Modular Methods for Solving Nonlinear Polynomial Systems

(Thesis format: Monograph)

by

Raqeeb Rasheed

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

Faculty of Graduate Studies
University of Western Ontario
London, Ontario, Canada

THE UNIVERSITY OF WESTERN ONTARIO
FACULTY OF GRADUATE STUDIES

CERTIFICATE OF EXAMINATION

Supervisor

Prof. Dr. Marc Moreno Maza

Supervisory Committee

Examiners

Prof. Robert Corless

Prof. Mark Daley

Prof. Robert Mercer

The thesis by
Raqeeb Rasheed
entitled:

Modular Methods for Solving Nonlinear Polynomial Systems

is accepted in partial fulfillment of the
requirements for the degree of
Master of Science

Date

Prof. Jamie Andrews
Chair of Examining Board

Abstract

Solving polynomial systems is a problem frequently encountered in many areas of mathematical sciences and engineering. In this thesis we discuss how well-known algebraic tools and techniques can be combined in order to design new efficient algorithms for solving systems of non-linear equations symbolically.

G. Collins (1971) invented a method to compute resultants by means of the *Chinese Remainder Algorithm* (CRA). In addition, M. van Hoeij and M. Monagan (2002) described a modular method for computing polynomial greatest common divisors over algebraic number fields, also via CRA. We observe that merging these two algorithms produces a modular method for solving bivariate polynomial systems. Then, we generalize this method for solving trivariate polynomial systems. We report on an implementation of this approach in the computer algebra system `Maple`. Our experimental results illustrate the efficiency of this new method.

Keywords: Symbolic Computation, Modular Method, Nonlinear Polynomial Systems, Resultant, GCD, Subresultant Theory, Triangular Set, Regular Chain.

Acknowledgments

I am grateful to my supervisor, Prof. Marc Moreno Maza for his excellent help, guidance, support with funding and encouragement throughout the research and writing of this thesis.

I am also grateful to all the members of the ORCCA Lab for their assistance and friendship throughout my studies. This work is dedicated to my parents and my wife.

Contents

List of Figures

Chapter 1

Introduction

Solving systems of linear or non-linear, algebraic or differential equations, is a fundamental problem in mathematical sciences and engineering, which is hard for both numerical and symbolic approaches. Symbolic solving provides powerful tools in scientific computing and is used in an increasing number of applications such as cryptology, robotics, geometric modeling, dynamical systems in biology, etc.

For systems of linear equations, symbolic methods can compete today with numerical ones in terms of running times [15]; moreover there are input systems for which numerical methods fail to provide accurate solutions while symbolic methods always do. For systems of non-linear equations, when both symbolic and numerical methods can be applied, the latter ones have the advantage of speed for most problems whereas the former ones have that of exactness.

The ultimate goal in this work is to develop and implement new symbolic algorithms for solving non-linear systems that could compete with numerical methods when the comparison makes sense, that is, for systems that have finitely many solutions, with exact input coefficients and with symbolic output of moderate size. Under these hypotheses, we anticipate that the successful methods of symbolic linear algebra could be extended to the non-linear case. In fact, a first step in this direction has already been made in [12] where solving polynomial systems with rational number coefficients is reduced to solving polynomial systems with coefficients modulo a prime number. Therefore, the algorithms discussed in this thesis are meant to help with this latter case. As in [12] we restrict to systems with finitely many solutions.

We aim at using specialization and lifting techniques for speeding up computations, see [17] for a comprehensive discussion of these techniques. In our case, this leads us to interpolate multivariate polynomials and reconstruct multivariate rational functions. These operations are still research questions, for both symbolic and numerical methods. Hence, we know that every progress there, will probably benefit our work.

We also rely on the software tools developed at the Ontario Research Center for Computer Algebra (ORCCA) such as the `RegularChains` library [23] in `Maple`. In the near future, we aim at integrating in our `Maple` packages the fast algorithms and the high performance C code developed at ORCCA [16, 26, 25] too. One driving idea in the design of our algorithm is the ability to take advantage of highly efficient low-level routines such as FFT-based univariate and multivariate polynomial arithmetic.

Other driving ideas are *modular methods*, *recycling intermediate computations* and *genericity assumptions*. Modular methods are well-developed techniques since the early days of symbolic computations. A celebrated example is the modular computation of the determinant of an integer matrix, sketched in Section 2.9. Recycling intermediate computations is another way to say avoiding unnecessary computations, which is also a major issue in symbolic computations. Lastly, by genericity assumptions, we mean that our algorithm should be able to take advantage of the shape of the targeted solution set.

