Abstract

MAY, JOHN PAUL. Approximate Factorization of Polynomials in Many Variables and Other Problems in Approximate Algebra via Singular Value Decomposition Methods. (Under the direction of Erich Kaltofen.)

Aspects of the problem of finding approximate factors of a polynomial in many variables are considered. The idea is that a polynomial may be the result of a computation where a reducible polynomial was expected but due to introduction of floating point coefficients or measurement errors the polynomial is irreducible. Introduction of such errors will nearly always cause polynomials to become irreducible. Thus, it is important to be able to decide whether the computed polynomial is near to a polynomial that factors (and hence should be treated as reducible). If this is the case, one would like to be able to find a closest polynomial that does indeed factor. Although this problem is computable there is no known polynomial-time algorithm to find the nearest polynomial that factors.

This dissertation gives a method to find a lower bound on the distance to the nearest polynomial that factors. If this lower bound is quite large, one can conclude that the polynomial does not have approximate factors. As part of finding this bound, a linear

condition for irreducibility of polynomials from bivariate polynomials is generalized to polynomials with many variables, and a general theory of low rank approximation to extend bounds results to many different polynomial norms is given.

The singular value decomposition methods used to find the above lower bound can be used to create another method to find a nearby polynomial that factors. This method is studied, and is shown to be practical. Similar methods are also shown to work for approximate greatest common divisor computation.

The results on bounding the distance to the nearest polynomial that factors can be applied to functional decomposition of univariate polynomials. Results on functional decomposition from the 1970's together with approximate factorization results allow for a method to compute a lower bound on the distance to the nearest polynomial that has a non-trivial functional decomposition and a new algorithm to compute approximate decompositions.

APPROXIMATE FACTORIZATION OF POLYNOMIALS IN MANY VARIABLES AND OTHER PROBLEMS IN APPROXIMATE ALGEBRA VIA SINGULAR VALUE DECOMPOSITION METHODS

BY

JOHN P. MAY

A DISSERTATION SUBMITTED TO THE GRADUATE FACULTY OF
NORTH CAROLINA STATE UNIVERSITY
IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

MATHEMATICS

RALEIGH, NORTH CAROLINA

MAY 2005

APPROVED BY:

ERICH KALTOFEN
CHAIR OF ADVISORY COMMITTEE

MARK GIESBRECHT

PIERRE GREMAUD

HOON HONG

AGNES SZANTO

To my wife, family, and friends

Biography

John P. May was born in Cardston, Alberta, Canada and grew up in various places in the western US and Canada, but mostly Kindersley, Saskatchewan, Canada where he attended Kindersley Composite School. After graduating from high school in June of 1994, he moved to Oregon City, Oregon where he attended Clackamas Community College for two years, earning an Associates of Arts in June of 1996. The following September he attended the University of Oregon in Eugene, Oregon as a mathematics major active in the undergraduate mathematics society there. During the summer of 1997 he participated in an NSF funded Research Experience for Undergraduates program at Trinity University in San Antonio, Texas. Research done there would become the basis for his undergraduate honors thesis completed under the supervision of Professor Richard Koch. In June of 1998, he graduated cum laude with a Bachelor of Arts with mathematics departmental honors. He continued on at the University of Oregon to earn a Masters of Arts in mathematics in August of 1999 before moving on to the doctoral program at North Carolina State University in Raleigh, North Carolina where he worked under the supervision of Erich Kaltofen. He was awarded a Doctor of Philosophy in Pure

Mathematics in August of 2005. He is currently a post-doctoral researcher and lecturer in the School of Computer Science at the University of Waterloo in Waterloo, Ontario.

Acknowledgments

This dissertation would not have been possible without the help and support of many people. First and foremost, I am grateful to my advisor, Erich Kaltofen, for providing guidance and support and always keeping things interesting. Without him constantly challenging me none of this would have been possible. Next, I would like to thank my committee for all their comments and suggestions. I would especially like to thank Mark Giesbrecht for all the interesting conversation and thorough proofreading. I would also like to thank: Denise Seabrooks, Brenda Curin, and all the support staff in the NC State University Math department for all the little things that had to be done, everyone in the Symbolic Computation Group at the University of Waterloo for their understanding and support as I finished this dissertation way later than expected, Chris Kuster, Sandy Johnson and Clarissa for their friendship and for putting me up for the summer, Dave Bortz for his most excellent and enjoyable roommateness, Phil Jones for his adequate roommateness, Eric Choate for being my road trip buddy, Bryan Sheldon, Greg Robbins and others from the UNC Math department for letting me hang out in their offices when I took classes over there, Will Turner, Wen-shin Lee, Markus Hitz, and Austin Lobo for

knowing what it is like, and helping me get through it, and my numerous office-mates and frequent lunch companions, including George Yuhasz, Jack Perry, Dmitry Morozov, Mike Maclean, and Scott Pope, as well as many others, for good times and great tastes. Finally, I thank my family: my parents for their love and support; without them, I would never have been born let alone even dream it possible to go so far, my siblings, Seth, Tricia, Beth, Kaytie and Ben, for all being crazy enough to let me feel almost normal, and especially, my wife Jennifer for putting up with me through it all (even three years in the south), I am not sure she thought it would ever get done.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

It is possible to find reasonable partial solutions for a number of problems in approximate algebra using singular value decomposition (SVD) methods. In particular, in this dissertation, approximate factorization, approximate irreducibility testing, approximate greatest common divisor (GCD) computation, and approximate univariate decomposability testing are shown to be handled well by these methods. In general one can use SVD techniques to find approximate solutions to any problems that can be written as homogeneous linear systems.

Several types of problems in approximate algebra will be considered in this dissertation. First is the *full optimization problem*. These are problems of the form: given a set of polynomials F, find the nearest set of polynomials (possibly subject to some restrictions) with a given property $\mathfrak{P}$ (e.g. not relatively prime). Second, is what we will call the *soft approximate problem*: given a set of polynomials F find a "nearby"

set with a given property $\mathfrak{P}$ where "nearby" is intentionally vague. A slightly stronger version of the soft problem is the *ϵ-approximate problem*: given a set of polynomials and a tolerance ϵ, find a set of polynomials with property $\mathfrak{P}$ that are not father away than ϵ from the original set. SVD methods can be used to find solutions for many instances of the soft approximation problem and a partial solution for the ϵ-approximation problem. The three main problems considered herein are: 1. factorization, where one considers $F = \{f\}$, a set containing one multivariate polynomial, and $\mathfrak{P}$, the property of f being reducible, 2. GCD computation, where one considers $F = \{g, h\}$, a set of a pair of polynomials, and $\mathfrak{P}$, the property of g and h begin relatively prime, and 3. decomposition, where one considers $F = \{f\}$, the set of one univariate polynomial and $\mathfrak{P}$, the property of f begin decomposable. In all cases, one wishes to also do additional computation as well as finding the set of polynomials. For example, one usually wants to find the factors of the nearest polynomial that is not irreducible. The SVD methods lend themselves naturally this. In fact, the factors can be computed as intermediate steps.

The singular value decomposition of a matrix has much in common with the eigenvalue decomposition and is described in detail in Golub and Van Loan (1996), for example. It was shown by Eckart and Young (1936) that the SVD can be used to solve the following approximate problem in linear algebra: given a matrix A, find the nearest (in the euclidean distance) matrix $\tilde{A}$ so that $\tilde{A}$ has lower rank than A. In fact, one can find the nearest matrix with any given lower rank. Many polynomial algebra problems can be formulated as homogeneous linear systems and the SVD can be applied to these systems

to find approximate solutions to the given problem. Though it should be noted that the linear systems arising in polynomial algebra tend to be highly structured, and the SVD ignores that structure. Thus SVD techniques will seldom lead directly to solutions to the optimization version of approximate problems. One of the first uses of SVD techniques in approximate algebra was for the univariate GCD problem in Corless *et al.* (1995).

The need for new methods for factoring polynomials that are given inexactly seems to have been first recognized in Kaltofen (1985, Section 6) (that also gave one of the first polynomial time algorithms for multivariate factorization). The idea to handle the inexact case as an optimization problems was first given in Kaltofen (1992, 2000) and independently in Sasaki *et al.* (1991a). Approximate factorization of univariate polynomials does not exist independently of numerical root finding over the complex numbers because of the fundamental theorem of algebra (all polynomials have $\deg f$ linear factors). For polynomials over the rational numbers, one optimization problem arising factorization is finding the nearest polynomial with a real root. A polynomial-time solution of the nearest real root problem was given in Hitz *et al.* (1999).

The multivariate optimization problem can be formulated as follows: given $f \in \mathbb{C}[x_1, \ldots, x_n]$ that is irreducible, find the nearest polynomial $\tilde{f}$ that factors over $\mathbb{C}$ such that $\deg \tilde{f} \leq \deg f$ and $\|f - \tilde{f}\|$ is minimal. Since this can be written as a non-linear optimization problem in the unknown coefficients of a pair of factors, it is clearly possible to compute $\tilde{f}$ although the computation time could grow exponentially with the norm of f. Indeed, in Hitz *et al.* (1999) an algorithm is given that will compute nearest polyno-

mial $\tilde{f}_d$ that has a factor of total degree d using parametric least squares. Computing the solution to the parametric least squares problem is done in polynomial-time in the length of the input, but exponential in d. Other approximate factorization algorithms (Sasaki *et al.*, 1991b, 1992; Galligo and Watt, 1997; Huang *et al.*, 2000; Sasaki, 2001; Galligo and Rupprecht, 2001; Corless *et al.*, 2001, 2002; Sommese *et al.*, 2004; Gao *et al.*, 2004) consider the soft and ϵ approximate versions of factorization. Many of these previous algorithms run in polynomial time, but generally only work when f is quite near to a polynomial that factors. In this dissertation, an algorithm will be given that will find a nearby polynomial that factors for any given input polynomial.

Since the approximate factorization optimization problem cannot be solved in polynomial time, it is also useful to turn the problem around and look at approximate irreducibility testing as first considered by Nagasaka (2002) and later improved in Kaltofen and May (2003). The idea of approximate irreducibility testing is: when given an irreducible polynomial and a tolerance ϵ, decide if it is possible to apply a perturbation of size ϵ to make the given polynomial factorizable. One way to realize this test is to compute a lower bound on the size of perturbation of the coefficients of the irreducible polynomial that will leave it irreducible. Ideally one would compute the distance to the nearest polynomial to f that factors. However, it is not clear that is any easier than computing that polynomial directly. Computing a lower bound on that distance is quite possible using SVD techniques.

Both the approximate factorization and approximate irreducibility testing problems

can be stated for various notions of degree and distance. The choice of norm is clearly important, and it can be shown that the choice of degree is important as well. One can choose to consider the multi-degree or vector degree of a polynomial, that is $\mathrm{mdeg} f = (\deg_{x_1} f, \ldots, \deg x_n f)$. This is the degree used in Gao (2003), for example. A more common, and more restrictive degree choice is the total degree. If one looks for the nearest factorizable polynomial with the same rectangular degree, it will, in general, be farther away than the nearest factorizable polynomial with the same total degree. This can be seen in Section 2.3.2 in Example 2.2. More generally, one can consider other restrictions on the support of $\tilde{f}$. This is done in Section 3.2.

The optimization version of the approximate GCD problem can be stated in a way similar to the way we stated the approximate factorization problem. Given a relatively prime pair of polynomials f and g, find $\tilde{f}$ and $\tilde{g}$ so that $\tilde{f}$ and $\tilde{g}$ are not relatively prime, $\deg \tilde{f} \leq \deg f$, $\deg \tilde{g} \leq \deg g$ and $\|f - \tilde{f}\| + \|g - \tilde{g}\|$ is minimal. If the solution is not unique, one may additionally require $\deg(\gcd(\tilde{f}, \tilde{g}))$ be maximal (although, this will still not guarantee uniqueness). There is a polynomial time algorithm that solves the approximate GCD problem for univariate polynomials given in Karmarkar and Lakshman Y. N. (1998), though in practice it seems to be slow. There has also been work on algorithms to solve the soft problem of finding nearby polynomials that have a GCD (Schönhage, 1985; Corless *et al.*, 1995; Emiris *et al.*, 1997; Rupprecht, 1999; Corless *et al.*, 2004; Zhi, 2003; Zeng, 2004). The QRGCD routine in Maple's SNAP (Jeannerod and Labahn, 2002) package is a good example of such an algorithm.

There is no polynomial time solution to the multivariate approximate GCD problem. There has been much work on the soft version of the problem (Ochi *et al.*, 1991; Sasaki and Sasaki, 1997; Zhi and Noda, 2000; Zhi *et al.*, 2001; Zeng and Dayton, 2004). This dissertation will present one algorithm for the soft version of this problem that can also be used to consider the approximate relatively prime testing problem. A very similar algorithm was discovered independently in Zeng and Dayton (2004). Using ideas similar to the approximate irreducibility testing, a method can be devised for testing approximate relative primeness. This problem has been studied for univariate polynomials (Beckermann and Labahn, 1998b,a) but the multivariate problem seems to remain largely unexamined.

Another polynomial problem we can consider is functional decomposition. That is, given a univariate polynomial f, find g and h, not linear, so that $f(x) = g(h(x))$. The soft version of the approximate decomposition of polynomials was first considered in Corless *et al.* (1999) where two iterative algorithms are proposed. We consider the problem in a different way: as a special case of approximate factorization. In doing so we can compute lower bounds for the distance to the nearest polynomial which decomposes as well as computing approximate decompositions.

Finally, we also briefly examine the problem of approximate polynomial division since it is an important subproblem of both factorization and GCD computation.

Chapter 2

Approximate Irreducibility

The goal of this chapter is to establish a lower bound on the radius of irreducibility of an irreducible polynomial f over $\mathbb{C}$. The radius of irreducibility of f is the distance from f to the nearest polynomial that factors. A lower bound can be established for several norms and for the multi-degree f (and later, in Section 3.2, for total degree and other restrictions of the support of f). It should be noted that there is an easy upper bound: the radius of irreducibility can never be larger than $\|f\|$. This is simply because any given monomial (except the monomial 1) is reducible, so if $\bar{f}$ is the sum of all but one of the terms of f, then $f - \bar{f}$ is reducible and clearly $\|\bar{f}\| \leq \|f\|$.

In this chapter we will consider polynomials in $\mathbb{K}[x, y_1, \ldots, y_n]$ in order to be indicative of the fact than many of the proof techniques consider these multivariate polynomials as univariate polynomials in x over the rational function field $\mathbb{K}(y_1, \ldots, y_n)$.

2.1 An Exact Irreducibility Test

2.1.1 The Generalized Ruppert Theorem

Testing if a bivariate polynomial f is absolutely irreducible, that is irreducible over the algebraic closure of the coefficient field of f, can be reduced to a linear problem in the coefficients of f as seen in Ruppert (1999). We generalize this approach to all multivariate polynomials.

To denote the rectangular multi-degree of $f \in \mathbb{K}[x, y_1, \ldots, y_m]$, we will write $\operatorname{mdeg} f = (d, e_1, \ldots, e_m)$. That is, $\deg_x f = d$, and $\deg_{y_i} f = e_i$. This notion of degree can be partially ordered: $\operatorname{mdeg} g \leq \operatorname{mdeg} f$ means that $\deg_x g \leq \deg_x f$, and $\deg_{y_i} g \leq \deg_{y_i} f$. The inequality $\operatorname{mdeg} g < \operatorname{mdeg} f$ will mean that at least one of the inequalities is strict. We will use f_x and f_{i,y_j} as short hand for $\partial f/\partial x$ and $\partial f_i/\partial y_j$ respectively.

Theorem 2.1. *Let $f \in \mathbb{K}[x, y_1, \ldots, y_m]$ have multi-degree $(d, e_1, \ldots, e_m)$, where $\mathbb{K}$ is a field of characteristic 0. The polynomial f is absolutely irreducible if and only if the following system of partial differential equations has no polynomial solution $(g, h_1, \ldots, h_m)$:*

$$\frac{\partial}{\partial y_i}\frac{g}{f} = \frac{\partial}{\partial x}\frac{h_i}{f},\ i = 1, \ldots, m \tag{2.1}$$

under the constraints

$$\begin{array}{lll} \deg_x(g) \le (d-2) & \deg_{y_i}(g) \le e_i & i = 1, \ldots, m \\ \deg_x(h_j) \le d & \deg_{y_i}(h_j) \le \begin{cases} e_i & \text{if } i \neq j \\ e_i - 1 & \text{if } i = j \end{cases} & j = 1, \ldots, m \end{array} \tag{2.2}$$

and $g \neq 0$.

Note that degree bounds (2.2) are specifically chosen to exclude the solution

$$(f_x, f_{y_1}, \ldots, f_{y_n}).$$

Proof. First notice that if p is a non-constant factor of f then

$$(f\frac{p_x}{p}, f\frac{p_{y_1}}{p}, \ldots, f\frac{p_{y_m}}{p}) \tag{2.3}$$

is a non-trivial solution to (2.1) that satisfies the bounds (2.2), except for the bound on the degree in x of $g = f\,p_x/p$. If f has more than one factor, any linear combination of solutions such as (2.3) will satisfy (2.1).

We will prove "the polynomial f is absolutely irreducible only if (2.1) has no solution of the form (2.2)" by proving the contrapositive by contradiction. By assuming f has a non-trivial factorization over $\mathbb{C}$, we will show it has solutions to (2.1) that satisfy the bound (2.2).

Suppose f factors as $f = f_1 f_2$ over $\mathbb{C}$. If $f_{1,x} = 0$ or $f_{2,x} = 0$ then (2.3) (using $p = f_1$

or $p = f_2$ respectively) satisfies (2.2) and we are done. Otherwise, if $f_{1,x}, f_{2,x} \neq 0$ then we want to find a and b so that $g = af_2f_{1,x} + bf_1f_{2,x}$ has degree less than $(d-2)$ in x. Notice that

$$\mathrm{lc}_x(f_2f_{1,x}) = \deg_x(f_1)\,\mathrm{lc}_x(f_1)\,\mathrm{lc}_x(f_2),$$

and

$$\mathrm{lc}_x(f_1f_{2,x}) = \deg_x(f_2)\,\mathrm{lc}_x(f_1)\,\mathrm{lc}_x(f_2).$$

So, if we let $a = \deg_x(f_2) \neq 0$ and $b = -\deg_x(f_1) \neq 0$, the leading terms will cancel and

$$(af\frac{f_{1,x}}{f_1} + bf\frac{f_{2,x}}{f_2}, af\frac{f_{1,y_1}}{f_1} + bf\frac{f_{2,y_1}}{f_2}, \ldots, af\frac{f_{1,y_m}}{f_1} + bf\frac{f_{2,y_m}}{f_2}) \tag{2.4}$$

is a solution, that satisfies the degree bounds. It is necessary to check that the first entry is not 0. Without loss of generality, we can assume that f_1 and f_2 are relatively prime. Proceeding by contradiction, let us assume that the first entry of (2.4) is 0, so we have

$$f_2f_{1,x} = -(b/a)f_1f_{2,x}. \tag{2.5}$$

Since, f_2 divides the left-hand side of (2.5), it must also divide the right-hand side of (2.5). Since f_1 and f_2 are relatively prime, this implies that f_2 divides $f_{2,x}$. That implies $f_{2,x} = 0$. But we assumed $f_{2,x} \neq 0$. Hence, by contradiction, (2.4) is a solution to (2.1) and (2.2).

We will prove "the polynomial f is absolutely irreducible if (2.1) has no solution of

the form (2.2)" directly. Suppose f is irreducible, and hence square-free. Write f as a product of distinct linear factors over the algebraic closure of $\mathbb{C}(y_1, \ldots, y_m)$:

$$f = \mathrm{lc}_x(f) \prod_{i=1}^{d} x - \alpha_i \,, \;\; \alpha_i \in \overline{\mathbb{C}(y_1, \ldots, y_m)}.$$

Let us form the partial fraction decompositions of the rational functions in (2.1):

$$\frac{g}{f} = \sum_{i=1}^{d} \frac{b_i}{x - \alpha_i}, \;\; \frac{h_j}{f} = \sum_{i=1}^{d} \frac{c_{ji}}{x - \alpha_i} + \overline{h}_j$$

where $\overline{h}_j \in \mathbb{C}[y_1, \ldots, y_m]$, $b_i = \dfrac{g(\alpha_i, y_1, \ldots, y_m)}{f(\alpha_i, y_1, \ldots, y_m)} \in \overline{\mathbb{C}(y_1, \ldots, y_m)}$, and $c_{ji} \in \overline{\mathbb{C}(y_1, \ldots, y_m)}$.

Now compute the partial fraction decompositions of the derivatives:

$$\frac{\partial}{\partial y_j}\frac{g}{f} = \sum_{i=1}^{d} \frac{1}{x - \alpha_i}\frac{\partial b_i}{\partial y_j} + \frac{b_i}{(x - \alpha_i)^2}\frac{\partial \alpha_i}{\partial y_j},$$
$$\frac{\partial}{\partial x}\frac{h_j}{f} = \sum_{i=1}^{d} \frac{c_{ji}}{(x - \alpha_i)^2}.$$

If g and h_i's are a solution to (2.1) then the uniqueness of the partial fraction decomposition implies that $\frac{\partial b_i}{\partial y_j} = 0$ for all $i = 1 \ldots d$, and $j = 1 \ldots m$. Hence, $b_i \in \mathbb{C}$ for all $i = 1 \ldots d$.

