003, Seattle, Washington, USA, June 16 - 20, 2003*, page 1, 2003.

[29] J Rissanen. Modelling by the shortest data description. *Automatica*, 14:465 471, 1978.

[30] J. B. Rosen, H. Park, and J. Glick. Total least norm formulation and solution for structured problems. *SIAM J. Matrix Anal. Appl.*, 17(1):110 126, 1996.

[31] N. Saito. *Wavelets in Geophysics: Simultaneous noise suppression and signal compression using a library of orthonormal bases and the minimum description length criterion.* Academic Press, Boston, MA, 1994.

[32] T. Sederberg and G. Chang. Best linear common divisors for approximate degree reduction. *Computer Aided Design*, 25:163–168, 1993.

[33] M. Spencer. *Polynomial Real Root Finding in Bernstein Form.* PhD thesis, Brigham Young University., 1994.

[34] J. V. Uspensky. *Theory of Equations.* McGraw-Hill, New York, USA, 1948.

[35] J. Wilkinson. The evaluation of zeros of ill-conditioned polynomials, part 1 and 2. *Numerische Mathematik*, 1:150–180, 1959.

[36] J. Wilkinson. *Rounding Errors In Algebraic Processes.* Prentice-Hall, Englewood Cliffs,N.J., USA, 1963.

[37] J. R. Winkler and J. D. Allan. Structured low rank approximations of the sylvester resultant matrix for approximate GCDs of Bernstein polynomials. *Submitted to Electronic Transactions on Numerical Analysis*, 2007.

[38] J. R. Winkler and J. D. Allan. The application of the principle of maximum likelihood for estimating the rank of a matrix. *Submitted to SIAM J. Matrix Analysis*, 2008.

[39] 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.

[40] S. Wolfe. When bad things happen to good cad users. `http://www.iti-oh.com/Education/Articles/BadThingsCad.htm`. [Accessed: May 2008].

[41] C. J. Zarowski. The MDL criterion for rank determination via effective singular values. *IEEE Trans. on Signal Processing*, (46):1741- 1744, 1998.

[42] Z. Zeng. A method computing multiple roots of inexact polynomials. In *ISSAC '03: Proceedings of the 2003 international symposium on Symbolic and algebraic computation*, pages 266- 272, New York, NY, USA, 2003. ACM.

[43] Z. Zeng. Apatools: A software toolbox for approximate polynomial algebra. 2007. Preprint.

Appendix A

The Bernstein Basis

Bernstein Polynomials: A degree n polynomial in Bernstein form is given by,

$$p(x) = \sum_{i=0}^{n} b_i B_i^{(n)}(x), \qquad B_i^{(n)}(x) = \binom{n}{i}(1-x)^{n-i}x^i, \qquad \binom{n}{i} = \frac{n!}{i!(n-i)!}, \quad \text{(A.1)}$$

and is know to be optimally stable on the unit interval [8]. This, however, in no way restricts generality as the extension of this to an arbitrary interval $\hat{I} = [\alpha, \beta]$ is easily achieved using the linear transform,

$$t = \frac{x - \alpha}{\beta - \alpha}, \quad \text{(A.2)}$$

where $\alpha \leq x \leq \beta$.
Many useful geometric properties may be associated with a polynomial in Bernstein form by stating it as an explicit Bézier curve [33] and in particular such curves possess the convex hull property. See Farouki and Rajan [10] for details of algorithms for polynomials in Bernstein form.

Appendix B

Resultant matrices

Two polynomials are co-prime if and only if the determinant of their resultant matrix is equal to zero, and it is this property makes it possible to ascertain whether they have a non-constant GCD. In addition to this, the rank deficiency of the matrix gives the degree of the GCD whose coefficients may be obtained by reducing the matrix to upper triangular form.

There are several different resultant matrices, including the Sylvester and Bezoutian matrices, and they may be considered equivalent in this instance as they all yield the same information on the GCD of two polynomials.

B.1 The Bezoutian resultant matrix for power basis polynomials

Another important resultant matrix is the Bezoutian matrix [2] which is an $n \times n$ symmetric matrix, B, whose elements $e_{i,j}$ are defined by,

$$\sum_{i=1}^{n}\sum_{j=1}^{n} e_{ij}\lambda^{i-1}\mu^{j-1} = \frac{p(x_1)q(x_2) - p(x_2)q(x_1)}{x_1 - x_2}. \tag{B.1}$$

If the degree m of $q(x_1)$ is less than n then the coefficients $\{q_i\}_{i=0}^{m}$ then $b_i \rightarrow b_i + (n-m)$ and $\{q_i\}_{i=0}^{n-m+1} = 0$.

Example B.1. When $n = 3$, expanding (B.1) and comparing terms shows that,

$$B = \begin{bmatrix} |p_2q_2| & |p_1q_3| & |p_0q_3| \\ |p_1q_3| & (|p_0q_3|) + (|p_1q_2|) & |p_0q_2| \\ |p_0q_3| & |p_0q_2| & |p_0q_1| \end{bmatrix}, \tag{B.2}$$

where $p_0 = 1$, and,

$$|p_iq_j| = \begin{vmatrix} p_i & q_j \\ p_j & q_i \end{vmatrix} = p_iq_j - p_jq_i. \tag{B.3}$$

If, for example, $m = 2$ then $q(x_2) = q_1x_2^2 + q_2x_2 + q_3$ and hence $q_0 = 0$, so $|p_0q_3| = p_0q_3 - p_3q_0 = q_3$, and similarly $|p_0q_2| = q_2$ and $|p_0q_1| = q_1$.