Among the works which have inspired this thesis are the modular algorithms of Collins [6, 7, 8, 9] van Hoeij with Monagan [19], Kaltofen and Monagan [20], Schost [29], Boulier, Moreno Maza and Oancea [4] Dahan, Jin, Moreno Maza and Schost [11].

Let us sketch the ideas developed in this thesis on a bivariate system of two non-linear equations

$$\begin{cases} f_1(X_1, X_2) &= 0 \\ f_2(X_1, X_2) &= 0 \end{cases} \tag{1.1}$$

We assume that the solution set of this input system can be given by a single system with a triangular shape

$$\begin{cases} t_1(X_1) &= 0 \\ t_2(X_1, X_2) &= 0 \end{cases} \tag{1.2}$$

We can choose for t_1 the square-free part of the resultant of f_1 and f_2 w.r.t. X_2, and, we choose for t_2 the GCD of f_1 and f_2 w.r.t. t_1.

The first key observation is that one can deduce t_2 from the intermediate computations of t_1. This is mainly due to the fact that a single triangular set is sufficient to describe the solution set of this input system. Full details with proofs are given in Chapter 3. The second key observation is that t_1, and thus t_2 can be computed by a modular algorithm, for instance the one of Collins [6, 7, 8, 9]. Thus, we have replaced the computation of t_2 (which was a priori a polynomial GCD over a number field) by univariate operations over the base field. Therefore, we have replaced a non-trivial operation by a much simpler one, which, in addition, can use fast arithmetic, such as FFT-based univariate multiplication.

When moving to the case of three variables, we manage to manipulate some intermediate polynomials by modular images without reconstructing them on their monomial basis. Of course, some technical difficulties need to be resolved such as the problem of bad specializations. Let us describe this problem in broad terms. It follows from resultant theory [18] that specializations of X_1, to successive values $v_0, v_1, \ldots$, in the input system can be used to compute t_1 by interpolation. However, not all specializations can be used for computing the GCD t_2. Let us consider for instance

$$\begin{cases} f_1(X_1, X_2) &= (X_2 + X_1)(X_2 + X_1 + 2) \\ f_2(X_1, X_2) &= (X_2 + X_1)(X_2 + 2). \end{cases} \tag{1.3}$$

Observe that $\gcd(f_1, f_2) = X_2 + X_1$ holds. Moreover, for all $v \neq 0$, we have

$$\gcd(f_1(X_1 = v, X_2), f_2(X_1 = v, X_2) = X_2 + v$$

However, for $v = 0$, we have

$$\gcd(f_1(X_1 = v, X_2), f_2(X_1 = v, X_2) = X_2(X_2 + 2)$$

Hence, the degree of $\gcd(f_1(X_1 = v, X_2), f_2(X_1 = v, X_2)$ depends on v. Therefore, we cannot construct $\gcd(f_1, f_2)$ from any $\gcd(f_1(X_1 = v, X_2), f_2(X_1 = v, X_2))$.

A second series of obstacles depend on the kind of variant of the Euclidean algorithm which is used to compute the images t_1 and t_2 . If we use the standard Euclidean algorithm, the bound on the number of specializations needed for t_2 can be essentially twice the bound on the number of specializations needed for t_1 If we use the subresultant algorithm, these two numbers can be the same.

In chapter 3 we describe modular algorithms for bivariate polynomial systems and give detailed experimental results. In chapter 4 we extend our work to trivariate systems. This adaptation is not straightforward and additional tricks, such as lifting techniques, are needed. Moreover, identifying the appropriate genericity conditions is much harder than in the bivariate case. However, these conditions can be checked easily during the solving process. We anticipate that these algorithms could be integrated into a general solver (not relying on any genericity conditions) and provide a substantial speed-up to it.

We have realized a preliminary implementation in `Maple`. To evaluate the quality of our algorithms, their implementation is parametrized by an implementation of multivariate polynomials; this can be the default `Maple` polynomial arithmetic based on DAGs or multivariate polynomials provided by `Maple` libraries such as `modp1/modp2` or `Recden`. We also have implemented a verifier to check and compare our results with `Maple`'s and `RegularChains`' built-in equivalent functions.

Our experimental results show that these new modular methods for solving bivariate and trivariate polynomial systems outperform solvers with similar specialization, such as the `Triangularize` command of the `RegularChains` library in `Maple`. In Chapter 5, we sketch what could be the adaptation of these modular methods to n-variate polynomial systems.