Since f is irreducible, all the α_i's are algebraic conjugates. Hence, the formula for the b_i's implies that they too are algebraic conjugates. But, the b_i's are all complex numbers, so they must be equal. Thus $b_i = b \in \mathbb{C}$ for $i = 1 \ldots d$. The partial fraction

decomposition above becomes:

$$\frac{g}{f} = \sum_{i=1}^{d} \frac{b}{x - \alpha_i} = b\,\frac{f_x}{f}.$$

Hence, $g = bf_x$. But, if $\deg_x g < d - 1 = \deg_x f_x$, then b must be 0. So, there is no solution to (2.1) that satisfies (2.2) and $g \neq 0$. □

It is not hard to show that for a solution $(g, h_1, \ldots, h_n)$, $g = 0$ implies that $h_i = 0$, for $i = 1 \ldots m$. See the discussion in Section 3.1.1.

2.1.2 The Ruppert Matrix

Notice that each of the partial differential equations (2.1) can be rewritten using the quotient rule to look like

$$f\frac{\partial g}{\partial y_i} - g\frac{\partial f}{\partial y_i} + h_i\frac{\partial f}{\partial x} - f\frac{\partial h_i}{\partial x} = 0, \tag{2.6}$$

which is $4de_i \prod_{j\neq i}(2e_j + 1)$ linear equations in the $(2de_i + e_i - 1)\prod_{j\neq i}(e_j + 1)$ unknown coefficients of g and the h_is. Collecting all these linear equations we can form a matrix which we will denote as $\mathrm{Rup}(f)$, the generalized Ruppert matrix of f, with dimensions

$$\left(\sum_{i=1}^{m} 4de_i \prod_{j\neq i}(2e_j + 1)\right) \times \left(\sum_{i=1}^{m}(2de_i + e_i - 1)\prod_{j\neq i}(e_j + 1)\right),$$

or when $m = 1$, dimensions $(4de_1) \times (2de_1 + e_1 - 1)$. Thus, testing irreducibility reduces to a problem of linear algebra.

Corollary 2.2. $\mathrm{Rup}(f)$ *is a full rank matrix over* $\mathbb{C}$ *if and only if* f *is an absolutely irreducible polynomial.*

Note that any polynomial basis can be used to construct the matrix $\mathrm{Rup}(f)$. For example if a polynomial is obtained empirically, often it is given by its values at certain evaluation points; in other words, it is given in terms of a Lagrange basis and converting to a power basis can be very unstable numerically. Numerical issues notwithstanding, our examples will be given in terms of the standard power basis, exclusively.

Example 2.1. Given the polynomial

$$\varphi = c_{2,2}x^2y^2 + c_{2,1}x^2y + c_{1,2}xy^2 + c_{2,0}x^2 + c_{0,2}y^2 + c_{1,1}xy + c_{1,0}x + c_{0,1}y + c_{0,0},$$

the matrix $\mathrm{Rup}(\varphi)$ is 12×9 with zero rows removed (see Figure 2.1). If we specialize to $f = x^2 + y^2 - 1$ we get a 12×9 matrix (with two zero rows) that has rank 9 since f is absolutely irreducible. Note that the symmetry of f is not being exploited here.

For the two variable case, a general description of a block structure of the Ruppert matrix is shown in Nagasaka (2004). If $f = \sum_{i,j} c_{i,j}\, x^i y^j$ then $\mathrm{Rup}(f)$ can be written as shown in figure 2.2 with G_i and H_i block matrices with structure shown in figure 2.3 and figure 2.4 respectively. The structure given in Nagasaka (2004) is written with a different order for the monomials than that used for figure 2.1.

$$
\begin{array}{r}
 \\
1 \\
y \\
x \\
y^2 \\
xy \\
x^2 \\
xy^2 \\
x^2y \\
x^3 \\
x^2y^2 \\
x^3y \\
x^3y^2
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc}
u_{0,0} & v_{0,0} & u_{0,1} & u_{1,0} & v_{1,0} & u_{0,2} & u_{1,1} & v_{2,0} & u_{1,2}
\end{array} \\
\left[\begin{array}{ccccccccc}
-c_{0,1} & c_{1,0} & c_{0,0} & 0 & -c_{0,0} & 0 & 0 & 0 & 0 \\
-2\,c_{0,2} & c_{1,1} & 0 & 0 & -c_{0,1} & 2\,c_{0,0} & 0 & 0 & 0 \\
-c_{1,1} & 2\,c_{2,0} & c_{1,0} & -c_{0,1} & 0 & 0 & c_{0,0} & -2\,c_{0,0} & 0 \\
0 & c_{1,2} & -c_{0,2} & 0 & -c_{0,2} & c_{0,1} & 0 & 0 & 0 \\
-2\,c_{1,2} & 2\,c_{2,1} & 0 & -2\,c_{0,2} & 0 & 2\,c_{1,0} & 0 & -2\,c_{0,1} & 2\,c_{0,0} \\
-c_{2,1} & 0 & c_{2,0} & -c_{1,1} & c_{2,0} & 0 & c_{1,0} & -c_{1,0} & 0 \\
0 & 2\,c_{2,2} & -c_{1,2} & 0 & 0 & c_{1,1} & -c_{0,2} & -2\,c_{0,2} & c_{0,1} \\
-2\,c_{2,2} & 0 & 0 & -2\,c_{1,2} & c_{2,1} & 2\,c_{2,0} & 0 & -c_{1,1} & 2\,c_{1,0} \\
0 & 0 & 0 & -c_{2,1} & 0 & 0 & c_{2,0} & 0 & 0 \\
0 & 0 & -c_{2,2} & 0 & c_{2,2} & c_{2,1} & -c_{1,2} & -c_{1,2} & c_{1,1} \\
0 & 0 & 0 & -2\,c_{2,2} & 0 & 0 & 0 & 0 & 2\,c_{2,0} \\
0 & 0 & 0 & 0 & 0 & 0 & -c_{2,2} & 0 & c_{2,1}
\end{array}\right]
\end{array}
$$

Figure 2.1: The matrix of the linear equations in (2.6) for a symbolic polynomial with degree two in x and y.

$$
\left[\begin{array}{ccccccccc}
G_n & 0 & \cdots & 0 & 0 \cdot H_n & 0 & \cdots & 0 & 0 \\
G_{n-1} & G_n & \ddots & \vdots & -H_{n-1} & H_n & \ddots & \vdots & \vdots \\
\vdots & G_{n-1} & \vdots & 0 & \vdots & 0 \cdot H_{n-1} & \vdots & 0 & \vdots \\
G_1 & \vdots & \ddots & G_n & (1-n)\,H_1 & \vdots & \ddots & (n-1)\,H_1 & 0 \\
G_0 & G_1 & \ddots & G_{n-1} & -n\,H_0 & (2-n)\,H_1 & \vdots & (n-2)\,H_{n-1} & n\,H_n \\
0 & G_0 & \ddots & \vdots & 0 & (1-n)\,H_0 & \ddots & \vdots & (n-1)\,H_{n-1} \\
\vdots & \ddots & \ddots & G_1 & \vdots & 0 & \vdots & 0 \cdot H_1 & \vdots \\
0 & \ldots & 0 & G_0 & 0 & \cdots & 0 & -H_0 & H_1
\end{array}\right]
$$

Figure 2.2: The block matrix structure of $\mathrm{Rup}(f)$ from Nagasaka (2004)

$$G_i = \begin{bmatrix} 0 & 0 & \cdots & 0 & 0 & 0 \\ c_{i,m-1} & -c_{i,m} & \ddots & \vdots & \vdots & 0 \\ 2\,c_{i,m-2} & 0 & \ddots & 0 & \vdots & \vdots \\ \vdots & c_{i,m-2} & \ddots & (2-m)\,c_{i,m} & 0 & \vdots \\ \vdots & \vdots & \ddots & \vdots & (1-m)\,c_{i,m} & 0 \\ m\,c_{i,0} & \vdots & \ddots & \vdots & \vdots & -m\,c_{i,m} \\ 0 & (m-1)\,c_{i,0} & \ddots & 0 & \vdots & \vdots \\ \vdots & 0 & \ddots & c_{i,1} & -c_{i,2} & \vdots \\ 0 & \vdots & \ddots & 2\,c_{i,0} & 0 & -2\,c_{i,2} \\ 0 & 0 & \cdots & 0 & c_{i,0} & -c_{i,1} \end{bmatrix}$$

Figure 2.3: The structure of the G_i blocks of $\mathrm{Rup}(f)$

$$H_i = \begin{bmatrix} 0 & \cdots & 0 \\ c_{i,m} & \vdots & \vdots \\ c_{i,m-1} & \vdots & 0 \\ \vdots & \vdots & c_{i,m} \\ c_{i,1} & \vdots & c_{i,m-1} \\ c_{i,0} & \vdots & \vdots \\ 0 & \ddots & c_{i,1} \\ \vdots & \cdots & c_{i,0} \end{bmatrix}$$

Figure 2.4: The structure of the H_i blocks of $\mathrm{Rup}(f)$

2.2 Some Linear Algebra

2.2.1 Euclidean Distance to the Nearest Rank Deficient Matrix

Given a matrix $A \in \mathbb{C}^{\mu\times\nu}$, the Frobenius norm of A is equal to the Euclidean norm of the matrix considered as a vector and will be denoted as $\|A\|_F$. That is $\|A\|_F^2 = \sum_{i,j} |A_{i,j}|^2$. It is a well known fact of matrix theory that $\|A\|_F$ is also equal to the sum of the singular values of A.

We now state a classic linear algebra theorem from Eckart and Young (1936), which is proved for complex matrices in Stewart (1973b, Theorem 6.7).

Fact 2.3. *Let $A \in \mathbb{C}^{\mu\times\nu}$ be a matrix of rank r. If $B \in \mathbb{C}^{\mu\times\nu}$ has rank strictly less than r, then $\|A-B\|_F \geq \sigma(A)$, where $\sigma(A)$ denotes the smallest positive singular value of the matrix A. Furthermore, there exists B of rank $r-1$ so that $\|A-B\|_F = \sigma(A)$.*

Let us now suppose that f is irreducible and both $f_\triangle$ and $\tilde{f} = f - f_\triangle$ have the same degree as f. Here $\tilde{f}$ denotes the perturbed polynomial and $f_\triangle$ the perturbation. Hence $\mathrm{Rup}(f_\triangle)$ and $\mathrm{Rup}(\tilde{f})$ have the same dimensions as $\mathrm{Rup}(f)$, thus $\mathrm{Rup}(f_\triangle) = \mathrm{Rup}(f - \tilde{f})$ is equal to $\mathrm{Rup}(f) - \mathrm{Rup}(\tilde{f})$. If $\tilde{f}$ is factorizable, then $\mathrm{Rup}(\tilde{f})$ must be rank deficient. Because $\mathrm{Rup}(f)$ is of full rank and $\mathrm{Rup}(\tilde{f})$ is rank deficient, Fact 2.3 implies

$$\|\mathrm{Rup}(f_\triangle)\|_F \geq \sigma(\mathrm{Rup}(f)). \tag{2.7}$$

Note that restricting $f_\triangle$ to have the same degree as f is artificial, and we will introduce notation later that will allow $f_\triangle$ to have smaller degree.

It should be noted that although the estimate (2.7) is sharp for general matrices, due to structure of the Ruppert matrices, it may be that

$$\min_{\substack{\deg(\tilde{f})\leq\deg(f)\\ \operatorname{rank}\operatorname{Rup}(\tilde{f})<\operatorname{rank}\operatorname{Rup}(f)}} \|\operatorname{Rup}(f)-\operatorname{Rup}(\tilde{f})\|_F \gg \sigma(\operatorname{Rup}(f)).$$

It is not at all obvious how to find a sharp bound. There has been some interesting work in Rump (2003) on finding sharp bounds for more elementary matrix structures. So there is hope that it might be possible to find better bounds for the Ruppert structure as well.

2.2.2 Other Norms

This theory of nearest rank deficient matrices can be stated more generally in other norms as well. We will write $\|A\|_{p,q}$ for the matrix operator norm defined as $\max_{x\neq 0}\|Ax\|_p/\|x\|_q$ or equivalently as $\max_{\|x\|_q=1}\|Ax\|_p$. By the definition of the norm, it follows that for all x,

$$\|Ax\|_p \leq \|A\|_{p,q}\|x\|_q.$$

In general, these norms seem to be non-trivial to compute. However, when $p=q$ we get the standard matrix operator norms, some of which can be computed quite easily. The 1-norm, for example, is the largest absolute column sum:

$$\|A\|_1 = \|A\|_{1,1} = \max_j\Big\{\sum_{i,j}|A_{i,j}|\Big\}.$$

The ∞-norm is the largest absolute row sum:

$$\|A\|_\infty = \|A\|_{\infty,\infty} = \max_i \Big\{ \sum_{i,j} |A_{i,j}| \Big\}.$$

The 2-norm, $\|A\|_2$, is equal to the largest singular value of A. The height of a matrix can also be represented as one of these norms.

Lemma 2.4. $\|A\|_{\infty,1} = H(A) = \max_{i,j}\{|A_{i,j}|\}$.

Proof. Clearly $H(A) \leq \|A\|_{\infty,1}$ since if z is the unit vector with a 1 in the position of the column containing the entry of largest absolute value, and zeros elsewhere, then $\|Az\|_\infty$ is just the maximum of the absolute values of entries in that column.

Now if $\|x\|_1 = 1$, then

$$|(Ax)_i| \leq \sum_j |A_{i,j}||x_j| \leq \max_j\{|A_{i,j}|\} \sum_j |x_j| = \max_j\{|A_{i,j}|\}. \tag{2.8}$$

So

$$\|A\|_{\infty,1} = \max_{\|x\|_1=1} \|Ax\|_\infty \leq \max_{i,j}\{|A_{i,j}|\} = H(A).$$

□

There seems to be no explicit formula for $\|A\|_{1,\infty}$, but we can get an upper bound that will be useful later.

Lemma 2.5. $\|A\|_{1,\infty} \leq \sum_{i,j} |A_{i,j}|$.

Proof. If $\|x\|_\infty = 1$, then

$$\|Ax\|_1 = \sum_i \left| \sum_j A_{i,j} x_j \right| \leq \sum_i \sum_j |A_{i,j}||x_j| \leq \max_j\{|x_j|\} \sum_{i,j} |A_{i,j}| = \sum_{i,j} |A_{i,j}|. \quad (2.9)$$

Notice that the bound is achieved when A has, for example, all positive real entries and it is possible to find small examples that have norm strictly less than the bound. □

Following Campbell and Meyer (1979), we will write $A^\dagger$ to indicate the Moore-Penrose pseudo-inverse of A. That is, the unique matrix such that

(i) $AA^\dagger A = A$,

(ii) $A^\dagger AA^\dagger = A^\dagger$,

(iii) $(AA^\dagger)^H = AA^\dagger$ and

(iv) $(A^\dagger A)^H = A^\dagger A$.

For a given matrix B, by B^H we mean the Hermitian (conjugate transpose) of B. In particular, we are interested in the following property of $A^\dagger$ (see Campbell and Meyer (1979, p. 9)): if x is in the row space of A then $A^\dagger Ax = x$.

The following is a slight variation of a theorem found in Campbell and Meyer (1979, Prop. 10.4.2), which is a generalization of a theorem by Gastinel for invertible matrices (Kahan, 1966, p. 775). This is essentially Fact 2.3 for operator norms.

Lemma 2.6. *Suppose A has full rank and A has more rows than columns. If $A - A_\triangle$*

has lower rank than A, then

$$\|A_\Delta\|_{p,q} \geq 1/\|A^\dagger\|_{q,p}.$$

In this case $A^H A$ is invertible and $A^\dagger = (A^H A)^{-1} A^H$.

Proof. If $A - A_\Delta$ is rank deficient, then there is a z so that $(A - A_\Delta)z = 0$ or, $Az = A_\Delta z$. Also note that since A has full rank, its row space contains all vectors of the appropriate dimension, including z. Now compute:

$$\|A_\Delta\|_{p,q} \geq \|A_\Delta z\|_p/\|z\|_q = \|Az\|_p/\|z\|_q = \|Az\|_p/\|A^\dagger Az\|_q$$
$$\geq \|Az\|_p/(\|A^\dagger\|_{q,p}\|Az\|_p) = 1/\|A^\dagger\|_{q,p}.$$

The formula for $A^\dagger$ is classical (see, for example, Campbell and Meyer (1979, Theorem 1.3.2)). □

Note that since $\mathrm{Rup}(f)$ always has more rows than columns, Lemma 2.6 applies.

2.3 The Lower Bounds

2.3.1 The Structure of $\mathrm{Rup}(f)$

Before we establish the lower bound, we first will demonstrate a number of relationships between the norms of f and the norms of $\mathrm{Rup}(f)$.

Lemma 2.7. *All non-zero entries of $\mathrm{Rup}(f)$ are integer multiples of coefficients of f, or are equal to 0. In fact, if $f = \sum c_{i,j_1,\ldots,j_m} x^i y_1^{j_1} \ldots y_m^{j_m}$ then:*

1. *a given coefficient appears at most twice in any row of* $\mathrm{Rup}(f)$*;*

2. *a given coefficient appears at most* m *times in any column of* $\mathrm{Rup}(f)$*;*

3. *a given coefficient appears in at most*

$$\sum_{i=1}^{m} \left((2de_i - d) \prod_{j \neq i} (e_j + 1) \right)$$

(when $m = 1$*,* $2de_1 - d$*) entries of* $\mathrm{Rup}(f)$*;*

4. *if* $(\mathrm{Rup}(f))_{k,l} = ac_{i,j_1,\ldots,j_m}$*, then* $|a| \leq \max\{d, e_1, \ldots, e_m\}$.

Proof. We will look at one of the partial differential equations from which $\mathrm{Rup}(f)$ is derived:

$$f\frac{\partial g}{\partial y_n} - g\frac{\partial f}{\partial y_n} + h_n\frac{\partial f}{\partial x} - f\frac{\partial h_n}{\partial x} = 0 \tag{2.10}$$

where

$$g = \sum u_{i,j_1,\ldots,j_m} x^i y_1^{j_1} \ldots y_m^{j_m}, \text{ and } h = \sum v_{i,j_1,\ldots,j_m} x^i y_1^{j_1} \ldots y_m^{j_m}.$$

A row of $\mathrm{Rup}(f)$ comes from the equation given by setting one of the coefficients of (2.10) to zero. So, we examine the coefficient of a given $x^i y^{j_1} \ldots y^{j_m}$ in (2.10):

$$\left(\sum_{\substack{k+s=i \\ l_n+t_n=j_n+1 \\ l_a+t_a=j_a,\, a\neq n}} c_{k,l_1,\ldots,l_m}\, t_n\, u_{s,t_1,\ldots,t_m} \right) - \left(\sum_{\substack{k+s=i \\ l_n+t_n=j_n+1 \\ l_a+t_a=j_a,\, a\neq n}} l_n\, c_{k,l_1,\ldots,l_m}\, u_{s,t_1,\ldots,t_m} \right)$$
$$+ \left(\sum_{\substack{k+s=i+1 \\ l_a+t_a=j_a}} k\, c_{k,l_1,\ldots,l_m}\, v_{s,t_1,\ldots,t_m} \right) - \left(\sum_{\substack{k+s=i+1 \\ l_a+t_a=j_a}} c_{k,l_1,\ldots,l_m}\, s\, v_{s,t_1,\ldots,t_m} \right).$$

Clearly, a given $c_{k,l_1,\ldots,l_m}$ can appear at most four times in this coefficient. Examination of the indexes reveals that both $u_{t,s_1,\ldots,s_m}$ corresponding to a given $c_{k,l_1,\ldots,l_m}$ have the exact same indexes (once $k, l_1, \ldots, l_m$ are fixed, so then must be $s, t_1, \ldots, t_m$). Similarly for the two $v_{s,t_1,\ldots,t_m}$ corresponding to a given $c_{k,l_1,\ldots,l_m}$. Hence, $c_{k,l_1,\ldots,l_m}$ appears at most twice in any given row of $\mathrm{Rup}(f)$ either from the term $(t_n - l_n)\, c_{k,l_1,\ldots,l_m}\, u_{s,t_1,\ldots,t_m}$, or the term $(k-s)\, c_{k,l_1,\ldots,l_m}\, v_{s,t_1,\ldots,t_m}$. Since $|k-s| \leq d$, and $|t_n - l_n| \leq e_n$ we have also proven that each non-zero entry of $\mathrm{Rup}(f)$ is of the form $a\, c_{k,l_1,\ldots,l_m}$ where $|a| \leq \max\{d, e_1, \ldots, e_m\}$.

Now let us look at how many times a given $c_{k,l_1,\ldots,l_m}$ can appear in a column of $\mathrm{Rup}(f)$. We will first look at how many times it can appear multiplied by a given $u_{s,t_1,\ldots,t_m}$ or $v_{s,t_1,\ldots,t_m}$ in (2.10). As we can see above there is only one possibility for each. Thus $c_{k,l_1,\ldots,l_m}$ appears at most once in each column of the part of the matrix corresponding to each of the equations (2.10). There are $m-1$ of these equations, so, $c_{k,l_1,\ldots,l_m}$ appears as most $m-1$ in the columns corresponding to the coefficients of g, and at most once in each of the columns corresponding to an h_i.

Notice also, there is not a multiple of $c_{k,l_1,\ldots,l_m}$ in every column. It appears in the terms

$$(t_n - l_n)\, c_{k,l_1,\ldots,l_m}\, u_{s,t_1,\ldots,t_m}\, x^{k+s} y_1^{l_1+t_1} \ldots y_n^{l_n+t_n-1} \ldots y_m^{l_m+t_m}$$

$$\text{and } (k-s)\, c_{k,l_1,\ldots,l_m}\, v_{s,t_1,\ldots,t_m}\, x^{k+s-1} y_1^{l_1+t_1} \ldots y_m^{l_m+t_m}.$$

For the case that $k = s = 0$, $(t_n - l_n)\, c_{0,l_1,\ldots,l_m}\, v_{0,t_1,\ldots,t_m}$ would have to be in the coefficient of $x^{-1} y_1^{l_1+t_1} \ldots y_m^{l_m+t_m}$ so $c_{0,l_1,\ldots,l_m}$ does not appear in the column corresponding to $v_{0,t_1,\ldots,t_m}$

for any l_a and t_b. Similarly for the case when $l_n = t_n = 0$. The term

$$j\, c_{k,l_1,\ldots,j,\ldots l_m}\, u_{s,t_1,\ldots,j,\ldots,t_m}\, x^{k+s} y_1^{l_1+t_1} \ldots y_n^{2j-1} \ldots y_m^{l_m+t_m}$$

appears in the polynomials $f g_{y_n}$ and $g f_{y_n}$ in (2.10), hence it cancels. So a given $c_{k,l_1,\ldots,j,\ldots l_m}$ does not appear in the column corresponding to $u_{s,t_1,\ldots,j,\ldots,t_m}$. And, similarly, $c_{k,l_1,\ldots,l_m}$ does not appear in any column corresponding to $v_{k,t_1,\ldots,t_m}$.

Thus for a each n, the matrix for (2.10) has at most $(2de_n - d)\prod_{l\neq n}(e_l + 1)$ entries that contain $c_{k,l_1,\ldots,j,\ldots l_m}$. So $\mathrm{Rup}(f)$ has at most

$$\sum_{i=1}^{m} \left((2de_i - d) \prod_{j\neq i} (e_j + 1) \right)$$

occurrences of each coefficient of f.

In addition, since the all the terms containing $y_k^{2e_k-1}$ in (2.10) must vanish we know that $\sum_i (2d \prod_{i\neq j}(2e_i + 1))$ rows of $\mathrm{Rup}(f)$ will always be zero. So the dimensions of $\mathrm{Rup}(f)$ are in fact:

$$\left(\sum_{i=1}^{m} (4de_i - 2d) \prod_{j\neq i} (2e_j + 1) \right) \times \left(\sum_{i=1}^{m} (2de_i + e_i - 1) \prod_{j\neq i} (e_j + 1) \right),$$

or $(4de_1 - 2d) \times (2de_1 + e_1 - 1)$ when $m = 1$. □

Suppose φ is a polynomial with symbolic coefficients with $\deg(\varphi) = \deg(f)$. For a perturbation $f_\triangle$ with $\deg(f_\triangle) \leq \deg(f)$, we consider the matrix $\mathrm{Rup}(\varphi)\big|_{\varphi=f_\triangle}$, which

denotes the Ruppert matrix of φ with the symbolic coefficients set to their values in $f_\triangle$. However, notice that $\mathrm{Rup}(\varphi)\big|_{\varphi=f_\triangle}$ is only the same as $\mathrm{Rup}(f_\triangle)$ if $f_\triangle$ has the same degree as φ, otherwise the two matrices have different dimensions.

Lemma 2.8. *Given φ,a polynomial with symbolic coefficients, and $f_\triangle$ a polynomial with the same or lower degree, the following bounds hold:*

1. $\|\mathrm{Rup}(\varphi)\big|_{\varphi=f_\triangle}\|_1 \leq \max\{d, e_1, \ldots, e_m\}\, \|f_\triangle\|_1$
2. $\|\mathrm{Rup}(\varphi)\big|_{\varphi=f_\triangle}\|_\infty \leq (m-1)\, \max\{d, e_1, \ldots, e_m\}\, \|f_\triangle\|_1$
3. $\|\mathrm{Rup}(\varphi)\big|_{\varphi=f_\triangle}\|_{\infty,1} \leq \max\{d, e_1, \ldots, e_m\}\, \|f_\triangle\|_\infty$
4. $\|\mathrm{Rup}(\varphi)\big|_{\varphi=f_\triangle}\|_2 \leq \|\mathrm{Rup}(\varphi\big|_{\varphi=f_\triangle})\|_F \leq \sqrt{C}\, \max\{d, e_1, \ldots, e_m\}\, \|f_\triangle\|_2$,
   $C = \sum_{i=1}^{m} \Big((2de_i - e_i) \prod_{j\neq i} (e_j + 1) \Big)$.

Note, for the two variable case, Nagasaka (2004) shows the right-hand side in part 4 can be replaced with:

$$1/6\; n\,(m\,(m+1)\,(2\,m+1) + (m-1)\,(n+1)\,(2n+1))\,\|f_\triangle\|_2 \text{ where } (m,n) = \mathrm{mdeg}(f).$$

Proof. We derive the bound on the 2-norm. The derivations of the other bounds are

similar.

$$\begin{aligned}\|\mathrm{Rup}(\varphi)\big|_{\varphi=f_\triangle}\|_F^2 &= \sum_{i,j} |\big(\mathrm{Rup}(\varphi)\big|_{\varphi=f_\triangle}\big)_{i,j}|^2 = \sum_{i,j} |a_{i,j}b_{k_{i,j}}|^2 \\ &\leq \max\{a_{i,j}^2\} \sum_{i,j} |b_{k_{i,j}}|^2 \leq (\max\{d, e_1, \ldots, e_m\})^2 \, C \, \|f_\triangle\|_2^2,\end{aligned}$$

where b_k is a coefficient of $f_\triangle$ or is zero and $C = \sum_{i=1}^m \Big((2de_i - e_i)\prod_{j\neq i}(e_j+1)\Big)$ is the number of times a given b_k can appear in $\mathrm{Rup}(\varphi)$ from Lemma 2.7. □

2.3.2 The Main Approximate Irreducibility Theorem

Now we prove a theorem that translates an upper-bound on an operator norm of $\mathrm{Rup}(f)$ in terms of a polynomial norm of f into a lower bound on the distance to the nearest factorizable polynomial to f.

Theorem 2.9. *Suppose that $f \in \mathbb{C}[x, y_1, \ldots, y_m]$ is an irreducible polynomial, and that $\tilde{f} \in \mathbb{C}[x, y_1, \ldots, y_m]$ is a polynomial of equal or lesser degree. Additionally suppose that there exists C so that*

$$\|\mathrm{Rup}(\varphi)\big|_{\varphi=f_\triangle}\|_{p,q} \leq C \, \|f_\triangle\|_r. \tag{2.11}$$

If

$$\|f - \tilde{f}\|_r < (C \, \|\mathrm{Rup}(f)^\dagger\|_{q,p})^{-1}. \tag{2.12}$$

then $\tilde{f}$ is irreducible.

Note that the relationships in Lemma 2.8 give four bounds of the form (2.11).

Proof. Begin by assuming that f and $\tilde{f}$ have the same degree.

By (2.11) and (2.12) applied to $f_\triangle = f - \tilde{f}$ we have

$$\|\mathrm{Rup}(f) - \mathrm{Rup}(\tilde{f})\|_{p,q} = \|\mathrm{Rup}(\varphi)\big|_{\varphi=f_\triangle}\|_{p,q} \leq C\,\|f_\triangle\|_r < \|\mathrm{Rup}(f)^\dagger\|_{q,p}^{-1}. \tag{2.13}$$

Now, since f is irreducible, $\mathrm{Rup}(f)$ is full rank by Theorem 2.1. Hence, Fact 2.6 with (2.13) implies that $\mathrm{Rup}(\tilde{f})$ is so close to $\mathrm{Rup}(f)$ it must also be full rank. Thus, $\tilde{f}$ is irreducible, by Theorem 2.1.

We now assume that $\tilde{f}$ has smaller degree than f. In order to derive a contradiction, let us assume now that $\tilde{f} = gh$ where g and h are non-unit factors, and

$$\|f - \tilde{f}\|_r < (C\,\|\mathrm{Rup}(f)^\dagger\|_{q,p})^{-1}.$$

Now we can construct

$$\tilde{\tilde{f}} = (\epsilon\,(x^s + y_1^{t_1} + \ldots + y_m^{t_m}) + g)\,h = \epsilon\,(x^s + y_1^{t_1} + \ldots + y_m^{t_m})\,h + \tilde{f},$$

with $s = \deg_x(f) - \deg_x(h)$ and $t_i = \deg_{y_i}(f) - \deg_{y_i}(h)$, and $\epsilon \in \mathbb{R}_{>0}$ chosen so that

$$\epsilon < \frac{(C\,\|\mathrm{Rup}(f)^\dagger\|_{q,p})^{-1} - \|f - \tilde{f}\|_r}{\|(x^s + y_1^{t_1} + \ldots + y_m^{t_m})h\|_r}.$$

Furthermore, we restrict ϵ so that $\tilde{\tilde{f}}$ has the same degree as f. This is possible because

there are at most $m+1$ values of ϵ that can cause

$$\epsilon\,(x^s + y_1^{t_1} + \ldots + y_m^{t_m})\,h + \tilde{f}$$

to have lower degree than f, while we have an infinite number of choices for ϵ that satisfy the inequality. Thus, $\tilde{\tilde{f}}$ factors and has the same degree as f, but

$$\|f - \tilde{\tilde{f}}\|_r \leq \|f - \tilde{f}\|_2 + \epsilon\,\|(x^s + y_1^{t_1} + \ldots + y_m^{t_m})\,h\|_r < (C\,\|\mathrm{Rup}(f)^\dagger\|_{q,p})^{-1},$$

which contradicts the first part of the proof. □

Note that this theorem does not hold if we allow the degree of $\tilde{f}$ to be greater than that f. For all f and $\epsilon > 0$, $\tilde{f} = (\epsilon\,x + 1)f$ is a polynomial of higher degree than f that is reducible. But,

$$\|\tilde{f} - f\| = \|\epsilon\,x\,f\| = \epsilon\,\|f\|$$

which, by choosing ϵ small enough, can be made arbitrarily small.

In practice, it is possible to get better bounds C than the ones given in Lemma 2.8 by forming the Ruppert matrix for the polynomial with symbolic coefficients having the same degree as f, computing the desired norm of the symbolic matrix, and finding the largest coefficient of a $|c_{i,j}|$ term. We use this technique in the following examples.

Example 2.2. (BOUNDS VS. TRUE DISTANCE.) Using the notation of Example 2.1, consider again the absolutely irreducible polynomial $f = x^2 + y^2 - 1$. Computing $\|\mathrm{Rup}(\varphi)\|_F^2$,

we get:

$$15\ |c_{0,2}|^2 + 15\ |c_{2,2}|^2 + 15\ |c_{2,0}|^2 + 12\ |c_{1,2}|^2 + 9\ |c_{2,1}|^2 + 6\ |c_{1,1}|^2 + 15\ |c_{0,0}|^2 + 12\ |c_{1,0}|^2 + 9\ |c_{0,1}|^2 .$$

The largest coefficient is 15 (the bound predicted in the theorem is 24), and the smallest singular value of $\mathrm{Rup}(f)$ is $\sigma(\mathrm{Rup}(f)) = \sqrt{15}/2 - \sqrt{7}/2 \approx 0.61361601757141$, so a perturbation that makes f singular must have 2-norm at least $\sigma(\mathrm{Rup}(f))/\sqrt{15} = 1/2 - \sqrt{105}/30 \approx 0.1584349745$.

This polynomial is small enough that it is possible to find the closest factorizable polynomial (with real coefficients, and same *total* degree) by using parametric least squares as in Hitz *et al.* (1999). This involves taking the equation

$$f - (a_1\, x + a_2\, y + a_3)(x + b_1\, y + b_2) = 0 \tag{2.14}$$

and considering it as a set of linear equations in a_1, a_2, and a_3. We can write this as a linear system: $Ma = F$, where the matrix M has entries in $\mathbb{R}[b_1, b_2]$, and F is a vector of the coefficients of f. Now the residual of the least squares solution of this system,

$$q = \|F - M(M^T M)^{-1} M^T F\|_2,$$

is a rational function in b_1 and b_2. We can find a global minimum of q by taking the

partial derivatives of its numerator, q_1 and q_2, and solving $\mathrm{Res}_{b_2}(q_1, q_2) = 0$. Once we have a real solution $b_1 = \alpha_1$, we can solve $\gcd(q_1(\alpha_1, b_2), q_2(\alpha_1, b_2)) = 0$ for corresponding real α_2's. We check all such pairs, and substitute the pair that leads to the smallest residual back into M. We thus obtain a linear system over $\mathbb{R}$ from which we can compute the least squares solution. Doing so, we find that closest reducible polynomials *with real coefficients* and total degree 2 are at least distance 1 from f, for example,

$$\tilde{f} = (x - 1)(x + 1).$$

The closest reducible polynomial with degree 2 in x and y is closer. For example, the following $\tilde{f}$ is only distance 0.6727223250 away from f:

$$\tilde{f} = (0.4906834y^2 + 0.8491482x - 0.9073464)(x + 1.214778).$$

Finding the closest factorizable polynomial *with complex coefficients* is computationally more difficult, since each parameter above turns into two parameters, one each for the real and the imaginary parts.

We can compute lower bounds for other norms as well. After computing the pseudo-inverse of $\mathrm{Rup}(f)$, a task that is accomplished handily with the `MatrixInverse` command in Maple version 8 or later, we compute its various norms.

$$\|\mathrm{Rup}(f)^\dagger\|_1 = 5/2,\ \|\mathrm{Rup}(f)^\dagger\|_\infty = 2,\ \|\mathrm{Rup}(f)^\dagger\|_{1,\infty} \leq 9.$$

The maximum multiple of an absolute column sum is 2. Looking at a symbolic Rup(φ) we find that the maximum multiple of an absolute row sum is 3, less than the worst case of 4 predicted above, and the maximum multiple of an entry is 2. Hence we get that the ∞-norm of a perturbation of f must be greater than $1/18 \approx 0.05556$, and the 1-norm must be greater th an $1/5 = 0.2$ (from the matrix 1-norm), and $1/6 \approx 0.16667$ (from the matrix ∞-norm).

□

Example 2.3. (COMPARISON WITH NAGASAKA (2002).) Apply Theorem 2.9 to Nagasaka's Example 1:

$$F = (x^2 + yx + 2y - 1)(x^3 + y^2x - y + 7) + 0.2x.$$

Here, Rup(F) is a 47×32 matrix after removing the zero rows. Forming the symbolic polynomial with the same degrees and computing its Frobenius norm, we get:

$$\begin{aligned}
&140|c_{0,1}|^2 + 140|c_{0,2}|^2 + 180|c_{0,3}|^2 + 92|c_{1,1}|^2 + 132|c_{1,3}|^2 \\
&+ 68|c_{2,1}|^2 + 92|c_{1,2}|^2 + 68c_{2,2}|^2 + 108|c_{2,3}|^2 + 68|c_{3,1}|^2 \\
&+ 92|c_{4,1}|^2 + 108|c_{3,3}|^2 + 68|c_{3,2}|^2 + 180|c_{5,3}|^2 + 140|c_{5,2}|^2 \\
&+ 132|c_{4,3}|^2 + 140|c_{5,1}|^2 + 92|c_{4,2}|^2 + 108|c_{3,0}|^2 + 108|c_{2,0}|^2 \\
&+ 132|c_{1,0}|^2 + 180|c_{0,0}|^2 + 180|c_{5,0}|^2 + 132|c_{4,0}|^2.
\end{aligned}$$

The largest coefficient is 180, so by Theorem 2.9, a perturbation with 2-norm at least

$\sigma(\mathrm{Rup}(F))/\sqrt{180}$ is needed to make F reducible. Numerically computing the singular value $\sigma(\mathrm{Rup}(F)) \approx 1.030023214 \times 10^{-2}$, we find $\sigma(\mathrm{Rup}(F))/\sqrt{180} \approx 7.677339751 \times 10^{-4}$. The norm $\|f\|_2 \approx 19.8504408$, and dividing the absolute bound by this gives a relative bound of 3.8676^{-5} compared to the bound Nagasaka computes: 5.53×10^{-6}.

In addition we find bounds for other norms as well. Computing the pseudo-inverse of $R(f)$ we get the norms

$$\|R(f)^\dagger\|_1 \approx 113.598, \quad \|R(f)^\dagger\|_\infty \approx 270.393,$$

$$\|R(f)^\dagger\|_{1,\infty} \leq 1192.372.$$

These lead to the bounds:

$$\|f_\triangle\|_1 > 0.001760594950 \text{ and } \|f_\triangle\|_\infty > 0.0001677328901$$

if $f - f_\triangle$ is factorizable.

□

Recently, in Nagasaka (2004), a scheme was presented which removes redundant rows from $\mathrm{Rup}(f)$ in order to achieve small (6%) improvements to the lower bounds for the two variable examples presented above.

Example 2.4. (TWO POLYNOMIALS IN THREE VARIABLES) Consider the absolutely

irreducible polynomials

$$f_1 = x^2 + y^2 + z^2 - 1,$$

the sphere, and

$$f_2 = 81\,x^4 + 16\,y^4 - 648.001\,z^4 + 72\,x^2y^2 + 0.002\,x^2z^2 + .001\,y^2z^2 - 648\,x^2 - 288\,y^2 - .007\,z^2 + 1296,$$

from Kaltofen (2000).

The largest coefficient of $\|\mathrm{Rup}(\varphi_1)\|_2^2$ (a 120×45 matrix) is 90, and the smallest singular value of $\mathrm{Rup}(f_1)$ is $\sigma \approx .852159217299423788$ giving a lower bound on the radius of irreducibility of 0.0898254685.

For the other polynomial, the largest coefficient of $\|\mathrm{Rup}(\varphi_2)\|_2^2$ (a 1008×275 matrix) is 2100, and the smallest singular value of $\mathrm{Rup}(f_2)$ is $\sigma \approx 4.76613104337138402 \times 10^{-11}$ giving a lower bound of $1.04005506 \times 10^{-12}$. This bound seems quite small since f_2 was constructed from the factorizable polynomial

$$81\,x^4 + 16\,y^4 - 648\,z^4 + 72\,x^2y^2 - 648\,x^2 - 288\,y^2 + 129,$$

that is only distance $7.416198487 \times 10^{-3}$ away from f_2.

□

Remark 2.1. The singular values in the above examples were computed to machine

precision from matrices with floating point entries in MAPLE using the `SingularValues` command which uses the NAG numerical library as its backend. To ensure that the bounds were numerically accurate, the same singular values were computed again with `SingularValues` but this time with matrices with rational number entries so that exact techniques would be employed (i.e. the singular values would be computed as the roots of a characteristic polynomial). Applying MAPLE's numerical root finder (via `evalf`) to these exact singular values, we find that they agree to 15 decimal digits with the ones computed using purely numerical techniques.

For larger examples, computing the singular values exactly would not be practical. Already, Example 2.3 has a characteristic polynomial that is degree 32 with coefficients with 90 decimal digits. In this case, one should be sure to use a numerical singular value implementation which is certified to be accurate.

It can been seen above that the Ruppert matrices get quite large very quickly (size of $\mathrm{Rup}(f)$ grows linearly with the number of terms of f and exponentially in the number of variables). Thus, computing lower bounds on the radius of irreducibility for polynomials with many variables and large degrees can benefit greatly from algorithms that compute the norms of $\|\mathrm{Rup}(f)^\dagger\|$ in ways which takes advantage of its structure. For example, the smallest singular value of a matrix can be computed with an inverse power method which is a black box iteration. That is, it does not need to know the entries of the matrix, it just needs to be able to multiply by the matrix and its transpose both of which can be done without explicitly constructing the Ruppert matrix.

It is not clear that there is any way to find a better radius of irreducibility in the multivariate case by projecting to the two variable case. Note, however, that a lower bound on the radius of irreducibility can also be computed from Noether irreducibility forms as in Kaltofen (1995). Such bounds are much easier to compute but in general are not as tight as the ones above. We discuss how to find these lower bounds in the next section.