Chapter 2

Background

The aim of this chapter is to provide a background review of the basic notions and techniques used in the remaining chapters. Computing polynomial resultants and GCDs is the core operation in this thesis and most all sections of this chapter are dedicated to this topic, directly or indirectly.

Sections 2.1, 2.2 and 2.3 are devoted to the celebrated *Euclidean Algorithm* for computing polynomial resultants and GCDs.

Sections 2.4, 2.5 2.6 present the *Subresultant PRS Algorithm* for computing polynomial resultants and GCDs. This latter has very important properties that are described in Sections 2.7, 2.8.

Finally, Sections 2.9, 2.10, 2.11, 2.12, 2.13, 2.14 and 2.15 present techniques either for performing the Subresultant PRS Algorithm in some efficient manner or for applying it to more general contexts.

We would like to stress the fact that Remark 8 is essential to Chapter 3 of this thesis. Note also that Specifications 1, 2, 3, 4 define operations that used in the algorithms of the remaining chapters.

2.1 Univariate polynomials

In this thesis, $\mathbb{A}$ is always a commutative ring with unity and $\mathbb{K}$ is always a field. Sometimes $\mathbb{A}$ has additional properties.

Definition 1 *A polynomial $f \in \mathbb{A}[X]$ is squarefree if it is not divisible by the square of any non-constant polynomial.*

Definition 2 *A field $\mathbb{K}$ is algebraically closed if every polynomial in $\mathbb{K}[X]$ has a root in $\mathbb{K}$; or equivalently: every polynomial in $\mathbb{K}[X]$ is a product of linear factors. The smallest algebraically closed field containing $\mathbb{K}$ is called algebraic closure of $\mathbb{K}$.*

Definition 3 *We say $\mathbb{K}$ is perfect field if for any algebraic extension field $\mathbb{L}$ of $\mathbb{K}$ we have: for all $f \in \mathbb{K}[X]$, if f is squarfree in $\mathbb{K}[X]$ then f is squarfree in $\mathbb{L}[X]$.*

Proposition 1 *Let f_1, f_2 be two polynomials in $\mathbb{K}[X]$ such that f_2 is a non-constant polynomial whose leading coefficient is a unit. Then, there exists a unique couple (q, r) of polynomials in $\mathbb{K}[X]$ such that*

$$f_1 = qf_2 + r \text{ and } (r = 0 \text{ or } \deg(r) < \deg(f_2)). \tag{2.1}$$

The polynomials q and r are called the quotient *and the* remainder *in the* division with remainder *(or simply* division*) of f_1 by f_2. Moreover, the couple (q, r) is computed by the following algorithm:*

Algorithm 1

Input: *univariate polynomials* $f_1 = \Sigma_{i=0}^{n} a_i X^i$ *and* $f_2 = \Sigma_{i=0}^{m} b_i X^i$ *in* $\mathbb{A}[X]$ *with respective degrees* n *and* m *such that* b_m *is a unit.*

Output: *the quotient* q *and the remainder* r *of* f_1 *w.r.t.* f_2.

$\mathrm{divide}(f_1, f_2) ==$

  $n < m \Rightarrow$ **return** $(0, f_1)$

  $r := f_1$

  for $i = n - m, n - m - 1, \ldots, 0$ **repeat**

    if $\deg r = m + i$ **then**

      $q_i := \mathrm{lc}(r) / b_m$

      $r := r - q_i X^i f_2$

    else $q_i := 0$

  $q := \Sigma_{j=0}^{n-m} q_j X^j$

  return (q, r)

Definition 4 *Let* $f \in \mathbb{A}[X]$ *with* $\mathbb{A}$ *a UFD (Unique Factorization Domain), we say that* f *is primitive if a GCD of its coefficients is a unit in* $\mathbb{A}$.

Definition 5 *Let* $\mathbb{A}$ *be a UFD, we say that* $f_1, f_2 \in \mathbb{A}[X]$ *are similar if there exist* $c_1, c_2 \in \mathbb{A}$ *such that*

$$c_1 f_1 = c_2 f_2$$

2.2 The Euclidean Algorithm

Definition 6 *An integral domain* $\mathbb{A}$ *endowed with a function* $d : \mathbb{A} \longmapsto \mathbb{N} \cup \{-\infty\}$ *is a* Euclidean domain *if the following two conditions hold*

- *for all* $f_1, f_2 \in \mathbb{A}$ *with* $f_1 \neq 0$ *and* $f_2 \neq 0$ *we have* $d(f_1\, f_2) \geq d(f_1)$*,*

- *for all* $f_1, f_2 \in \mathbb{A}$ *with* $f_2 \neq 0$ *there exist* $q, r \in \mathbb{A}$ *such that*

$$f_1 \; = q\, f_2 + r \quad \text{and} \quad d(r) < d(f_2). \tag{2.2}$$

The elements q *and* r *are called the* quotient *and the* remainder *of* f_1 *w.r.t.* f_2 *(although* q *and* r *may not be unique). The function* d *is called the* Euclidean size.