2.4 More Effective Noether Irreducibility Forms

For a fixed number of variables and a fixed total degree d, it is possible to compute a set of polynomials with rational number coefficients which vanish when evaluated at the coefficients of a rational multivariate polynomial of total degree d that is not absolutely irreducible and that do not vanish when evaluated at the coefficients of a rational multivariate polynomial of total degree d that is absolutely irreducible. Such a set of polynomials are called a Noether Irreducibility Form named for Emmy Noether who proved that such forms exist (Noether, 1922). Noether's construction was analyzed in Schmidt (1976) and shown to lead to impractically large forms. Later, Kaltofen (1995) constructed new, smaller, more effective forms. Here we present forms which are even more effective.

We shall derive new Noether irreducibility forms using Theorem 2.1. We will first reduce the problem of constructing these forms to the case of two variables using the following fact. This is done in part so that our forms will not grow as quickly with the

number of variables. First, suppose that

$$f = \sum_{e_1+\ldots+e_n \leq d} c_{e_1,\ldots,e_n} x_1^{e_1} \cdots x_n^{e_n} \in \mathbb{K}[x_1, \ldots, x_n],$$

where $\mathbb{K}$ is a field of characteristic 0.

Fact 2.10. (Lemma 7 in Kaltofen (1995)) *Let f be as above and*

$$L = \mathbb{K}(v_1, \ldots, v_n, w_2, \ldots, w_n, z_2, \ldots, z_n),$$

where $v_1, \ldots, v_n, w_2, \ldots, w_n, z_2, \ldots, z_n$ are indeterminants. The bivariate polynomial

$$\hat{f}(x, y) = f(x + y + v_1, w_2 x + z_2 y + v_2, \ldots, w_n x + z_n y + v_n) \in L[x, y]$$

is irreducible over the algebraic closure of L if and only if f is irreducible over the algebraic closure of $\mathbb{K}$.

Note that in Kaltofen (1995), the substitution for x_1 is $x + v_1$, but the proof follows through using $x + y + v_1$ as given above. The advantage of the latter substitution is that $\deg_x \hat{f} = \deg_y \hat{f} = \text{tdeg}(\hat{f}) = \text{tdeg}(f)$ where by tdeg we mean the total degree. This allows us to formulate the following theorem using total degree even though Theorem 2.1 depends on the rectangular degrees.

Theorem 2.11. (cf. Theorem 7 in Kaltofen (1995)) *There exists a finite set of*

polynomials

$$\Phi_t \in E := \mathbb{Z}[\ldots, b_{e_1,\ldots,e_n}, \ldots],\ 1 \le t \le T$$

where the $b_{e_1,\ldots,e_n}$*'s are indeterminants, so that*

$$\forall t\colon \Phi_t(\ldots, c_{e_1,\ldots,e_n}, \ldots) = 0 \iff f \text{ is not absolutely irreducible or } \operatorname{tdeg}(f) < d.$$

Furthermore, for all t*,*

$$\operatorname{tdeg}(\Phi_t) \le 2d^2 + d \text{ and } \|\Phi_t\|_1 \le (2d)^{4d^2+3d+n}.$$

Proof. The proof will closely follow the proof of Theorem 7 in Kaltofen (1995). First, write

$$\varphi = \sum_{e_1+\ldots+e_n \le d} b_{e_1,\ldots,e_n} x_1^{e_1} \cdots x_n^{e_n},$$

and

$$\hat{\varphi} = \varphi(x + y + v_1, w_2 x + z_2 y + v_2, \ldots, w_n x + z_n y + v_n) \in L'[x, y]$$

where $L' = E(v_1, \ldots, v_n, w_2, \ldots, w_n, z_2, \ldots, z_n)$.

Let $\{\Delta_s\}$ be the set of all maximal minors of the matrix $\operatorname{Rup}(\hat{\varphi})$. Define the set

$$S := \left\{ \tau \in E \;\middle|\; \begin{array}{l} \tau \text{ is a coefficient of a term in} \\ \\ v_1, \ldots, v_n, w_2, \ldots, w_n, z_2, \ldots, z_n \text{ of some } \Delta_s\}. \end{array} \right\}.$$

We shall define the set of irreducibility forms as follows:

$$\{\Phi_t = b_{e_1,\ldots,e_n}\tau \in E \mid e_1 + \ldots + e_n = d,\, \tau \in S\}.$$

Now let us substitute the coefficients of f (the $c_{e_1,\ldots,e_n}$'s) for the indeterminants $b_{e_1,\ldots,e_n}$. Note that one of our forms $\Phi_t(\ldots, c_{e_1,\ldots,e_n}, \ldots) = 0$ if and only if $\tau(\ldots, c_{e_1,\ldots,e_n}, \ldots) = 0$ for $\tau \in S$ or $c_{e_1,\ldots,e_n} = 0$ for all $e_1+\ldots+e_n = d$. The condition $c_{e_1,\ldots,e_n} = 0$ for all $e_1+\ldots+e_n = d$ holds if and only if f does not have total degree d. Notice, $\tau(\ldots, c_{e_1,\ldots,e_n}, \ldots) = 0$ for all $\tau \in S$ if and only if $\Delta_s(\ldots, c_{e_1,\ldots,e_n}, \ldots) = 0$. This is true if and only if $\mathrm{Rup}(\hat{\varphi}\big|_{\hat{\varphi}=\hat{f}})$ does not have full rank. Now, if $\mathrm{tdeg}(f) = d$, then $\deg_x \hat{f} = \deg_x \hat{\varphi} = \deg_y \hat{f} = \deg_y \hat{\varphi} = d$, hence, $\mathrm{Rup}(\hat{f}) = \mathrm{Rup}(\hat{\varphi}\big|_{\hat{\varphi}=\hat{f}})$. Thus, by Theorem 2.1, $\mathrm{Rup}(\hat{f})$ is rank deficient if and only if $\hat{f}$ factors over the algebraic closure of L which is true if and only if f factors over $\mathbb{C}$, by Fact 2.10.

Now we establish the bounds on Φ_t. Notice that all the coefficients of terms $x^i y^j$ in $\hat{\varphi}$ are linear in the $b_{e_1,\ldots,e_n}$'s, hence the entries of $\mathrm{Rup}(\hat{\varphi})$ are also linear in the $b_{e_1,\ldots,e_n}$'s by Lemma 2.7. Thus, any minor of $\mathrm{Rup}(\hat{\varphi})$ will have total degree in the $b_{e_1,\ldots,e_n}$'s at most $2d^2 + d - 1$, the number of columns of $\mathrm{Rup}(\hat{\varphi})$. Therefore the total degree of any Φ_t will be at most $2d^2 + d$. To bound the 1-norm, note that each coefficient of $\hat{\varphi}$ has 1-norm at most

$$\binom{d+n}{n} 3^d =: A \leq (2d)^{d+n}.$$

Therefore, a minor of $\text{Rup}(\hat{\varphi})$ has 1-norm at most

$$A\,(2d^2+d-1)(2d^2+d-1)! \leq (2d)^{d+n}(2d^2+d-1)^{2d^2+d} \leq (2d)^{4d^2+3d+n},$$

and $\|\Phi_t\|_1$ must certainly be smaller than this as well. □

The bounds

$$B := 2d^2+d \text{ and } D := (2d)^{4d^2+3d+n}.$$

on these new Noether irreducibility forms lead to a separation bound for multivariate polynomials. For the following, we will assume that $f \in \mathbb{Z}[x_1,\ldots,x_n]$, though similar results can be derived for f that have coefficients in $\mathbb{Z}[\xi]$ where ξ is an algebraic integer over $\mathbb{Q}$.

Fact 2.12. (Theorem 10 in Kaltofen (1995)) *If $\tilde{f} \in \mathbb{C}[x_1,\ldots,x_n]$ has the same total degree as f and*

$$\|f-\tilde{f}\|_\infty < 2^{-(d+m+1)}D^{-1}B^{-1}(\|f\|_\infty+1)^{-D}$$

then $\tilde{f}$ is irreducible.

The Noether forms in Kaltofen (1995) have bounds

$$D' = 12d^6 \text{ and } B' = (2d)^{12d^7+(12n+36)d^6}$$

which are much larger than the bounds in Theorem 2.11. It should be noted that the forms in Kaltofen (1995) apply to fields of positive characteristic. While the new forms in Theorem 2.11 are given for characteristic 0 they can be made to work in characteristic p so long as p is large enough. Indeed, Gao (2003) shows that the reducibility test we use will work when $p > e_1 (2d - 1)$ in the bivariate case.

The bounds (2.4) lead to very smaller separation bounds using the above fact:

$$\|f - \tilde{f}\|_\infty < (2d)^{-(12d^7+29nd^6)}(\|f\|_\infty + 1)^{-12d^6}.$$

Using the B and D from the new Noether forms in Theorem 2.11, we get a better bound:

$$\|f - \tilde{f}\|_\infty < (2d)^{-(4d^2+4d+2n+1)}(\|f\|_\infty + 1)^{-2d^2-d}. \tag{2.15}$$

Using (2.15) on the two polynomials in Example 2.4 we get radii of irreducibility of 2.12×10^{-22} for the first polynomial and 2.32×10^{-191} for the second. While much easier to compute, these are clearly worse than the bounds found using the theory in the previous section.

Chapter 3

Exact Factorization

3.1 A Multivariate Generalization of Gao's Factorization Algorithm

In many cases, for efficiency, factoring polynomials in more that two variables is done by evaluating away variables to reduce to the two variable case then lifting the results via interpolation or Hensel lifting back to the original variables. In this chapter, we consider a technique for factoring multivariate polynomials directly. This is done with numerical and approximate factorization in mind. Since interpolation and lifting can be very difficult numerically, we wish to have a factorization technique which avoids them.

3.1.1 Structure of the PDE Solutions

It is possible to explicitly write down the structure of the solutions to the PDE in (2.1) when we additionally restrict that f must be square-free over $C(y_1, \ldots, y_m)$. The following lemma is a direct generalization of the bivariate result found in Gao (2003).

Lemma 3.1. *Suppose* $f = f_1 \cdots f_r$, *and* f *is square-free over* $\mathbb{C}(y_1, \ldots, y_m)$. *If the polynomials* $(g, h_1, \ldots, h_m)$ *satisfy* (2.1) *and* (2.2) *then*

$$g = \sum_{j=1}^{r} \lambda_j \frac{f}{f_j} \frac{\partial f_j}{\partial x} \text{ and } h_i = \sum_{j=1}^{r} \lambda_j \frac{f}{f_j} \frac{\partial f_j}{\partial y_i} \text{ with } \lambda_j \in \mathbb{C}.$$

Proof. The assumption that f is square-free is enough to conclude (as in the proof of Theorem 2.1) that the partial fraction decomposition of g looks like:

$$\frac{g}{f} = \sum_{j=1}^{d} \frac{b_j}{x - \alpha_j},$$

where the b_js are complex numbers and if α_j and α_k are roots of the same factor f_i then $\lambda_i := b_j = b_k$. Thus

$$\frac{g}{f} = \sum_{i=1}^{r} \frac{\lambda_i}{f_i} \frac{\partial f_i}{\partial x}.$$

Since each h_i appears in only one equation in (2.1) a different argument is needed for their structure. If we consider the equation

$$\frac{\partial}{\partial y_1} \frac{g}{f} = \frac{\partial}{\partial x} \frac{h_1}{f} \tag{3.1}$$

over $\mathbb{C}(y_2, \ldots, y_m)[x, y_1]$, then the bivariate result in Gao (2003) shows that

$$h_1 = \sum_{j=1}^{r} \lambda_j \frac{f}{f_j} \frac{\partial f_j}{\partial y_1} \text{ with } \lambda_j \in \mathbb{C}(y_2, \ldots, y_m).$$

Similarly this holds for $i \neq 1$ as well.

Now we argue that if

$$g = \sum_{j=1}^{r} b_j \frac{f}{f_j} \frac{\partial f_j}{\partial x} \text{ and } h_1 = \sum_{j=1}^{r} c_j \frac{f}{f_j} \frac{\partial f_j}{\partial y_1} \text{ with } b_j, c_j \in \mathbb{C}(y_2, \ldots, y_m),$$

then if g and h_1 also satisfy (3.1) then $a_i = b_i$ for all i. In this case, (3.1) can be rewritten as

$$\sum_{j=1}^{r} b_j \frac{\partial}{\partial y_1} \frac{1}{f_j} \frac{\partial f_j}{\partial x} - \sum_{j=1}^{r} c_j \frac{\partial}{\partial x} \frac{1}{f_j} \frac{\partial f_j}{\partial y_1} = 0.$$

But,

$$\frac{\partial}{\partial y_1} \frac{1}{f_j} \frac{\partial f_j}{\partial x} = \frac{\partial}{\partial x} \frac{1}{f_j} \frac{\partial f_j}{\partial y_1}$$

so it can further be rewritten as

$$\sum_{j=1}^{r} (b_j - c_j) \frac{\partial}{\partial y_1} \frac{1}{f_j} \frac{\partial f_j}{\partial x} = 0. \tag{3.2}$$

A look at the partial fraction decomposition

$$\frac{\partial}{\partial y_1} \frac{1}{f_j} \frac{\partial f_j}{\partial x} = \frac{1}{f_j} \frac{\partial^2 f_j}{\partial x y_1} - \frac{1}{f_j^2} \frac{\partial f_j}{\partial x} \frac{\partial f_j}{\partial y_1}$$

shows that (3.2) holds if and only if $b_j = c_j$. Thus, since $b_j \in \mathbb{C}$ from the first part of the proof, h_i has the given structure. □

3.1.2 Counting the Factors

It is worth noting that the bounds (2.2) exclude the "natural" solutions to the PDE, namely

$$g = \frac{f}{f_i}\frac{\partial f_i}{\partial x},\ h_j = \frac{f}{f_i}\frac{\partial f_i}{\partial y_j}, \tag{3.3}$$

because $\deg_x g = d-1$. If we adjust the bounds (2.2) of the PDE to allow $\deg_x g = d-1$ then they become

$$\begin{array}{lll} \deg_x(g) \le (d-1) & \deg_{y_i}(g) \le e_i & i = 1, \ldots, m \\ \deg_x(h_j) \le d & \deg_{y_i}(h_j) \le \begin{cases} e_i & \text{if } i \neq j \\ e_i - 1 & \text{if } i = j \end{cases} & j = 1, \ldots, m \end{array} \tag{3.4}$$

and we admit the solutions (3.3), and (it is not difficult to show) no other new independent solutions, and by Lemma 3.1 solutions of this form are a linear basis for the set of solutions to the PDEs. This gives an easy algorithm to count the number of factors of a polynomial based on the following corollary.

Corollary 3.2. *For a given $f \in \mathbb{C}[x, y_1, \ldots, y_n]$ that is square-free over $\mathbb{C}(y_1, \ldots, y_n)$ the dimension (over $\mathbb{C}$) of the solution space of the system of PDEs (2.1) with bounds (3.4) is equal to the number of absolutely irreducible factors of f over $\mathbb{C}$.*

We realize Corollary 3.2 as an algorithm simply by writing the system of PDEs as a matrix as before. This time the bounds of possible solutions are different, so for a given polynomial f we will denote the matrix of the system (2.1) with bounds (3.4) as $\mathrm{Rup}_1(f)$. To count the factors of f, one has merely to find the dimension of the nullspace (the nullity) of $\mathrm{Rup}_1(f)$.

3.1.3 Algorithm Description

Using the structure of the solutions of the PDEs from Lemma 3.1, we can give a direct generalization of the factorization algorithm in Gao (2003).

Since

$$f_x = \sum_{i=1}^{r} \frac{f}{f_i} \frac{\partial f_i}{\partial x},$$

if we have a solution

$$g = \sum_{i=1}^{r} \lambda_i \frac{f}{f_i} \frac{\partial f_i}{\partial x},$$

with all the λ_i distinct, then we can recover most of the factors of f by computing the GCDs:

$$\hat{f} = \prod_{i=1}^{r} \gcd(f, g - \lambda_i f_x),$$

where $\hat{f}$ is the product of all the factors of f that are not constant in x (i.e. the primitive part of f over $\mathbb{C}(y_1, \ldots, y_m)$). Since we do not know the λ_is for a g computed from a random null vector of $\mathrm{Rup}_1(f)$, we look for the values λ that give a nontrivial GCD. One method of finding such λs is to compute $\mathrm{Res}_x(f, g - z\, f_x)$ over $\mathbb{C}(y_1, \ldots, y_m)$ then find

its roots (as a polynomial in z) that live in $\mathbb{C}$. Another method for finding these λs by constructing a matrix A_g whose eigenvalues are exactly the λs that give nontrivial GCDs is given in Gao (2003, Theorem 2.8). In either case, we can choose a random projection $y_i \to \alpha_i \in \mathbb{C}$ for a probabilistic method to find the λ_is very quickly. The following Fact sketches Gao's method to find the λ_is and gives a bound on the probability that a randomly chosen solution will have distinct λ_is.

Fact 3.3. *(Gao, 2003, Theorems 2.8 and 2.10) Given a bivariate polynomial $f \in \mathbb{C}[x, y]$, suppose that $[g_1, h_1], \ldots, [g_r, h_r]$ (thought of as vectors of their coefficients) form a basis for the nullspace of $\mathrm{Rup}_1(f)$ over $\mathbb{C}$.*

Select $s_i \in S \subset \mathbb{C}$ uniform randomly and independently for all $1 \leq i \leq r$, and let $g = \sum_{i=1}^{r} s_i g_i$.

There is a unique $r \times r$ matrix $A = [a_{i,j}]$ over $\mathbb{C}$ such that

$$g g_i \equiv \sum_{j=1}^{r} a_{i,j} g_j f_x \,(mod\ f) \text{ in } \mathbb{C}(y)[x]. \tag{3.5}$$

Furthermore, let $E_g(x) = \det(Ix - A)$, the characteristic polynomial of A. Then the probability that

$$f = \prod_{\lambda \in \mathbb{C}:\ E_g(\lambda)=0} \gcd(f, g - \lambda f_x) \tag{3.6}$$

gives a complete factorization of f over $\mathbb{C}$ is at least $1 - r(r-1)/(2|S|)$.

Generalization to the multivariate case can be done with a straightforward extension of Gao's proofs.

3.1.4 The Algorithm

Algorithm 3.1 (FMP: Factor Multivariate Polynomial).

INPUT: A polynomial $f \in \mathbb{C}[x, y_1, \ldots, y_m]$ such that f is square-free

OUTPUT: A list of factors of f over $\mathbb{C}[x, y_1, \ldots, y_m]$

Let $\deg_{y_i}(f) = e_i > 1$ and let S be a finite set $S \subset \mathbb{C}$ with $|S| \geq \prod_i e_i$

1. Compute null vectors:

   (a) Form the matrix $\mathrm{Rup}_1(f)$ from the linear equations (2.1) and (3.4).

   (b) Form a basis $g_1, \ldots, g_r$ of the null space of $\mathrm{Rup}_1(f)$.

2. Compute a separable E_g

   (a) Choose $s_i \in S$, uniform randomly and independently, and set $g := \sum_{i=1}^{r} s_i g_i$.

   (b) Select randomly proper values for variable $x_i = \alpha_i$, $i \neq 1$ that do not change the degree (in x_1) or the square-free property of f.

   (c) For $x_i = \alpha_i$, compute $a_{i,j}$ so that the remainders vanish:

$$\mathrm{rem}(gg_i - \sum_{j=1}^{r} a_{i,j} g_j f_x, f) = 0.$$

   (d) Let $E_g(x) = \det(Ix - A)$, where $A = [a_{i,j}]$. If the roots $\lambda_i, 1 \leq i \leq r$ of E_g are not distinct go back to Step 2.1 (by 3.3 the probability that E_g has distinct roots is high).

3. Compute factors via GCDs Compute $f_i = \gcd(f, g - \lambda_i f_x)$ over $\mathbb{C}[x, y_1, \ldots, y_m]$ for $1 \leq i \leq r$.

4. Output the factors $f_1, \ldots, f_r$.

   Note that step 3 can be replaced with a resultant computation as noted above.

3.2 Exploiting Polynomial Structure

Using information about the structure of solutions to the PDEs (2.1) given in Lemma 3.1 it is possible for Theorem 2.1 to be adapted to account for the structure of f. This will allow us to apply the results of Theorem 2.9 with the additional requirement that perturbations preserve the (Newton polytope of the) support of f.

The following is a direct generalization of Lemma 4 in Gao and Rodrigues (2003); the proof is essentially the same.

Lemma 3.4. *Let $P(f)$ be the Newton polytope of f, where f is as in Lemma 3.1. Then if $(g, h_1, \ldots, h_m)$ satisfies (2.1) and (2.2) then*

$$P(x\,g) \subseteq P(f) \text{ and } P(y_i\,h_i) \subseteq P(f).$$

Proof. Recall that $P(a + b) \subseteq P(a) \cup P(b)$ and $P(a\,b) = P(a) + P(b)$, where '+' is the Minkowski sum of polytopes. Also, note that $P(x\,\partial f/\partial x) \subseteq P(f)$. Given the structure

of g in Lemma 3.1 we consider the polytopes of one of its summands:

$$P\left(x\lambda_j \frac{f}{f_j}\frac{\partial f_j}{\partial x}\right) = P\left(\frac{f}{f_j}\right) + P\left(x\frac{\partial f_j}{\partial x}\right) \subseteq P\left(\frac{f}{f_j}\right) + P\left(f_j\right) = P(f).$$

Therefore

$$P(x\,g) = P\left(\sum_{j=1}^{r} x\lambda_j \frac{f}{f_j}\frac{\partial f_j}{\partial x}\right) \subseteq P(f).$$

Similarly,

$$P(y_i\,h_i) = P\left(\sum_{j=1}^{r} y_i\lambda_j \frac{f}{f_j}\frac{\partial f_j}{\partial y_i}\right) \subseteq P(f).$$

□

If f has the structure $P(f) \subset S$, then Lemma 3.4 allows us to form a variant of the Ruppert matrix which has only the rows corresponding to monomials in the convex hull of S and columns corresponding to the monomials in the convex hull of the supports of g and the h_is indicated in Lemma 3.4 (see figure 2.1). Let us denote this variant as $\mathrm{Rup}_S(f)$. Then, given $\mathrm{Rup}_S(f)$, the nearest $\mathrm{Rup}_S(\tilde{f})$ that is singular gives $\tilde{f}$, a reducible polynomial that has the structure S as well.

Now, even though we require f to be square-free in some of the previous results (i.e. the ones that give explicit descriptions of the solutions g and h_i) we have structured irreducibility results that apply to all f. The only problem that could arise is that $\tilde{f}$, the closest factorizable polynomial to f with $P(\tilde{f}) = P(f)$, might not be square-free or primitive in which case it could be that $\mathrm{Rup}_S(\tilde{f})$ is full rank when $\mathrm{Rup}(\tilde{f})$ is not. This seems to be possible since if f is not square-free, it is possible for there to be very

different solutions to (2.1) than the ones given in Lemma 3.1. However, even if $\tilde{f}$ is not square-free and primitive, the corresponding equations (2.1) and (2.2) still have solutions of the form given in Lemma 3.1 (see the first part of the proof of Theorem 2.1). Loss of the square-free condition breaks the proof that these are the only solutions, but the proof that these are solutions still holds. Lemma 3.4 only describes the support of the solutions given in Lemma 3.1 and does not depend on the fact that f is square-free or primitive. So, for any f, if it is irreducible then $\mathrm{Rup}_S(f)$ is full rank and if f is not irreducible then $\mathrm{Rup}_S(f)$ is not full rank. Thus we have the following structured version of Theorems 2.1 and 2.9:

Corollary 3.5. *Let $f \in \mathbb{C}[x, y_1, \ldots, y_m]$, $S = P(f)$, and $\mathrm{Rup}_S(f)$ be the matrix described above (i.e. the matrix of the equations (2.1) and (2.2) with the supports of the indeterminant polynomials restricted so that $P(x\,g) = P(y_i\,h_i) = S$). Then we have*

1. *$\mathrm{Rup}_S(f)$ is full rank if and only if f is absolutely irreducible*

2. *Suppose f is an irreducible polynomial, $\tilde{f} \in \mathbb{C}[x, y_1, \ldots, y_m]$ is a polynomial of equal or lesser degree, $P(\tilde{f}) \subseteq S$, and there exists C so that*

$$\left\|\mathrm{Rup}_S(\varphi)\big|_{\varphi=f_\triangle}\right\|_{p,q} \leq C\,\|f_\triangle\|_r.$$

If

$$\|f - \tilde{f}\|_r < (C\,\|\mathrm{Rup}_S(f)^\dagger\|_{q,p})^{-1}.$$

then $\tilde{f}$ is irreducible.

If S is the set of all monomials with multi-degree less than or equal to $\mathrm{mdeg}(f)$ then this Corollary is the same as the results in Chapter 2. If S is the set of all monomials with total degree less than or equal to $\mathrm{tdeg}(f)$ then we get the results of Chapter 2 for total degree as in the first example below. It also allows us to find lower bounds on distance to nearest polynomials with more generally constrained support as in the second example below.

Example 3.1. Let $f = x^2+y^2-1$ as in examples 2.1 and 2.2. Notice that $\mathrm{mdeg}\, f = (2,2)$ and that the computed Ruppert matrix for f considered all monomials t with $\mathrm{mdeg}\, t \leq (2,2)$ (a rectangular support). The support of f is actually smaller than the rectangle since $\mathrm{tdeg}\, f = 2$. So for f, the monomials inside the convex hull of its support is the set $S = \{1, x, y, xy, x^2, y^2\}$ (all monomials with $\mathrm{tdeg}\, t \leq 2$).

Using the above, we can compute a bound on the distance to the nearest factorizable polynomial to f with total degree 2. We start with indeterminant g and h so that $\mathrm{mdeg}\, g = \mathrm{mdeg}(f) - 2$, $\mathrm{mdeg}\, h = \mathrm{mdeg}(f) - 1$, $x\, P(g) = S$, and $y\, P(h) = S$. That is:

$$g = b_1 + b_2\, y\,,\ h = c_1 + c_2\, x + c_3\, y.$$

Plugging g and h into the pde (2.1) we arrive at the structured Ruppert matrix

$$\mathrm{Rup}_S(f) = \begin{bmatrix} 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 \\ -2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & -1 & 0 & -1 & 0 \end{bmatrix}$$

which is almost half the size of $\mathrm{Rup}(f)$ (computed in example 2.1). The smallest singular value of $\mathrm{Rup}_S(f)$ is ≈ 1.4142135624.

Following example 2.2 we compute $\mathrm{Rup}_S(\varphi)$ for symbolic φ with $P(\phi) = S$. Doing so, we find the largest coefficient of $\|\mathrm{Rup}_S(\varphi)\|_2^2$ is 10. So a lower bound on the distance to the closest polynomial to f that has total degree 2 is 0.4472135954. Comparing to example 2.2 we see that this structured bound is quite a bit larger than the unstructured bound of 0.1584349745, which is expected. Also in example 2.2, we saw that the distance from f to nearest polynomial with total degree 2 is 1 so this lower bound is not tight, but not too bad. □

Example 3.2. Let us look at an example with a smaller support. Let $f = x^3\,y^2 + x^2\,y^3 - 1$ which has support $S = P(f) = \{1, xy, x^2y^2, x^3y^2, x^2y^3\}$. Then the structured g and h are

$$g = b_1\,y + b_2\,xy^2 + b_3\,xy^3 \text{ and } h = c_1x + c_2\,x^2y + c_3\,x^3y + c_4\,x^2y^2$$

which lead to a $\mathrm{Rup}_S(f)$ which is 9×7 (vs $\mathrm{Rup}(f)$'s 20×24). Its smallest singular value is 1 and the largest coefficient of the 2-norm of $\mathrm{Rup}_S(\varphi)$ is 32 which leads to a lower bound of $1/\sqrt{32} \approx 0.1767766953$. As expected, the unstructured bound is much smaller at 0.0380267374. □

3.3 Other Fields

Although the results of this chapter are stated for the fields $\mathbb{Q}$ and $\mathbb{C}$, some of them certainly apply to other fields. First, all of these results directly apply to other fields of characteristic 0. As is often the case, these results are also applicable for fields of large positive characteristic. In Gao (2003) the bivariate version of the factorization algorithm given above is shown to work in fields where the characteristic p is larger than $(2\deg_x(f) - 1)\deg_y(f)$ with the conjecture that it also works so long as p is larger than $\max\{\deg_x(f), \deg_y f\}$. The proof of the bound $(2\deg_x(f) - 1)\deg_y(f)$ could be generalized to the multivariate case, but the bound becomes quite large and it seems clear that it is not efficient to use this algorithm in fields of non-zero characteristic. For finite fields, in the case of more than two variables, it is probably best to use a substitution to project down to two variables and later lift the results.

Chapter 4

Approximate GCD Computation

It is our goal to adapt the factorization algorithm from Chapter 3 to polynomials with approximately given coefficients. In order to actually compute the factors in that algorithm, we must be able to compute the GCD of two multivariate polynomials that in turn will require computation of approximate division. Thus, if we wish to create an approximate factorization algorithm, we must first create an approximate GCD algorithm and an approximate division algorithm. The algorithm presented in this chapter will use the same SVD approach that will be used for the approximate factorization algorithm in Chapter 5. Though we considered multiple norms in Chapter 2, we will restrict ourselves to considering the 2-norm in the following chapters since our primary too, SVD, applies to that norm only.

4.1 Approximate Polynomial Division

The simplest interesting problem in approximate polynomial algebra seems to be the problem of polynomial "exact" division. Multiplication by a given polynomial is a linear operation so we can represent multiplication of polynomials of total degree d by a given f as $C^{[d]}(f)$, the convolution matrix associated with f and d. Note that the convolution matrix can be formed for other notions of degree (or polynomials with a given support), but for simplicity of discussion we will use total degree exclusively in this chapter.

4.1.1 Least Squares Division

If we are given polynomials f and g with $\text{tdeg}(g) \geq \text{tdeg}(f)$ such that f does not divide g exactly then we want to apply a perturbation so that f does divide g. If we fix the coefficients of f then $\tilde{g}$, the closest polynomial to g that f divides, can be found by solving the least squares problem:

$$\min_{\text{tdeg}(q)=\text{tdeg}(g)-\text{tdeg}(f)} \|f\,q - g\|_2.$$

We can write $\tilde{g}$ exactly in terms of a convolution matrix, $\tilde{g} = C^{[\text{tdeg}(g)]}(f)\,(C^{[\text{tdeg}(g)]}(f))^{\dagger}\,g$ (where we are being intentionally sloppy about the distinction between g as a polynomial and g as a vector of its coefficients). One of the shortcomings of this approach is, although it does solve the approximation problem completely, it does not allow for f to vary as well (or instead of) g.

4.1.2 SVD Based Division

If we are allowed to perturb the coefficients of both f and g, then the above becomes a total least squares (TLS) problem (perturbations are allowed in the right-hand side vector as well as the entries of the matrix). In fact, since the matrix $C^{[d]}(f)$ has a very specific structure, approximate division becomes a structured total least squares (STLS) problem. The TLS problem can be solved using low rank approximation using the SVD ideas seen in Section 2.2. Unfortunately, in general, STLS can be only be solved heuristically (c.f. Chu *et al.* (2003) for example).

The approximate division problem is actually a special case of the approximate GCD problem where one is looking for a GCD of specified degree: $\deg f = \max\{\deg f, \deg g\}$. Thus, we cannot necessarily expect to find a solution to the STLS division problem independent of a solution to the STLS GCD problem.

4.2 SVD Based Approximate Multivariate GCD

4.2.1 Algorithm Description

An approximate version of Gao's factorization needs not only an approximate multivariate GCD algorithm but in fact requires one that will handle the case when the given polynomials are quite far from having a common factor.

We first make some general comments on GCDs of arbitrary multivariate polynomials. The following simple lemma is the key to our approximate GCD algorithm.

Lemma 4.1. *Let $g, h \in \mathbb{C}[x_1, \ldots, x_n]$, both nonzero. Let $g_1 = g/\gcd(g,h)$ and $h_1 = h/\gcd(g,h)$. Then all the solutions $u, v \in \mathbb{C}[x_1, \ldots, x_n]$ to the equation*

$$ug + vh = 0 \tag{4.1}$$

must be of the form

$$u = h_1 q, \quad v = -g_1 q, \tag{4.2}$$

where $q \in \mathbb{C}[x_1, \ldots, x_n]$.

The proof of the lemma is straightforward since $\mathbb{C}[x_1, \ldots, x_n]$ is a unique factorization domain (think of (4.1) as $\frac{u}{v} = \frac{h}{g}$ where $\frac{u}{v}$ is the unknown reduced rational function form of $\frac{h}{g}$). Note that the equation (4.1) is a linear system for the coefficients of u and v. To make it a finite system, we need to restrict the degrees of u and v. There are several ways to do this. For example, one can consider the total degree, the individual degrees for each variable, or any weighted degree. Of the possible degree choices, we will consider the total degree and require that

$$\mathrm{tdeg}(u) \leq \mathrm{tdeg}(h) - 1, \quad \mathrm{tdeg}(v) \leq \mathrm{tdeg}(g) - 1. \tag{4.3}$$

Then $\gcd(g,h) = 1$ iff (4.1) and (4.3) have no nonzero solution for u and v.

For univariate polynomials, the coefficient matrix for the linear system (4.1) and (4.3) is nothing but the well-known Sylvester matrix for g and h. In Corless *et al.* (1995) the Sylvester matrix is used to get an approximate algorithm for the univariate GCD. For

multivariate polynomials, we still call the coefficient matrix corresponding to (4.1) and (4.3) the Sylvester matrix for g and h. We shall apply it to multivariate GCDs. Note that the cofactors g_1 and h_1 appear as the solution with the smallest degree; they are the solution we are looking for.

In general, there is an explicit relation between the total degree of $\gcd(g, h)$ and the dimension of the solution space to (4.1). To see this, note that the number of terms $x_1^{i_1} \cdots x_n^{i_n}$ with total degree $\leq d$ is the binomial number

$$\beta(d, n) = \binom{d+n}{n}.$$

Hence u has $\beta(\mathrm{tdeg}(h) - 1, n)$ coefficients and v has $\beta(\mathrm{tdeg}(g) - 1, n)$ coefficients. Thus the number of variables for the linear system is

$$m = \beta(\mathrm{tdeg}(g) - 1, n) + \beta(\mathrm{tdeg}(h) - 1, n).$$

By (4.2) and (4.3), all the solutions for u and v are determined by $q \in \mathbb{C}[x_1, \ldots, x_n]$ with

$$\mathrm{tdeg}(q) \leq \ell - 1,$$

where ℓ is the total degree of $\gcd(g, h)$. Hence the dimension of the solution space for u and v is exactly

$$\beta(\ell - 1, n).$$

Therefore one can compute the rank p of the coefficient matrix and then determine ℓ directly from $m - p = \beta(\ell - 1, n)$.

We will denote the Sylvester matrix of g and h as Syl_1. To find the GCD we need to find a basis for the nullspace of Syl_1. In the case of exact arithmetic, the cofactors g_1 and h_1 can be found by performing Gaussian elimination on the nullspace basis. The GCD can then be found by division. Doing this numerically, we face two difficulties: First, Syl_1 may be full rank in exact arithmetic, and second, recovering the smallest degree polynomial from the nullspace using Gaussian elimination is not stable numerically. To overcome the first difficulty, we use singular value decomposition to find the nearest matrix with (lower) rank p. The last $m - p$ singular vectors form a basis for the nullspace of this nearest low rank matrix. To determine what p should be, we look at the singular values of Syl_1. Typically, when determining rank numerically, one would specify a tolerance ϵ and find a gap in the singular values:

$$\sigma_m \geq \cdots \geq \sigma_{m-p-1} > \epsilon \geq \sigma_{m-p} \geq \cdots \geq \sigma_1.$$

Such a gap is treated as the boundary such that everything larger is considered non-zero while everything smaller is considered to be indistinguishable from zero. Given a numerical tolerance on the coefficients of the polynomials g and h there are various ways to derive such an ϵ, depending on what one is trying to achieve (see Corless *et al.* (1995) for example).

We, however, do not wish to specify an ϵ in advance, instead we try to infer the "best"

ϵ from the largest gap (i.e. the largest ratio σ_{i+1}/σ_i) in the singular values. In Emiris *et al.* (1997), it is shown that when given a tolerance ϵ it is possible to certify the degree of the approximate GCD using gaps in the sequence $\tau_i = \sigma_1(\text{Syl}_i(g,h))$ instead of the singular values of Syl_1. The size of the gap needed to certify the degree is very large however, and we have found that in practice the largest gap in the τ_is seems to give the same degree as the largest gap in the σ_is most of the time.

Another approach is to evaluate g and h at all of their variables but one, and find the "best" degree of the univariate GCD. Our experiments show that this seems to work well when a small tolerance is given (in case one is going to use the GCD as a component in a factorization algorithm, the tolerance could be inferred from the factorization problem), but for larger tolerances does not work as well when inferring an ϵ from the singular values, as above. The advantage of evaluating to find the degree is that the univariate Sylvester matrix is a great deal smaller, so even if we have to compute SVDs for several evaluations, it will be much faster than computing the SVD of the multivariate Sylvester matrix.

The second difficulty, that of recovering the GCD from the nullspace of Syl_1, can be handled by removing rows and columns from Syl_1. Once we have determined what the numerical rank of Syl_1 should be, and hence he degree of the approximate GCD of g and h, we can take a sub-matrix of Syl_1 found by using stronger degree restrictions (on the unknown polynomials u and v) in the linear system $ug + vh = 0$ so that we have a new Sylvester matrix Syl_k that has a nullspace of dimension 1, where k is the degree

of $\gcd(g, h)$. In this case, the single basis vector for the nullspace will give a constant multiple of the cofactors g_1 and h_1. This null-vector can be computed numerically without computing the full SVD by using an iterative method. One such method that we found to work well in practice can be found in Li and Zeng (2003). Once we have our approximations of g_1 and h_1 we can compute an approximate GCD by doing approximate division. Since the approximate GCD could be computed from divided g_1 in g or h_1 in h, ideally a modified division should be used that computes a d that minimizes $\|h - d\,h_1\|^2 + \|g - d\,g_1\|^2$. A least squares style division can be easily modified to compute such a d.

A very similar multivariate approximate GCD algorithm was proposed in Zeng and Dayton (2004) but a pre-specified tolerance ϵ is required there. In addition, a Gauss-Newton iteration step is introduced to improve the GCD further. In practice just a few steps of iteration can improve the backward error by at least an order of magnitude and so it is usually worth the extra computation, especially when the g and h started quite close to a pair with a non-trivial GCD. For example, if g and h are nearly machine precision distance from a pair with an exact GCD, the approximate GCD computed from the SVD method is generally limited to about half of the machine precision, while Gauss-Newton iteration can usually pick up several more digits of precision. Consider the following

example (from a private communication with Zhonggang Zeng):

$$g = 9\,x^4\,y + 24\,x^3\,y^2 - 36\,x^2\,y^2 + 128\,x\,y^3 - 18\,x^3\,y$$
$$- 48\,y^3\,x^2 - 64\,y^4 - 45\,x^3 + 60\,x^2\,y - 180\,x\,y + 80\,y^2 + 90\,x^2$$

and

$$h = 12\,y^3\,x^2 + 32\,x\,y^4 + 56.0000040\,x\,y^3 - 3\,x^3\,y + 7\,x^2\,y^2 - (0.10 \times 10^{-5})\,x^2\,y$$
$$- 59.9999990\,y^2\,x - 64\,y^5 - 79.9999840\,y^4 + 80.0000200\,y^3 + 99.9999800\,y^2$$
$$+ 15\,x^2 - 95\,x\,y + 0.50 \times 10^{-5}\,x - 0.250 \times 10^{-4}\,y$$

that have an exact GCD of $5 - x\,y - 4\,y^2$. Using 16 digit hardware precision Zeng's SVD based code (implemented in MATLAB) finds an approximate GCD with about 8 digits accuracy:

$$- 5.000000034500952 + .9999999999999999\,x\,y$$
$$+ 4.000000023172650\,y^2 - 0.2223886833245011 \times 10^{-8}\,x$$
$$+ 0.1584772295651007 \times 10^{-7}\,y + 0.7429584859159186 \times 10^{-8}\,y^2\,x$$

After iterative refinement twice as many digits of accuracy in the approximate GCD are

obtained:

$$5.000000000000001 - 1.000000000000000\,x\,y - 4.000000000000001\,y^2.$$

Due to such easy improvements possible from iterative refinement, we integrated this step into our approximate GCD algorithm as well.

4.2.2 The Algorithm

Algorithm 4.1 (AMVGCD: Approximate Multivariate GCD).

INPUT: g and h in $\mathbb{C}[x_1, \ldots, x_n]$

OUTPUT: d, a non-constant approximate GCD of g and h

1. Determine k, the degree of the approximate GCD of g and h in one of two ways below:

   (a) Form $S = \mathrm{Syl}_1(g, h)$, the matrix of the linear system $ug + vh = 0$, where $g, h \in \mathbb{C}[x, y]$ with $\mathrm{tdeg}(u) < \mathrm{tdeg}(h)$ and $\mathrm{tdeg}(v) < \mathrm{tdeg}(g)$. Finding the largest gap in the singular values of S and inferring the degree from the numerical rank of S.

   (b) Computing the degrees of the GCDs of several random univariate projections of g and h by looking for the numerical rank of the corresponding univariate Sylvester matrices.

2. Reform S as $\mathrm{Syl}_k(g,h)$ that is, use $\mathrm{tdeg}(u) = \mathrm{tdeg}(h) - k$ and $\mathrm{tdeg}(v) = \mathrm{tdeg}(g) - k$ as the constraints on u and v in the linear system in the first step. This new S will have a dimension 1 nullspace.

3. Compute a basis for the nullspace of S by computing the singular vector corresponding the smallest singular value of S. This vector gives a solution $[u,v]^T$.

4. Find d, the approximate quotient of h and u (or g and v); alternately minimize $\|h - d\,u\|_2^2 + \|g + d\,v\|_2^2$, using least squares.