Example 1 *Let* $\mathbb{A} = \mathbb{K}[X]$ *where* $\mathbb{K}$ *is a field with* $d(f_1) = \deg(f_1)$ *the degree of* f_1 *for* $f_1 \in \mathbb{A}, f_1 \neq 0$ *and* $d(0) = -\infty$*. Uniqueness of the quotient and the remainder is given by Proposition 1. They are denoted respectively* $\mathrm{quo}(f_1, f_2)$ *and* $\mathrm{rem}(f_1, f_2)$*.*

Definition 7 *The GCD of any two polynomials* f_1, f_2 *in* $\mathbb{K}[X]$ *is the polynomial g in* $\mathbb{K}[X]$ *of greatest degree which divides both* f_1 *and* f_2*. We denote the GCD of two polynomials* f_1, f_2 *by* $\gcd(f_1, f_2)$*. Clearly* $\gcd(f_1, f_2)$ *is uniquely defined up to multiplication by a non-zero scalar.*

Proposition 2 *For the Euclidean domain* $\mathbb{A}$*, and all* $f_1, f_2 \in \mathbb{A}$ *Algorithm 2 computes a GCD of* f_1 *and* f_2*. This means that the following properties hold*

(i) g *divides* f_1 *and* f_2*, that is, there exist* ${f_1}', {f_2}' \in \mathbb{A}$ *such that* $f_1 = {f_1}' g$ *and* $f_2 = {f_2}' g$*,*

(ii) *there exist* $u, v \in \mathbb{A}$ *such that* $u f_1 + v f_2 = g$ *holds.*

Algorithm 2

Input: $f_1, f_2 \in \mathbb{A}$.

Output: $g \in \mathbb{A}$ *a GCD of* f_1 *and* f_2.

$r_0 := f_1$

$r_1 := f_2$

$i := 2$

while $r_{i-1} \neq 0$ **repeat**

  $r_i := r_{i-2} \operatorname{rem} r_{i-1}$

  $i := i + 1$

return r_{i-2}

Remark 1 *Algorithm 2 is known as the* Euclidean Algorithm. *This algorithm can be modified in order to compute* $u, v \in \mathbb{A}$ *such that* $uf_1 + f_2v = g$. *This enhanced version, Algorithm 3 below, is called the* Extended Euclidean Algorithm *(EEA). Proposition 3 gives an important application of the EEA.*

Algorithm 3

Input: $f_1, f_2 \in \mathbb{A}$.

Output: $g \in \mathbb{A}$ *a GCD of* f_1 *and* f_2 *together with* $s, t \in \mathbb{A}$ *such that* $g = s\, f_1 + t\, f_2$.

$r_0 := f_1$; $s_0 := 1$; $t_0 := 0$

$r_1 := f_2$; $s_1 := 0$; $t_1 := 1$

$i := 2$

while $r_{i-1} \neq 0$ **repeat**

  $q_i := r_{i-2} \operatorname{quo} r_{i-1}$

  $r_i := r_{i-2} \operatorname{rem} r_{i-1}$

  $s_i := s_{i-2} - q_i\, s_{i-1}$

  $t_i := t_{i-2} - q_i\, t_{i-1}$

  $i := i + 1$

return$(r_{i-2}, s_{i-2}, t_{i-2})$

Proposition 3 *Let* $\mathbb{A}$ *be an Euclidean domain and let* f_1, m *be in* $\mathbb{A}$*. Then* $f_1 \bmod m$ *is a unit of* $\mathbb{A}/m$ *iff* $\gcd(f_1, m) = 1$*. In this case the Extended Euclidean Algorithm can be used to compute the inverse of* $f_1 \bmod m$*.*

2.3 Resultant of univariate polynomials

Let $f_1, f_2 \in \mathbb{A}[X]$ be two non-zero polynomials of respective degrees m and n such that $n + m > 0$. Suppose

$$f_1 = a_m X^m + a_{m-1} X^{m-1} + \cdots + a_1 X + a_0 \ \text{ and } \ f_2 = b_n X^n + b_{n-1} X^{n-1} + \cdots + b_1 X + b_0$$