5. Use Gauss-Newton iteration to iteratively compute a local minimum ($[u,v,d]$) to $\|F(u,v,d) - [g,h]^T\|_2$, where $F(u,v,d) = [u\,d, v\,d]^T$.

If one wishes to specify a tolerance, then only the first step is affected. In that case it is possible that in this step we could find that $k = 0$, in which case the method would return $d = 1$, declaring g and h to be approximately relatively prime to the given tolerance.

4.2.3 Convergence of the Algorithm

In this section we restrict ourselves to bivariate case ($n = 2$) for ease of notation and discuss convergence of the algorithm without use of Gauss-Newton iteration. Discussion of convergence of the Gauss-Newton iteration step can be found in Zeng and Dayton (2004). Let us start with $\tilde{g}$ and $\tilde{h}$ relatively prime and normalized so that $\|\tilde{h}\|_2 = \|\tilde{g}\|_2 = 1$. Suppose that $\gcd(g,h) = d \neq 1$, $\mathrm{tdeg}(g) = \mathrm{tdeg}(\tilde{g})$, $\mathrm{tdeg}(h) = \mathrm{tdeg}(\tilde{h})$, $\|g - \tilde{g}\|_2 = \epsilon_1$,

and $\|h-\tilde{h}\|_2 = \epsilon_2$. We will show that as $\epsilon_1, \epsilon_2 \to 0$ that the computed approximate GCD for $\tilde{g}$ and $\tilde{h}$ converges to d.

Let $S = \mathrm{Syl}_k(g,h)$ where k is chosen so that S has rank p, and rank deficiency 1. Then $w = [u,v]$ is a basis for the nullspace of S, where $u = h/d$, and $v = -g/d$, and without loss of generality we can assume that u is unit length. Let $\tilde{S} = \mathrm{Syl}_k(\tilde{g},\tilde{h})$. We can bound the distance between these two Sylvester matrices:

$$\|S - \tilde{S}\|_2^2 \leq \|S - \tilde{S}\|_F^2 = a_1\epsilon_1^2 + a_2\epsilon_2^2 = \epsilon_3^2,$$

where

$$a_1 = \beta(\mathrm{tdeg}(h) - k, 2) \text{ and } a_2 = \beta(\mathrm{tdeg}(g) - k, 2)$$

depend only on k and the degrees of g and h.

We can use the SVD to find M so that:

$$\min_{\mathrm{Rank}(M)=\mathrm{Rank}(S)} \|\tilde{S} - M\|_2 = \sigma_{m-p}(\tilde{S}) \leq \epsilon_3.$$

Note that M is not a Sylvester matrix and $\|M - S\|_2 \leq 2\epsilon_3$. Now, M has a dimension 1 nullspace, so let $\widetilde{w} = [\tilde{u}, \tilde{v}]$ be the vector that spans the nullspace of M with $\|\tilde{u}\|_2 = 1$. Theorem 6.4 in Stewart (1973a) (reformulated for our purpose in (Golub and Van Loan, 1996, section 8)) bounds the distance between w and $\widetilde{w}$ in terms of ϵ_3 so that as $\epsilon_3 \to 0$, $\epsilon_4 = \|w - \widetilde{w}\|_2 \to 0$. Thus for sufficiently small ϵ_1 and ϵ_2 we have $\mathrm{tdeg}(\tilde{u}) = \mathrm{tdeg}(u)$.

In the following we will make repeated use of the multivariate factor coefficient bound

found in (Gelfond, 1960, pages 134-139): $\|f_1\|_2\|f_2\|_2 \leq 2^{\sum_i(\deg_{x_i} f)}\|f\|_2$, where $f = f_1 f_2$.

Now, using least squares division, we compute $\tilde{d}$ as the polynomial that minimizes

$$\epsilon_5 = \min_{\bar{d}:\ \mathrm{tdeg}(\bar{d})\leq\mathrm{tdeg}(d)} \|\bar{d}\,\tilde{u} - \tilde{h}\|_2.$$

We can bound ϵ_5:

$$\begin{aligned}\epsilon_5 = \|\tilde{d}\,\tilde{u} - \tilde{h}\|_2 &\leq \|d(\tilde{u}-u) - (\tilde{h}-h)\|_2 \leq \|d(\tilde{u}-u)\|_1 + \epsilon_2 \leq \|d\|_1\|\tilde{u}-u\|_1 + \epsilon_2 \\ &\leq \tbinom{k+2}{2}^{1/2}\|d\|_2\tbinom{\mathrm{tdeg}(h)-k+2}{2}^{1/2}\|\tilde{u}-u\|_2 + \epsilon_2 \leq a_4\,\epsilon_4 + \epsilon_2,\end{aligned}$$

where

$$a_4 = 2^{\deg_x(h)+\deg_y(h)}\,\|h\|_2\tbinom{k+2}{2}^{1/2}\tbinom{\mathrm{tdeg}(h)-k+2}{2}^{1/2}$$

via the multivariate factor coefficient bound. Now,

$$\begin{aligned}\|(d-\tilde{d})u\|_2 = \|h - \tilde{d}\,\tilde{u} - \tilde{d}(u-\tilde{u})\|_2 &\leq \|h - \tilde{h} + (\tilde{h} - \tilde{d}\,\tilde{u})\|_2 + \|\tilde{d}(u-\tilde{u})\|_2 \\ &\leq \|h-\tilde{h}\|_2 + \|\tilde{h} - \tilde{d}\,\tilde{u}\|_2 + \|\tilde{d}(u-\tilde{u})\|_1 \leq \epsilon_2 + \epsilon_5 + \|\tilde{d}\|_1\|u-\tilde{u}\|_1 \\ &\leq 2\epsilon_2 + a_4\epsilon_4 + \tbinom{k+2}{2}^{1/2}\|\tilde{d}\|_2\tbinom{\mathrm{tdeg}(h)-k+2}{2}^{1/2}\epsilon_4,\end{aligned}$$

where $\|\tilde{d}\|_2$ is bounded as follows:

$$\begin{aligned}\|\tilde{d}\|_2 &\leq 2^{\deg_x(h)+\deg_y(h)}\|\tilde{d}\,\tilde{u}\|_2 \leq 2^{\deg_x(h)+\deg_y(h)}(\|\tilde{h}\|_2 + \|\tilde{d}\,\tilde{u} - \tilde{h}\|_2) \\ &\leq 2^{\deg_x(h)+\deg_y(h)}(\|\tilde{h}\|_2 + \epsilon_2 + a_4\epsilon_4).\end{aligned}$$

Thus, using the multivariate factor coefficient bound again, we have that

$$\|d - \tilde{d}\|_2 \leq 2^{\deg_x(h)+\deg_y(h)}(2\epsilon_2 + a_5\epsilon_4),$$

where a_5 is a constant (depending only on the degrees and norms of $\tilde{g}$ and $\tilde{h}$) which can be derived explicitly from the previous two estimates. So, as $\epsilon_1, \epsilon_2 \to 0$, then $\epsilon_3 \to 0$ so (as shown above) $\epsilon_4 \to 0$ and hence $\|d - \tilde{d}\|_2 \to 0$. It should be noted that in practice, we seem to always be able to compute $\tilde{d}$ very much closer than given bound. This suggests that it might be possible to compute a better bound for $\|d - \tilde{d}\|_2$, one that is not exponential in the degree of g and h.

Chapter 5

Approximate Factorization

In this chapter we adapt the factorization algorithm given in Chapter 3 for approximate factorizing using many of the same ideas as those used for the approximate GCD in Chapter 4.

5.1 The Factorization Algorithm and Experiments

In order to apply the factorization algorithm FMP given in Section 3.1.4 to approximate polynomials we must be able to solve the following problems:

1. compute the approximate GCDs of bivariate polynomials: $\gcd(f, g - \lambda_i f_x)$,
2. reduce the polynomial f so that $\gcd(f, f_x) = 1$ approximately,
3. determine the numerical dimension of G, and
4. compute an E_g that has no cluster of roots.

For the first problem, Chapter 4 provides a robust algorithm to compute the approximate GCDs of multivariate polynomials. The second problem is also handled by way of the approximate GCD; we can compute the approximate GCD of f and f_x. Then with an approximate division, $f/\gcd(f, f_x)$, we may, heuristically, reduce to the case where $\gcd(f, f_x) = 1$ approximately. Details on this approach follow below in Section 5.1.2.

To solve the third problem, we proceed similarly to the discussion in Section 4.2; we can determine the numerical dimension of G by the SVD of the matrix $\mathrm{Rup}_1(f)$. Let σ_i be the i^{th} singular value of $\mathrm{Rup}_1(f)$. If a tolerance ϵ is given, then the numerical dimension of G is the r such that

$$\cdots \geq \sigma_{r+2} \geq \sigma_{r+1} > \epsilon \geq \sigma_r \geq \cdots \geq \sigma_1.$$

However, if we do not know the relative error in the coefficients of f, it is difficult to provide a tolerance ϵ that is consistent with the error in the data. As before, if we have no tolerance given, we infer a tolerance from the largest gap in the singular values. That is, we choose $\epsilon = \sigma_r$ so that σ_{r+1}/σ_r is as large as possible. Looking back to Chapter 2, the singular value σ_r bounds the distance from f to a polynomial $\tilde{f}$ that has r absolutely irreducible factors:

$$\min_{\substack{\deg \tilde{f}=(m,n) \\ \dim \mathrm{Nullspace}(\mathrm{Rup}_1(\tilde{f}))=r}} \|\mathrm{Rup}_1(f) - \mathrm{Rup}_1(\tilde{f})\|_2 \geq \sigma_r.$$

This inferred tolerance σ_r can also be used in estimating the degree of multivariate approximate GCD by using it as a tolerance when projecting to univariate GCD problems, as was mentioned in Section 4.2.

For the fourth problem, suppose we have obtained approximate basis $g_1, \ldots, g_r$ of G from the singular vectors corresponding to the last r singular values of $\mathrm{Rup}_1(f)$. It is easy to see that $\|\mathrm{Rup}_1(f)g_i\|_2 \leq \sigma_i \leq \sigma_r$. So the g_is form an approximate basis for G with tolerance σ_r. Following construction of the matrix A_g as described in Fact 3.3, we find a random element of G by choosing $s_1, \ldots, s_r \in S \subset \mathbb{C}$ uniform randomly and let $g = \sum_{i=1}^{r} s_i g_i$ and substitute arbitrary values of $\alpha_i \in \mathbb{C}$ for y_i with the property that $f(x, \alpha_1, \ldots, \alpha_m)$ remains square-free. The matrix A_g can be formed in the following manner: first reduce the polynomials gg_i and $g_j f_x$ modulo f (evaluated at $y_k = \alpha_k$) for $1 \leq i, j \leq r$ by using approximate division of univariate polynomials Zhi (2003); then solve the least squares problem:

$$\min \|\mathrm{rem}(gg_i - (a_{i,1}g_1 f_x + \cdots + a_{i,r}g_r f_x),\ f)\|_2$$

to find the value of unknown elements $a_{i,j}$. Let $E_g(\lambda) = \det(I\lambda - A)$, the characteristic polynomial of A_g. We compute all the numerical roots $\lambda_1, \ldots, \lambda_r$ of the univariate polynomial E_g over $\mathbb{C}$ as the eigenvalues of A_g, and find the smallest distance between these roots:

$$min\_dist(g) = \min\{|\lambda_i - \lambda_j|,\ \ 1 \leq i < j \leq r\}.$$

If the distance is small then numerically E_g has a cluster of roots, and we should choose another set of s_is and try to find a separable E_g. In practice, since Fact 3.3 says g should give a separable E_g with high probability, we compute a number of random gs and keep the g with the largest $min\_dist(g)$.

In Chapter 3 the absolutely irreducible factors are obtained from g by computing GCDs over algebraic extension fields given by the irreducible factors of E_g. In our case, all the roots of E_g are given as numerical values in $\mathbb{C}$. Hence there is no need to deal with field extensions, and we can compute directly in $\mathbb{C}$. We compute the multivariate approximate GCDs $\tilde{f}_i = \gcd(f, g - \lambda_i f_x)$ according to the method in Section 4.2 for each numerical root λ_i of E_g and we obtain a proper approximate factorization of f over $\mathbb{C}$: $f \approx \prod_{i=1}^{r} \tilde{f}_i$.

One we have computed an approximate factorization, there are a number of ways to improve it. First, we can compute a scaling c that minimizes the backward error of the approximate factorization:

$$\min_{c \in \mathbb{C}} \|f - c \prod_{i=1}^{r} \tilde{f}_i\|_2 / \|f\|_2.$$

The factorization can be improved further by solving a minimization problem (for example the one in Huang *et al.* (2000)) or by setting up a minimization problem to which we can apply Gauss-Newton iteration, similar to what was done to refine the approximate GCD in Zeng and Dayton (2004). First note that the optimization version of the approximate factorization problem is finding a least squares solution to the non-linear system

of the form $F(v_1, \ldots, v_r) = f$ where $v_i \in \mathbb{C}[x, y_1, \ldots, y_m]$ and

$$F(v_1, \ldots, v_r) = \left[\begin{array}{c} C^{[\mathrm{tdeg}(v_2 \cdots v_r)]}(v_1) \cdots C^{[\mathrm{tdeg}(v_r)]}(v_{r-1})\, v_r \end{array} \right].$$

Here $C^{[k]}(v)$ denotes the matrix of the linear map multiplication with polynomials of total degree k as described in Section 4.1. Clearly there is a solution when $f = f_1 \cdots f_r$ and $v_i = f_i$; otherwise we will solve

$$\min_{v_i} \|F(v_1, \ldots, v_r) - f\|_2.$$

There exists such a minimum at one or more of the points where

$$(DF(v_1, \ldots, v_r))^H \, F(v_1, \ldots, v_r) = 0$$

(DF denotes the Jacobian of F). When formulated this way, it is easy to see that we can apply Gauss-Newton iteration to attempt to find the solution. That is, given an initial $[v_1^0, v_2^0, \ldots, v_r^0]$ we refine with the update

$$[v_1^{i+1}, \ldots, v_r^{i+1}] = [v_1^i, \ldots, v_r^i] - (DF(v_1^i, \ldots, v_r^i))^\dagger \, F(v_1^i, \ldots, v_r^i).$$

Given the description of F above, the product rule gives that the Jacobian of F is a block matrix of the form:

$$DF(v_1, \ldots, v_r) = [C^{[\mathrm{tdeg}(v_1)]}(v_2\, v_3 \cdots v_r)\, C^{[\mathrm{tdeg}(v_2)]}(v_1\, v_3 \cdots v_r) \ldots C^{[\mathrm{tdeg}(v_r)]}(v_1\, v_2 \cdots v_{r-1})]$$

which has full rank (so long as not all the v_i's are 0) since every matrix $C^{[k]}(v)$ has full rank (so long as $v \neq 0$).

As with any type of Newton method, if the initial input $[v_1^0, v_2^0, \ldots, v_r^0]$ is close enough and DF is not rank deficient at the least squares solution, then the iteration will converge to the least squares solution according to Kelley (1999, Theorem 2.4.1):

Fact 5.1. *Let $w_0 = [v_1^0, \ldots, v_r^0]$ be the initial point and $w_\star = [v_1^\star, \ldots, v_r^\star]$ be a local minimum for F. If DF is full rank then there exist $K > 0$ and $\delta > 0$ so that if $\|w_0 - w_\star\| < \delta$ then the error of the Gauss-Newton iteration update at step k (e_k) satisfies:*

$$\|e_k\|_2 < K\,(\|e_{k-1}\|_2^2 + \|F(w_\star) - f\|_2\, \|e_{k-1}\|_2).$$

Although, as with the GCD, it is possible to bound the distance of the output of the SVD method for factorization from the closest approximate factorization, that bound is quite large (exponentially large in the degree). We need a much tighter bound in order to prove something about when the output of the SVD method will be within the basin of attraction of the global minimum. Finally, it is worth mentioning that Fact 5.1 implies that Gauss-Newton iteration converges at a quadratic rate if the nearby local

minimum is an exact factorization of f. Otherwise, iteration converges at a linear rate that is inversely proportional to the error of the factorization at the global minimum ($\|F(w_\star) - f\|_2$). In practice, the iteration converges in very few steps (≈ 7 for most polynomials tested).

5.1.1 Algorithm

Algorithm 5.1 (AFMP: Approximate Factoring Multivariate Polynomials).

INPUT: A polynomial $f \in \mathbb{C}[x, y_1, \ldots, y_m]$ such that f and f_x are approximately relatively prime, that is f is approximately square-free and has no approximate factors in $\mathbb{C}[y_1, \ldots, y_m]$ (see section 5.1.2 below).

OUTPUT: A list of approximate factors f_i and an optimal scaling c.

Let S be a finite set $S \subset \mathbb{C}$ with $|S| \geq \text{tdeg}(f)^2$.

1. Compute approximate nullspace solutions:

   (a) Form the matrix $\text{Rup}_1(f)$;

   (b) Compute the singular value decomposition of the Ruppert matrix, and find the last $\text{tdeg}(f) + 1$ singular values σ_i;

   (c) Find the biggest gap in the singular values and decide the numerical dimension r of G, assuming $r \geq 2$;

   (d) Form a basis $g_1, \ldots, g_r$ of G from the last r right singular vectors of $\text{Rup}_1(f)$.

2. Compute an E_g with well spaced roots:

   (a) Evaluate at randomly selected values for the variables $y_i = \alpha_i$ that do not change the degree or the square-free property of f;

   (b) For k from 1 to K do ($K = 4$ seems to work well in practice)

      i. Pick $s_{i,k} \in S$ randomly, and set $\mathsf{g}_k = \sum_{i=1}^{r} s_{i,k}\, g_i$

      ii. Compute $a_{i,j,k}$ that minimize the norm of the univariate remainder:

$$\min \|\mathrm{rem}(\mathsf{g}_k\, g_i - \sum_{j=1}^{r} a_{i,j,k} g_j f_x,\ f)\|_2;$$

      iii. Let $E_{\mathsf{g}_k}(x) = \det(Ix - A)$, where $[A_{\mathsf{g}_k}]_{i,j} = [a_{i,j,k}]$. Compute the numerical roots $\lambda_{i,k}, 1 \le i \le r$ of E_{g_k} (the numerical eigenvalues of A_{g_k}) and set $min\_dist_k = \min_{1 \le i < j \le r}\{|\lambda_{i,k} - \lambda_{j,k}|\}$;

   (c) Let $g = \mathsf{g}_k$ where min_dist_k is minimal.

3. Compute factors via approximate GCDs:

   Compute $f_i = \gcd(f, g - \lambda_i f_x)$ over $\mathbb{C}[x, y]$ for $1 \le i \le r$.

4. Solve optimizations to refine the factorization:

   (a) Apply Gauss-Newton iteration to improve the approximate factors;

   (b) Compute $\min_{c \in \mathbb{C}} \|f - c \prod_{i=1}^{r} f_i\|_2 / \|f\|_2$.

Remark 5.1. It should be clarified how an implementation should obtain the degree of the GCD in Step 3. It is fastest to project to univariate problems using a tolerance ϵ

in Step 1 of Algorithm AMVGCD in Section 4.2.1. The use of σ_r as the tolerance for the approximate GCD is not accurate due to the large norm of the projected univariate polynomials, and must be increased (e.g. multiplied by the ratio of the norm of the projected polynomial to the norm of the original polynomial) to obtain suitable GCDs.

Remark 5.2. It is clear that output polynomials can not be guaranteed to be approximately irreducible. For example, in the case that the input does not lie near a factorizable polynomial then the approximate GCDs may place a factor near a reducible polynomial. One may, of course, always achieve approximate irreducibility certification by applying the tests given in Chapter 2 to the produced factors and apply the algorithm again if necessary.

5.1.2 Multiple Factors

In the case that f is quite close to a polynomial that is not square-free, our factorization algorithm does not work well. This is related to the fact that the exact algorithm does not work at all on polynomials with repeated factors. In that case $\mathrm{Rup}_1(f)$ has many extraneous null vectors that do not correlate with factors (at least, not in the same way). When an irreducible polynomial is near a repeated factor polynomial, the approximate null vectors and numerical rank of $\mathrm{Rup}_1(f)$ exhibit some of these same problems. Another, similar but lesser problem is the removal of approximate factors in $\mathbb{C}[y_1, \ldots, y_m]$, that essentially amounts to a univariate approximate GCD computation.

One method to deal with the non-square-free case is to compute f_{sqfr}, the approxi-

mate quotient of f and the approximate GCD of f and f_x. Then compute the distinct approximate factors of $f_{\text{sqfr}} \approx f_1 \cdots f_r$ using our algorithm. Finally, determine powers for each factor by looking for gaps in the sequence $\alpha_{i,j} = \sigma_1(S_1(f_i, \partial_{x,j} f))$.

We can only definitively call f approximately square-free if all of the nearest polynomials that factor are square-free. We cannot compute the nearest polynomial that factors, but we can bound the distance to the nearest polynomial that factors using the singular values of $\text{Rup}_1(f)$ as in Kaltofen and May (2003), and similarly bound the distance to the nearest polynomial that is not square-free using the singular values of $S_1(f, f_x)$. If the two bounds are very close we have to compute the factorization both ways and use the one with smaller backwards error.