Definition 8 *The Sylvester matrix of* f_1 *and* f_2 *is the square matrix of order* $n+m$ *with coefficients in* $\mathbb{A}$*, denoted by* $\mathrm{sylv}(f_1,f_2)$ *and defined by*

$$\begin{pmatrix} a_m & 0 & \cdots & 0 & b_n & 0 & \cdots & 0 \\ a_{m-1} & a_m & \ddots & \vdots & b_{n-1} & b_n & \ddots & \vdots \\ a_{m-2} & a_{m-1} & \ddots & 0 & b_{n-2} & b_{n-1} & \ddots & 0 \\ \vdots & & \ddots & a_m & \vdots & & \ddots & b_n \\ & \vdots & & a_{m-1} & & \vdots & & b_{n-1} \\ a_0 & & & & b_0 & & & \\ 0 & a_0 & & \vdots & 0 & b_0 & & \vdots \\ \vdots & \ddots & \ddots & & \vdots & \ddots & \ddots & \\ 0 & \cdots & 0 & a_0 & 0 & \cdots & 0 & b_0 \end{pmatrix}$$

Its determinant is denoted by $\mathsf{res}(f_1, f_2)$ *and called the resultant of* f_1 *and* f_2*.*

When $\mathbb{A}$ is a field $\mathbb{K}$, the Euclidean Algorithm can be enhanced as follows in order to compute $\gcd(f_1, f_2)$ when $\mathsf{res}(f_1, f_2) = 0$, or $\mathsf{res}(f_1, f_2)$ otherwise.

Algorithm 4

Input: $f_1, f_2 \in \mathbb{K}[X]$ *with* $f_1 \neq 0$*,* $f_2 \neq 0$ *and* $\deg(f_1) + \deg(f_2) > 0$.

Output: *if* $\mathsf{res}(f_1, f_2) = 0$ *then monic* $\gcd(f_1, f_2)$ *else* $\mathsf{res}(f_1, f_2)$.

$m := \deg(f_1)$

$n := \deg(f_2)$

if $m < n$ **then**

  $r := (-1)^{nm}$

  $(f_1, f_2, m, n) := (f_2, f_1, n, m)$

else

  $r := 1$

repeat

  $b_n := \mathsf{lc}(f_2)$

  if $n = 0$ ***then return*** $r\, b_n^m$

  $h := f_1 \operatorname{rem} f_2$

  if $h = 0$ ***then return*** $(1/b_n) f_2$

  $p := \deg(h)$

  $r := r\,(-1)^{nm} b_n^{m-p}$

  $(f_1, f_2, m, n) := (f_2, h, n, p)$

Algorithm 4 applies in particular when trying to compute the GCD of two polynomials in $(\mathbb{K}[X_1]/\langle R_1\rangle)[X_2]$ where R_1 is an irreducible polynomial of $\mathbb{K}[X_1]$. Indeed, in this case the residue class ring $f_2[X_1]/\langle R_1\rangle$ is a field, for instance with $R_1 = X_1^2 + 1$. When R_1 is a square-free polynomial of $\mathbb{K}[X_1]$ (but not necessarily an irreducible polynomial) then Algorithm 4 can be adapted using the *D5 Principle* [14]. This will be explained in Remark 8. The resultant of f_1 and f_2 has the following fundamental property.

Proposition 4 *Assume that* $\mathbb{A}$ *is a unique factorization domain (UFD). Then, the polynomials* f_1, f_2 *have a common factor of positive degree if and only if* $\mathsf{res}(f_1, f_2) = 0$ *holds.*

2.4 Pseudo-division of univariate polynomials

The Euclidean Algorithm has several drawbacks. In particular, it suffers from intermediate expression swell. The *Subresultant PRS Algorithm* (Algorithm 6) described in Section 2.5 provides a way to better control intermediate expression sizes. One step toward this algorithm is the notion of *pseudo-division*, which allows one to *emulate* polynomial division over rings that are not necessarily fields.