In Zeng and Dayton (2004) a different method is proposed, that is based entirely on multivariate approximate GCDs and that generalizes the univariate algorithm in Zeng (2003). Experimentally, the two approaches seem to work similarly well (compare the example 14 from the table below to the ASFF example in Zeng and Dayton (2004)).

5.1.3 Implementation and Experiments

The AFMP algorithm and its variants in Maple have been implemented in Gao *et al.* (2004) without using any Gauss-Newton iteration to improve approximate GCDs or the final factorization. Experiments with that implementation are shown in Table 5.1. Timings are given for some well known or randomly generated examples on a Pentium 4 at 2.0 Ghz for *Digits* = 10 in Maple 9 under Linux. Here σ_r and σ_{r+1} are singular values

Ex.	$\deg(f_i)$	$\frac{\sigma_{r+1}}{\sigma_r}$	$\frac{\sigma_r}{\|\mathrm{Rup}_1(f)\|_2}$	*coeff. error*	*backward error*	*time(sec)*
1	2,3	11	10^{-3}	10^{-2}	1.08e–2	14.631
2	5,5	10^{9}	10^{-10}	10^{-13}	8.30e–10	5.258
3	10,10	10^{5}	10^{-6}	10^{-7}	1.05e–6	85.96
4	7,8	10^{7}	10^{-8}	10^{-9}	1.41e–8	19.628
5	3,3,3	10^{8}	10^{-10}	0	1.29e–9	9.234
6	6,6,10	10^{3}	10^{-6}	10^{-5}	2.47e–4	539.67
7	9,7	486	10^{-4}	10^{-4}	2.14e–4	43.823
8	4,4,4,4,4	273	10^{-6}	10^{-5}	1.31e–3	3098
9	3,3,3	1.70	10^{-3}	10^{-1}	7.93e–1	29.25
10	12,7,5	658	10^{-6}	10^{-5}	1.56e–4	968
11	12,7,5	834	10^{-6}	10^{-5}	3.19e–4	1560
12	12,7,5	8.34	10^{-4}	10^{-3}	8.42e–3	4370
13	$5,(5)^2$	10^{3}	10^{-5}	10^{-5}	6.98e–5	34.28
14	$(5)^3,3,(2)^4$	10^{7}	10^{-9}	10^{-10}	2.09e–7	73.52
15	5,5	10^{4}	10^{-5}	10^{-5}	1.72e–5	332.99
15a	2,2	10^{9}	10^{-10}	10^{-4}	1.02e–9	13.009
16	18,18	10^{4}	10^{-7}	10^{-6}	3.75e–6	3173
17	18,18	10^{4}	10^{-7}	10^{-6}	4.10e–6	4266
18	6,6	10^{6}	10^{-8}	10^{-7}	2.97e–7	30.034

Table 5.1: Algorithm performance on benchmarks

around the biggest gap—the given values are orders of magnitude; *coeff. error* indicates the noise imposed on the input, namely the relative 2-norm coefficient error to the original product of polynomials. The *time* is that for the entire factorization in seconds of a single run; the timings can vary significantly (up-to a factor of 4) with the randomization.

The 23 test cases and the Maple implementation can be found online at `http://www.mmrc.iss.ac.cn/~lzhi/Research/appfac.html` or `http://www.math.ncsu.edu/~kaltofen/` (click on the "Software" link).

In Table 5.1:

- Example 1 is from Nagasaka (2002) where an approximate factorization with backward error 0.000753084 is also given (although this is slightly smaller than the backward error computed in the table the given factorization does not seem to have been generated with any sort of general technique and no timing was reported);

- Examples 2 and 3 are from Sasaki (2001); Sasaki's algorithm takes 430ms and 2080ms on a SPARC 5 (CPU: microSPARC II, 70 MHz) and produced backward errors of 10^{-9} and 10^{-5}, respectively;

- Example 4 is from Corless *et al.* (2001); the backward error for their approximate factorization is reported as 0.47×10^{-4}, compared to ours of backward error 0.14×10^{-7} (no timings were reported);

- Example 5 is from Corless *et al.* (2002), which is the factorization of an exact polynomial of degree 9 (here their and our backward errors are about the same; no timings were reported);

- Examples 6 to 13 and 15 to 17 were constructed by choosing factors with random integer coefficients in the range $-5 \leq c \leq 5$ and then adding a perturbation; for noise we choose a relative tolerance 10^{-e}, then randomly choose a polynomial that has the same degree as the product, 25% as many terms (5% for Example 10 and 99% for Example 17) and coefficients in $[-10^e, 10^e]$; finally, we scale the perturbation so that the relative error is 10^{-e};

- Examples 10, 11 and 12 approximately factorize the same polynomial with pertur-

bations of different noise level and sparseness;

- Example 13 and 14 have repeated factors denoted with exponents in the degrees column;

- Example 14 is Zeng's ASFF example in Zeng and Dayton (2004). The forward errors of the factors we compute are about 10^{-8}, similar to Zeng's forward error.

  By forward error we mean the relative 2-norm coefficient vector distance of a computed approximate factor to the nearest originally chosen factor, before noise was added to the product. For the examples our implementation produced forward errors that are of the same magnitude of the stated backward errors, with the exception of Example 9 where the degrees of the produced approximate factors are 4 and 5, hence the forward error is, in some sense, infinite. The approximate factorization is poor because better backward error can obtained simply by setting terms to 0. Polynomials with 10% relative noise are not handled well by our implementation.

- Example 15 and 15a are polynomials in three variables; 15a is from Kaltofen (2000);

- Example 18 is a polynomial with complex coefficients, where the real and imaginary parts of the coefficients of the factors were chosen random integers in $[-5, 5]$. Noise was added to the real and imaginary parts of all terms;

The implementation reported in Gao *et al.* (2004) also successfully found the approximate factors of four examples, provided by Jan Verschelde, which arise in the engineering of

Stewart-Gough platforms (see Sommese *et al.* (2004)). The input polynomials in 2 and 3 variables of degree 12 have small absolute coefficient error, 10^{-16}, and have approximate factors of multiplicities 1, 3 and 5. The trivariate approximate factors were computed via sparse numerical interpolation using the techniques of Giesbrecht *et al.* (2004), (which is possible in this example because the forward error in the approximate factor coefficients is near machine precision). The running times, no more than 200 seconds with a backward error of no more than $7.62 \cdot 10^{-9}$, appear much faster than what Sommese *et al.* (2004) report for their solution, though this is part due to the advantage gained by using the sparse interpolation code reported in Giesbrecht *et al.* (2004).

5.1.4 Iterative Refinement Implementation

To demonstrate the potential for improvement using iterative refinement, we created a basic Maple implementation to perform iterative refinement for factorizations of polynomials in two variables. This code was applied to several of the examples from the previous section and the results compiled in Table 5.2. The only two variable example omitted is 14 which is problematic due to not being content-free. The other non-square-free example, 13, is content-free, and the iteration works just fine. The improved factorization found by this method still has a squared factor even though the refinement iteration does not treat the identical factors differently than non-identical factors. It should be noted that refinement was not used on the approximate GCD computations performed in the AFMP algorithm, only on the factorization. Experiments seem to indicate that

Ex.	$\deg(f_i)$	*coefficient error*	*error w/o iter*	*error w/ iter*	*iterations*	*improvement*
1	2,3	10^{-2}	1.08e-2	1.02e-3	7	10.6×
2	5,5	10^{-13}	4.64e-10	4.90e-14	2	9468×
3	10,10	10^{-7}	1.05e-6	2.87e-7	3	3.6×
4	7,8	10^{-9}	1.41e-8	2.38e-9	2	5.9×
5	3,3,3	0	1.20e-9	1.03e-14	2	99613×
6	6,6,10	10^{-5}	2.47e-4	7.24e-6	4	34×
7	9,7	10^{-4}	2.14e-4	7.07e-5	4	3.0×
8	4,4,4,4,4	10^{-6}	1.31e-3	8.56e-6	4	153×
9	3,3,3	10^{-1}	7.93e-1	1.42e-1	16	5.6×
10	12,7,5	10^{-6}	1.56e-4	8.02e-6	4	19.5×
11	12,7,5	10^{-6}	3.19e-4	7.66e-6	4	41.7×
12	12,7,5	10^{-4}	8.42e-3	7.66e-4	6	11.0×
13	$5,(5)^2$	10^{-5}	1.72e-5	6.53e-6	3	10.7×
16	18,18	10^{-6}	3.75e-6	6.55e-7	3	5.6×
17	18,18	10^{-6}	4.10e-6	6.61e-7	3	6.2×
18	6,6	10^{-7}	3.00e-7	6.03e-8	2	4.9×

Table 5.2: Iterative refinement on most of the benchmarks from Table 5.1

using refinement on the approximate GCD computations leads to better pre-refinement factorizations, but that the Gauss-Newton iterations converge to the same factorization as when refinement was not used in the GCD computations.

Timings are not included in Table 5.2 since the iterative refinement code runs many times faster than the factoring code (less than 10% of the time of the original factorization). The next to last column indicates the number of Gauss-Newton iterations that were run before convergence – further iteration did not improve the factorization. Notice that the number of iterations increases as the backward error of the solution found (i.e. the number in the fifth column) increases (as discussed in the paragraph following Fact 5.1). The last column indicates the factor by which the backwards error was im-

proved by iterative refinement. Our experiments seem to indicate that refinement will tend improve backward error by about one order of magnitude. The improvement can be quite a bit more pronounced if the original polynomial was within machine precision of being factorizable. In example 9, the factorization found before refinement had backward error worse than 2.37e-1, the backward error of the trivially factorizable polynomial $f(x, y) - f(0, y)$, while that is beaten slightly after refinement. As can be seen by the number of iterations, when $\|noise\| \approx 10^{-1}$ it is still very difficult to get good results.

Chapter 6

Polynomial Decomposition

Given a polynomial $f \in \mathbb{Q}[x]$, we consider the problem of determining if f decomposes, that is, determining if there exists $g, h \in \mathbb{Q}[x]$, both of at least degree 2, such that $f(x) = g(h(x)) = (g \circ h)(x)$. The polynomial h is called a right composition factor of f. If f does decompose, we want to be able to compute its decomposition into indecomposable factors. We will consider approximate versions of these problems.

In this section we present a series of results on polynomial decomposition, including a new analysis of an older decomposition algorithm. Though not optimal, this new analysis will allow us to compute approximate polynomial decompositions by reducing them to approximate bivariate factorization problems.

6.1 Improvements to Barton-Zippel

The first known algorithm to compute the decomposition of a polynomial, given in Barton and Zippel (1976, 1985), relied on the following theorem of Fried and MacRae:

Fact 6.1 (Fried and MacRae (1969)). *Let x, y be independent indeterminants over $\mathbb{Q}$ and $f, h \in \mathbb{Q}[x]$. Then $h(x) - h(y)$ divides $\Phi(f) = (f(x) - f(y))/(x - y)$ if and only if $f(x) = g(h(x))$ for some $g \in \mathbb{Q}[x]$.*

This theorem leads directly to the decomposition algorithm from Barton and Zippel (1985):

Algorithm 6.1.

INPUT: A polynomial $f \in \mathbb{Q}[x]$.

OUTPUT: $g, h \in \mathbb{C}$ such that $f = g \circ h$, or f is indecomposable.

1. Form $\Phi(f) = (f(x) - f(y))/(x - y)$;
2. Factor $\Phi(f)$ completely over $\mathbb{C}[x, y]$; if $\Phi(f)$ is irreducible, f is indecomposable;
3. Examine the factors for polynomials of the form $\Phi(h)$ (if there are no irreducible factors of the right form, examine factors of two irreducible factors and so on);
4. If no factors of the form $\Phi(h)$ exist then f is indecomposable; otherwise, we have found a factor h from which we can compute g.

Note that in step 4 computing g from a given h is merely a matter of solving the

system:

$$f - \sum_{i=0}^{\deg f - \deg h} g_i \, h^i = 0 \tag{6.1}$$

which is a system of linear equations in $g_0, g_1, \ldots, g_{\deg f - \deg h}$, the coefficients of g.

The running times of steps 1, 2, and 4 of Algorithm 6.1 are clearly polynomial in the degree of f. The need to possibly check what could be exponentially many combinations of factors of f in step 3 prevents the algorithm from having polynomial running time. However, we can get around this exponential time step using the following stronger version of Fact 6.1 which was apparently unknown to Barton and Zippel in 1985.

Fact 6.2 (Fried (1970)). *Let $f \in \mathbb{Q}[x]$ be indecomposable of degree $n > 1$. Suppose that n is not an odd prime and it is not the case that $f(x) = \alpha D_n(a, x + b) + \beta$ for $\alpha, \beta, a, b \in \mathbb{Q}$, where $a = 0$ if $n = 3$.*

If $f(x)$ is indecomposable, then $(f(x) - f(y))/(x - y)$ is absolutely irreducible.

The notation $D_n(a, x)$ refers to a Dickson polynomial. The Dickson polynomials can be defined as the compositions of Chebyshev polynomials, linear polynomials, and polynomials of the form x^m. More explicitly, for any $n \geq 0$ and any $a \in \mathbb{Q}$ define the n^{th} Dickson polynomial as

$$D_n(x, a) = \sum_{i=0}^{\lfloor n/2 \rfloor} \frac{n}{n-i} \binom{n-i}{i} (-a)^i x^{n-2i},$$

which expands to

$$D_n(a,x) = \begin{cases} x^n - nax^{n-2} + n(n-3)/2 \cdot a^2 x^{n-4} + \cdots + 2(-a)^{n/2} & \text{for } n \text{ even;} \\ x^n - nax^{n-2} + n(n-3)/2 \cdot a^2 x^{n-4} + \cdots + n(-a)^{(n-1)/2} x & \text{for } n \text{ odd.} \end{cases}$$

A nice proof of Fact 6.2 can be found in Turnwald (1995).

For decomposition, Fact 6.2 implies that if f is not of prime degree and $\Phi(f)$ factors, then f decomposes. Also, if h is an indecomposable composition factor of f then $\Phi(h)$ is absolutely irreducible unless h is a Dickson polynomial. Thus, searching combinations of factors of $\Phi(f)$ will not be necessary unless f has only Dickson polynomials as right composition factors. That case is not a problem though, since we can detect if a polynomial f has Dickson polynomials as right composition factors simply by examining the first three coefficients of f as follows:

Lemma 6.3. *If f is monic, and has a Dickson polynomial composition factor of prime degree $q \geq 3$ then f has the form:*

$$f = x^n + nb\,x^{n-1} + \left(\frac{n(n-1)}{2}\,b^2 - n\,a\right) x^{n-2} + \cdots$$

Proof. If $f \in \mathbb{Q}[x]$ is monic and $f = g \circ D_q(a, x+b)$ then g is monic as well (since D_q is monic). If $q \geq 3$ is prime, then

$$D_q(a, x+b) = x^q + q\,b\,x^{q-1} + \left(\frac{q(q-1)}{2} b^2 - q\,a\right) x^{q-2} + \cdots$$

so

$$g(D_q(a, x+b)) = \left(x^q + q\,b\,x^{q-1} + \left(\frac{q(q-1)}{2} b^2 - q\,a \right) x^{q-2} + \cdots \right)^k + \cdots$$

$$= x^{q\,k} + k\,(q\,b\,x^{q-1})\,(x^q)^{k-1} + \frac{k(k-1)}{2}\,(q\,b\,x^{q-1})^2\,(x^q)^{k-2} + k\left(\frac{q(q-1)}{2} b^2 - q\,a \right) x^{q-2}\,(x^q)^{k-1} + \cdots$$

$$= x^n + nb\,x^{n-1} + \frac{n(n-q)}{2}\,b^2\,x^{n-2} + \left(\frac{n(q-1)}{2} b^2 - n\,a \right) x^{n-2} + \cdots$$

$$= x^n + nb\,x^{n-1} + \left(\frac{n(n-1)}{2}\,b^2 - n\,a \right) x^{n-2} + \cdots$$

□

Thus we have the following improved version of the Barton-Zippel algorithm over $\mathbb{Q}$.

Algorithm 6.2.

INPUT: A polynomial $f \in \mathbb{Q}[x]$.

OUTPUT: $g, h \in \mathbb{C}$ such that $f = g \circ h$, or f is indecomposable.

1. Form $\Phi(f) = (f(x) - f(y))/(x - y)$;

2. Factor $\Phi(f)$ completely; if $\Phi(f)$ is irreducible, f is indecomposable;

3. Examine the irreducible factors for polynomials of the form $\Phi(h)$;

4. (a) If no such h found, for each prime number q which divides $\deg f$ compute h

of the form $h(x) = D_q(a, x + b)$ with

$$b = f_{n-1}/n,\ a = \frac{1}{n}\left(\frac{n(n-1)}{2}\left(f_{n-1}/n\right)^2 - f_{n-2}\right)$$

where

$$f = x^n + f_{n-1}\,x^{n-1} + f_{n-2}\,x^{n-2} + \ldots$$

It will be possible to solve for g in (6.1) given one such h;

(b) If such an h is found, compute g from the system (6.1).

Algorithm 6.2 is clearly polynomial time since the factor combination of Algorithm 6.1 has been eliminated and step 4a involves trying fewer than $\deg f$ possibilities for q.

Algorithm 6.2 is not that useful for the exact decomposition problem. A faster polynomial time algorithm from Kozen and Landau (1989) and a nearly linear time algorithm in von zur Gathen (1990) have been known for over 15 years. However, Algorithm 6.2 reduced to a structured linear problem which allows us to apply many of the techniques from previous chapters to the approximate version of the decomposition problem.

6.2 Approximate Decomposability Testing

The relationship between decomposition and bivariate factorization presented in the previous section make it trivial to transform decomposition into a linear problem.

For a given $f = \sum_{i=0}^{d} c_i\,x^i \in \mathbb{Q}[x]$ (d not prime), we know f decomposes if and only

if

$$\Phi(f) = (f(x) - f(y))/(x - y) = \sum_{i=1}^{d} c_i \sum_{j+k=d-1} x^i y^j \tag{6.2}$$

factors in $\mathbb{C}[x, y]$. And, $\Phi(f)$ has factors if and only if the matrix $\mathrm{Rup}(\Phi(f))$ does not have full rank. We can bound the distance of an indecomposable f to a decomposable polynomial by bounding the distance of $\mathrm{Rup}(\Phi(f))$ to a matrix of lower rank as we did in Chapter 2.

Corollary 6.4. *If $f \in \mathbb{Q}[x]$ is an indecomposable polynomial, and $\tilde{f} \in \mathbb{Q}[x]$ is a decomposable polynomial with $\tilde{f}(0) = f(0)$ and $\deg \tilde{f} \leq \deg f$ then*

$$\|f - \tilde{f}\|_2 \geq \frac{\sigma(\mathrm{Rup}(\Phi(f)))}{d^2\sqrt{2d^2 - d}}$$

Proof. Since $\Phi(f)$ is irreducible and $\Phi(\tilde{f})$ is not, and $\deg_x \Phi(f) = \deg_y \Phi(f) = d$ we have, from Theorem 2.9,

$$\|\Phi(f) - \Phi(\tilde{f})\|_2 \geq \frac{\sigma(\mathrm{Rup}(\Phi(f)))}{d\sqrt{2d^2 - d}}.$$

Looking at (6.2) it is easy to see that:

$$\|\Phi(f) - \Phi(\tilde{f})\|_2 = \|\Phi(f - \tilde{f})\|_2 \leq d\,\|(f - \tilde{f}) - (f - \tilde{f} \bmod x)\|_2 = d\,\|f - \tilde{f}\|_2.$$

Thus

$$\|f - \tilde{f}\|_2 \geq \frac{\sigma(\mathrm{Rup}(\Phi(f)))}{d^2\sqrt{2d^2 - d}}.$$

□

Now we have the ability to compute a radius of indecomposability about any indecomposable polynomial and, as with irreducibility, this gives a simple algorithm to test the indecomposability of an approximate polynomial when a tolerance on the coefficients is specified.

Note that, as with factorization, if we omit the degree bound, it is possible to find a decomposable polynomial which is arbitrarily close to f, for example $f \circ (\epsilon x^2 + x)$. It may be that the degree bound in Corollary 6.4) is too tight; approximate decompositions up to degree $2 \deg f - 1$ may still be meaningful. However, we will consider only approximate decompositions of the same degree or smaller.

Example 6.1. Let us begin with a decomposable polynomial with a large noise term added to it (making it indecomposable):

$$f = (4\,x^2 + 3\,x - 1) \circ (x^2 - x + 1) + .02\,x^3 = 4\,x^4 - 7.98\,x^3 + 15\,x^2 - 11\,x + 6.$$

Then we compute

$$\Phi(f) = -11 + 15\,y - 7.98\,y^2 + 4\,y^3 + (15 - 7.98\,y + 4\,y^2)\,x + (-7.98 + 4\,y)\,x^2 + 4\,x^3$$

and the matrix $R(\Phi(f))$ which is 27×20. Computing the largest coefficient of $\|R(\Phi)\|^2$ we get 200 (compared to the bound $d^2\,(2d^2 - d) = 7168$ from Corollary 6.4). Computing the smallest singular value of $\mathrm{Rup}(\Phi(f))$, we find a lower bound on the distance from f

to the nearest polynomial which decomposes (in the 2-norm) is 0.0001914703496. Thus, any polynomial with constant coefficient 6 which is closer than the bound must also be decomposable. □

6.3 Approximate Decomposition

In the same way that approximate irreducibility testing can be used for approximate decomposability testing we can build an algorithm to compute approximate decompositions of polynomials.

For most polynomials that are nearly decomposable it is probably best to use the fast algorithms described in Corless *et al.* (1999). However, those algorithms are not guaranteed to converge. In the case that they do not, we can use a straightforward application of the factoring algorithm in Chapter 5 to $\Phi(f)$ creates an approximate version of Algorithm 6.2.

Algorithm 6.3.

INPUT: An indecomposable polynomial $f \in \mathbb{Q}[x]$.

OUTPUT: $g, h \in \mathbb{C}$ such that $f \approx g \circ h$.

1. Form $\Phi(f) = (f(x) - f(y))/(x - y)$;

2. Compute an approximate decomposition of $\Phi(f)$ over $\mathbb{C}[x, y]$; discard all factors p such that $\operatorname{tdeg} p + 1$ does not divide $\deg f$ – if no factors remain, skip to step 5;

3. For each remaining factor compute its distance to a factor of the form $\Phi(h)$:

   (a) For j from 0 to tdeg ϕ_i, compute the standard deviation of the coefficients of the terms of total degree j (a measure of how far those coefficients are from being equal);

   (b) Choose the maximum of the deviations over all values of j – this is the distance;

4. Choose the factor ϕ with the smallest distance and form $h = \sum_{i=1}^{d} c_i x^i$ where c_i is the average of the coefficients of the terms of ϕ with total degree $i-1$; use h to compute a least squares solution g to the system (6.1);

5. Find a and b as in Algorithm 6.2 step 4a, and compute a corresponding gs by least squares for possible choice of q

6. return the g, h pair with the smallest value of $\|f - g \circ h\|_2$

In practice, if f is a perturbed decomposable polynomial then the approximate factors of $\Phi(f)$ tend to contain polynomials very close to the form $\Phi(h)$. However, there is no guarantee on how close the approximate factors will come to having this form. For a slower algorithm, one can compute a decomposition for all the factors instead of the "best" on in Step 3 and use the one which gives an approximate decomposition closest to the the original polynomial.

If one finds that an approximate factorization of $\Phi(f)$ does not have any factors of the right degree, it is possible to modify the approximate GCD algorithm used in the approximate factorization to produce factors of a predetermined degree (by choosing what the numerical rank of the Sylvester matrix should be, rather than trying to compute what

it should be). In this way one can always find an approximate decomposition without a Dickson polynomial as a right decomposition factor though the backward error may be quite bad in some cases.

It is also straightforward to improve the computed approximate decomposition with a Gauss-Newton iteration in the same way approximate GCD and factor computations were computed in the previous chapters. Doing so should lead to modest improvements in backward error, much like the improvements seen for factorization. The iteration from Corless *et al.* (1999) could also be used to improve a decomposition computed from our algorithm.

In the following examples, step 5 of the algorithm is omitted.

Example 6.2. Beginning with the same polynomial as in Example 6.1

$$f = (4\,x^2 + 3\,x - 1) \circ (x^2 - x + 1) + .02\,x^3,$$

we feed $\Phi(f)$ into the approximate factorization algorithm (with iterative improvement) and get the following factorization:

$$\begin{aligned}\Phi(f) \approx (3.328611061 - 3.334456909\,x - 3.334456909\,y)\,(-3.304470928 + 1.196214923\,x \\ + 1.196214923\,y - 1.199266070\,x^2 - 0.0004743182219\,xy - 1.199266070\,y^2)\end{aligned}$$

The first factor is closer to the form $\Phi(h)$ than the second, and so monic best fit h is

$$h(x) = -0.998246836550386\,x + x^2$$

which we then make monic to give a neater decomposition. Using least squares to solve for the best corresponding g, we get

$$g = 6 + 11.0166611850816842\,x + 4\,x^2$$

Composing, we get

$$g(h(x)) = 4\,x^4 - 7.985974692\,x^3 + 15.00264817\,x^2 - 10.99734718\,x + 6$$

which is relative distance 0.0003282568052744 from the original f. □

As with factoring, if the smallest singular value of $\mathrm{Rup}(\Phi(f))$ is quite large then it may be trivial to find a closer decomposition than the one produced by the algorithm. Most trivial approximate decompositions have relative backward error of about 1. For example, setting the coefficients of all the odd power terms to 0 will given a polynomial which has a right composition factor of x^2. For a randomly generated polynomial, not close to one that decomposes, this may be the best we can do.

Example 6.3. Begin with a random polynomial

$$f = x^6 - 0.640\,x^5 - 1.62\,x^4 + 0.520\,x^3 + 0.800\,x^2 + 0.740\,x - 1.80.$$

In this case, we can easily construct

$$\tilde{f} = (-1.80 + 0.800\,x - 1.62\,x^2 + x^3) \circ x^2.$$

which has a backwards error of 0.37494846.

Approximate factorization gives a factorization consisting of one factor of degree 1 and one of degree 4 and having a relative backwards error of 0.16416249664. The degree 1 factor is closer to having the form $\Phi(h)$ and the degree 4 factor would lead to a decomposition of higher degree than f so we use the degree 1 factor:

$$f_2 = -0.0608141094394855 + 0.999999999958777\,y + x$$

to compute the monic approximate decomposition factor

$$h = -0.0608141094407389\,x + x^2.$$

Using this h to find g gives an approximate decomposition

$$\tilde{f} = x^6 - 0.182442\,x^5 - 1.65840\,x^4 + 0.202833\,x^3 + 0.752193\,x^2 - 0.0461194\,x - 1.80$$

with a relative backward error of 0.326648519, only very slightly better than the trivial approximate decomposition given above. □

Chapter 7

Conclusion

We have demonstrated several algorithms using SVD-based methods that provide reasonable partial solutions for a number of problems in approximate algebra. In particular, we discussed approximate factorization, approximate irreducibility testing, approximate greatest common divisor computation, and approximate decomposition computations. We presented algorithms that use SVD techniques to find solutions to the soft approximate factorization and GCD problems. We also used the SVD to derive bounds on the distance to nearest polynomials that factor, and nearest non-relatively prime polynomial pairs that in turn lead to tests of approximate irreducibility and approximate relative primeness.

As mentioned in the introduction it is possible to apply these same ideas to find solutions to the soft approximation versions of any problem that can be formulated as a homogeneous linear system. One future avenue of work is to identify such problems

and implement SVD based algorithms for them. A few problems that may possibly be tackled this way are approximate division and GCRD of Ore polynomials, GCRD of matrix polynomials, and functional decomposition of univariate polynomials.

The dimensions of the matrices for the multivariate factorization and GCD problems grow linearly with the number terms of polynomials, that means exponentially in the number of variables. In order to deal with very sparse problems with many variables we often need to project down to the dense two variable case (as was done with the Verschelde problem mentioned in Chapter 5). If the numerical errors are small the results in Giesbrecht *et al.* (2004) work quite well. But if errors much larger than machine precision appear, then this interpolation no longer works well. An optimization version of sparse interpolation is needed here.

It is clear from theory and experiments that SVD techniques cannot possibly solve the full optimization version of these approximate problems, since it ignores the structure in the linear systems derived from these algebraic problems. In fact, SVD based approximate factorization does not even find a factorization in a basin of attraction of factorization that achieves the global minimum. That is, iterative refinement (by Gauss-Newton) will still not lead to the global minimum, in general.

Solving the full optimization version of one of these problems is actually an instance of the structured total least squares (STLS) problem (for the given matrix structure in question). There is currently no practical way to solve STLS in general, though solutions have been sought in many disciplines. There are, however, heuristics that can find ap-

proximations of STLS solutions better than those found using regular SVD techniques. As presented in Botting (2004) an implementation of a heuristic to compute Riemannian SVD can find closer approximate solutions to the structured total least squares (STLS) problems arising in approximate algebra. The implementation presented in Botting (2004) is quite a bit slower than any of the SVD-based methods presented herein, but it is also completely general. It can handle any type of linear matrix structure (that is, optimization over a linear subspace of $\mathbb{C}^{n\times n}$). Very promising avenues of future work are using the approximate solutions to the corresponding STLS problems to compute approximate factorizations and approximate decompositions and optimizing those types of heuristics to specific matrix structures in order get implementations with speeds comparable to those of SVD-based algorithms.

References

D. R. Barton and R. Zippel (1976). A polynomial decomposition algorithm. In *SYMSAC '76: Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 356–358. ACM Press, New York, NY, USA.

D. R. Barton and R. Zippel (1985). Polynomial Decomposition Algorithms. *Journal of Symbolic Computation*, **1**: 159–168.

B. Beckermann and G. Labahn (1998a). A fast and numerically stable Euclidean-like algorithm for detecting relative prime numerical polynomials. *Journal of Symbolic Computation*, **26**(6): 691–714. Special issue on Symbolic Numeric Algebra for Polynomials S. M. Watt and H. J. Stetter, editors.

B. Beckermann and G. Labahn (1998b). When are two numerical polynomials relatively prime? *Journal of Symbolic Computation*, **26**(6): 691–714. Special issue on Symbolic Numeric Algebra for Polynomials S. M. Watt and H. J. Stetter, editors.

B. Botting (2004). *Structured Total Least Squares for Approximate Polynomial Operations*. Master's thesis, School of Computer Science, University of Waterloo.

S. L. Campbell and C. D. Meyer, Jr. (1979). *Generalized Inverses of Linear Transformations*. Pitman Publ. Ltd., London.

M. T. Chu, R. E. Funderlic, and R. J. Plemmons (2003). Structured low rank approximation. *Linear Algebra and Applications*, **366**: 157–172.

R. M. Corless, A. Galligo, I. S. Kotsireas, and S. M. Watt (2002). A geometric-numeric algorithm for absolute factorization of multivariate polynomials. In Mora (2002), pages 37–45.

R. M. Corless, P. M. Gianni, B. M. Trager, and S. M. Watt (1995). The singular value decomposition for polynomial systems. In A. H. M. Levelt (editor), *Proc. 1995 Internat. Symp. Symbolic Algebraic Comput. ISSAC'95*, pages 96–103. ACM Press, New York, N. Y.

R. M. Corless, M. Giesbrecht, D. Jeffrey, and S. M. Watt (1999). Approximate Polynomial Decomposition. In Samuel S. Dooley (editor), *ACM International*

Symposium on Symbolic and Algebraic Computation, pages 213–220. ACM, Vancouver, Canada.

R. M. Corless, M. W. Giesbrecht, M. van Hoeij, Ilias S. Kotsireas, and S. M. Watt (2001). Towards Factoring Bivariate Approximate Polynomials. In Mourrain (2001), pages 85–92.

R. M. Corless, S. M. Watt, and L. Zhi (2004). QR Factoring to Compute the GCD of Univariate Approximate Polynomials. *IEEE Transactions on Signal Processing*, **52**(12): 3394–3402.

C. Eckart and G. Young (1936). The approximation of one matrix by another of lower rank. *Psychometrika*, **1**(3): 211–218.

I. Z. Emiris, A. Galligo, and H. Lombardi (1997). Certified Approximate Univariate GCDs. *J. Pure Applied Algebra*, **117 & 118**: 229–251. Special Issue on Algorithms for Algebra.

M. Fried (1970). On a conjecture of Schur. *Mich. Math. Journal*, **17**: 41–55.

M. D. Fried and R. E. MacRae (1969). On the invariance of chains of Fields. *Ill. J. Math.*, **13**: 165–171.

A. Galligo and D. Rupprecht (2001). Semi-numerical determination of irreducible branches of a reduced space curve. In Mourrain (2001), pages 137–142.

A. Galligo and S. Watt (1997). A numerical absolute primality test for bivariate polynomials. In W. Küchlin (editor), *ISSAC 97 Proc. 1997 Internat. Symp. Symbolic Algebraic Comput.*, pages 217–224. ACM Press, New York, N. Y.

S. Gao (2003). Factoring multivariate polynomials via partial differential equations. *Math. Comput.*, **72**(242): 801–822.

S. Gao, E. Kaltofen, J. P. May, Z. Yang, and L. Zhi (2004). Approximate factorization of multivariate polynomials via differential equations. In Gutierrez (2004), pages 167–174.

S. Gao and V. M. Rodrigues (2003). Irreducibility of polynomials modulo p via Newton polytopes. *J. Number Theory*, **101**: 32–47.

J. von zur Gathen (1990). Functional Decomposition of Polynomials: the tame case. *Journal of Symbolic Computation*, **9**: 281–299.

A. O. Gelfond (1960). *Transcendental and algebraic numbers. Translated from the 1st Russian ed. by Leo F. Boron.* Dover, New York.

M. Giesbrecht, G. Labahn, and Lee, W.-s. (2004). Symbolic-Numeric Sparse Interpolation of Multivariate Polynomials (Extended Abstract). In *Proc. Ninth Rhine Workshop on Computer Algebra (RWCA'04), University of Nijmegen, the Netherlands*, pages 127–139. Full paper under preparation.

G. H. Golub and C. F. Van Loan (1996). *Matrix Computations*. Johns Hopkins University Press, Baltimore, Maryland, third edition.

J. Gutierrez (Editor) (2004). *ISSAC 2004 Proc. 2004 Internat. Symp. Symbolic Algebraic Comput.* ACM Press, New York, N. Y.

M. A. Hitz, E. Kaltofen, and Lakshman Y. N. (1999). Efficient Algorithms for Computing the Nearest Polynomial With A Real Root and Related Problems. In S. Dooley (editor), *Proc. 1999 ACM International Symposium on Symbolic and Algebraic Computation (ISSAC'99)*, pages 205–212. ACM Press, New York, N. Y.

Y. Huang, W. Wu, H. J. Stetter, and L. Zhi (2000). Pseudofactors of Multivariate Polynomials. In C. Traverso (editor), *Internat. Symp. Symbolic Algebraic Comput. ISSAC 2000 Proc. 2000 Internat. Symp. Symbolic Algebraic Comput.*, pages 161–168. ACM Press, New York, N. Y.

C.-P. Jeannerod and G. Labahn (2002). The SNAP package for arithmetic with numeric polynomials. In A. M. Cohen, X.-S. Gao, and N. Takayama (editors), *Proc. First Internat. Congress Math. Software ICMS 2002, Beijing, China*, pages 61–71. World Scientific, Singapore.

W. Kahan (1966). Numerical Linear Algebra. *Canadian Math. Bull.*, **9**: 757–801.

E. Kaltofen (1985). Fast parallel absolute irreducibility testing. *Journal of Symbolic Computation*, **1**(1): 57–67. Misprint corrections: *J. Symbolic Comput.* vol. 9, p. 320 (1989).

E. Kaltofen (1992). Polynomial factorization 1987-1991. In I. Simon (editor), *Proc. LATIN '92*, volume 583 of *Lect. Notes Comput. Sci.*, pages 294–313. Springer Verlag, Heidelberg, Germany.

E. Kaltofen (1995). Effective Noether irreducibility forms and applications. *J. Comput. System Sci.*, **50**(2): 274–295.

E. Kaltofen (2000). Challenges of Symbolic Computation My Favorite Open Problems. *Journal of Symbolic Computation*, **29**(6): 891–919. With an additional open problem by R. M. Corless and D. J. Jeffrey.

E. Kaltofen and J. P. May (2003). On Approximate Irreducibility of Polynomials in Several Variables. In Sendra (2003), pages 161–168.

N. K. Karmarkar and Lakshman Y. N. (1998). On Approximate GCDs of Univariate Polynomials. *Journal of Symbolic Computation*, **26**(6): 653–666. Special issue on Symbolic Numeric Algebra for Polynomials S. M. Watt and H. J. Stetter, editors.

C. T. Kelley (1999). *Iterative Methods for Optimization*. Number 18 in Frontiers in Applied Mathematics. SIAM, Philadelphia.

D. Kozen and S. Landau (1989). Polynomial Decomposition Algorithms. *Journal of Symbolic Computation*, **7**: 445–456.

T. Y. Li and Zhonggang Zeng (2003). A rank-revealing method and its application. Manuscript available at `http://www.neiu.edu/~zzeng/papers.htm`.

T. Mora (Editor) (2002). *ISSAC 2002 Proc. 2002 Internat. Symp. Symbolic Algebraic Comput.* ACM Press, New York, N. Y.

B. Mourrain (Editor) (2001). *ISSAC 2001 Proc. 2001 Internat. Symp. Symbolic Algebraic Comput.* ACM Press, New York, N. Y.

K. Nagasaka (2002). Towards certified irreducibility testing of bivariate approximate polynomials. In Mora (2002), pages 192–199.

K. Nagasaka (2004). Towards More Accurate Separation Bounds of Empirical Polynomials. *SIGSAM Bulletin*, **38**(4): 119–129. Formally Reviewed Communication.

E. Noether (1922). Ein algebraisches Kriterium f ur absolute Irreduzibilit at. *Math. Ann.*, **85**: 26–33.

M.-A. Ochi, M. Noda, and T. Sasaki (1991). Approximate Greatest Common Divisors of Multivariate Polynomials and its Application to Ill-conditioned Systems of Algebraic Equations. *J. Inf. Proces*, **12**: 292–300.

S. M. Rump (2003). Structured Perturbations Part I: Normwise Distances. *SIAM Journal on Matrix Analysis and Applications*, **25**(1): 1–30.

W. M. Ruppert (1999). Reducibility of Polynomials *f(x,y)* Modulo *p*. *J. Number Theory*, **77**: 62–70.

D. Rupprecht (1999). An Algorithm for Computing Certified Approximate GCD of n Univariate Polynomials. *J. Pure Applied Algebra*, **139**: 255–284.

T. Sasaki (2001). Approximate multivariate polynomial factorization based on zero-sum relations. In Mourrain (2001), pages 284–291.

T. Sasaki, M., M. Kolář, and M. Sasaki (1991a). Approximate Factorization of Multivariate Polynomials and Absolute Irreducibility Testing. *Japan J. of Industrial and Applied Mathem.*, **8**(3): 357–375.

T. Sasaki, T. Saito, and T. Hilano (1992). Analysis of Approximate Factorization Algorithm I. *Japan J. of Industrial and Applied Mathem.*, **9**(3): 351–368.

T. Sasaki and M. Sasaki (1997). Polynomial Remainder Sequence and Approximate GCD. *ACM SIGSAM Bulletin*, **31**: 4–10.

T. Sasaki, M. Suzuki, M. Kolář, and M. Sasaki (1991b). Approximate Factorization of Multivariate Polynomials and Absolute Irreducibility Testing. *Japan J. of Industrial and Applied Mathem.*, **8**(3): 357–375.

W. M. Schmidt (1976). *Equations over finite fields. An elementary approach.* Number 536 in Springer Lect. Notes Math. Springer Verlag, New York, N.Y.

A. Schönhage (1985). Quasi-GCD computations. *J. Complexity*, **1**: 118–137.

J. R. Sendra (Editor) (2003). *ISSAC 2003 Proc. 2003 Internat. Symp. Symbolic Algebraic Comput.* ACM Press, New York, N. Y.

A. J. Sommese, J. Verschelde, and C. W. Wampler (2004). Numerical Factorization of Multivariate Complex Polynomials. *Theoretical Comput. Sci.*, **315**(2–3): 651–669. Special issue on Algebraic and Numerical Algorithms.

G. W. Stewart (1973a). Error and Perturbation Bounds for Subspaces Associated with Certain Eigenvalue Problems. *SIAM Review*, **15**(4): 727–764.

G. W. Stewart (1973b). *Introduction to Matrix Computations.* Academic Press, Inc., New York.

G. Turnwald (1995). On Schur's conjecture. *J. Austral. Math. Soc. Ser. A*, **58**(3): 312–357.

Z. Zeng (2003). A Method Computing Multiple Roots of Inexact Polynomials. In Sendra (2003), pages 266–272.

Z. Zeng (2004). The approximate GCD of inexact polynomials Part I: a univariate algorithm. Manuscript, 8 pages.

Z. Zeng and B. H. Dayton (2004). The approximate GCD of inexact polynomials Part II: a multivariate algorithm. In Gutierrez (2004).

L. Zhi (2003). Displacement Structure in Computing Approximate GCD of Univariate Polynomials. In Z. Li and W. Sit (editors), *Proc. Sixth Asian Symposium on Computer Mathematics (ASCM 2003)*, volume 10 of *Lecture Notes Series on Computing*, pages 288–298. World Scientific, Singapore.

L. Zhi, K. Li, and M.-T. Noda. (2001). Approximate GCD of Multivariate Polynomials using Hensel lifting. MMRC Preprint, 7 pages.

L. Zhi and M.-T. Noda (2000). Approximate GCD of Multivariate Polynomials. In X.-S. Gao and D. Wang (editors), *Proc. Fourth Asian Symposium on Computer Mathematrics (ASCM 2000)*, Lecture Notes Series on Computing, pages 9–18. World Scientific, Singapore.