Proposition 5 *Let* $f_1, f_2 \in \mathbb{A}[X]$ *be univariate polynomials such that* f_2 *has a positive degree w.r.t.* X *and the leading coefficient of* f_2 *is not a zero-divisor. We define*

$$e = \min(0, \deg(f_1) - \deg(f_2) + 1)$$

Then there exists a unique couple (q, r) *of polynomials in* $\mathbb{A}[x]$ *such that we have:*

$$(\mathrm{lc}(f_2)^e f_1 = q f_2 + r) \ and \ (r = 0 \ \text{ or } \ \deg(r) < \deg(f_2)). \tag{2.3}$$

The polynomial q *(resp.* r*) is called the* the pseudo-quotient *(*the pseudo-remainder*) of* f_1 *by* f_2 *and denoted by* pquo(a, b) *(*prem(a, b)*). The map* $(f_1, f_2) \longmapsto (q, r)$ *is called the* pseudo-division *of* f_1 *by* f_2*. In addition, Algorithm 5 computes this couple.*

Algorithm 5

Input: $f_1, f_2 \in \mathbb{A}[X]$ *with* $f_2 \notin \mathbb{A}$.

Output: $q, r \in \mathbb{A}[X]$ *satisfying Relation (2.3) with* $e = \min(0, \deg(f_1) - \deg(f_2) + 1)$.

prem$(f_1, f_2) ==$

r := f_1

q := 0

$e := \max(0, \deg(f_1) - \deg(f_2) + 1)$

while $r \neq 0$ **or** $\deg(r) \geq \deg(f_2)$ **repeat**

d := $\deg(r) - \deg(f_2)$

t := lc$(r) y^d$

q := lc$(f_2) q + t$

r := lc$(f_2) r - t f_2$

e := $e - 1$

r := lc$(f_2)^e r$

q := lc$(f_2)^e q$

return (q, r)

2.5 The Subresultant PRS Algorithm

We now review *Collins's PRS Algorithm*, also called the *Subresultant PRS Algorithm* of Brown and Collins [7, 5]. On input $f_1, f_2 \in \mathbb{A}[X]$ Algorithm 6 produces a sequence $(f_1, f_2, \ldots, f_k)$ of polynomials in $\mathbb{A}[X]$ defined by the following relations

$$f_{i+1} \; = \; \text{prem}(f_{i-1}, f_i)/\beta_i \;\; \text{for} \;\; (i = 1, \ldots, k-1) \qquad (2.4)$$

where $\beta_1, \beta_2, \ldots, \beta_{k-1}$ forms a sequence of elements of $\mathbb{A}$ such that the division shown in Equation (2.4) is exact. We define

$$\delta_i := \deg(f_i) - \deg(f_{i+1}), \text{ and } a_i := \mathrm{lc}(f_i). \tag{2.5}$$

Then, the sequence $\beta_1, \beta_2, \ldots, \beta_{k-1}$ is given by

$$\beta_{i+1} := \begin{cases} (-1)^{\delta_0+1} & \text{if } i = 0 \\ (-1)^{\delta_i+1}(\psi_i)^{\delta_i} a_i & \text{if i} = 1, \ldots, \mathrm{k} - 2 \end{cases} \tag{2.6}$$

where $(\psi_0, \ldots, \psi_{k-1})$ is an auxiliary sequence given by

$$\psi_0 := 1, \text{ and } \psi_{i+1} := \psi_i(a_{i+1}/\psi_i)^{\delta_i} = ((a_{i+1})^{\delta_i})/(\psi_i)^{\delta_i-1} \text{ for} i = 0, \ldots, k-2. \tag{2.7}$$

Algorithm 6 *Subresultant PRS Algorithm*

Input *:* $f_1, f_2 \in \mathbb{A}[X]$

Output*:* prs*: a list of polynomials in* $SubresultantPRS(f_1, f_2)$ *over* $\mathbb{A}[X]$.

$(P_1, P_2) := (f_1,\ f_2)$

$prs := [P_1, P_2]$

(m_1, m_2):=$(\deg(P_1, X)$,$\deg(P_2, X))$

$d_1 := m_1 - m_2$

$b = (-1)^{d_1+1}$

$P_3 := prem(P_1, P_2, X)/b$

$m_3 := \deg(P_3, X)$

$g_1 := -1$

while $P_3 \neq 0$ ***do***

add P_3 *into the list* prs

$d_2 := m_2 - m_3$

$a :=$ lc(P_2, X)

$g_2 := (-a)^{d_1} \,/\, g_1^{d_1-1}$

$b := -a\, g_2^{d_2}$

$(P_1, P_2, m_2, g_1, d_1)$:=$(P_2, P_3, m_3, g_2, d_2)$

$P_3 := prem(P_1, P_2, X)$

$m_3 := \deg(P_3, X)$

$P_3 := P_3/b$

return prs

Example 2 *Consider the two polynomials*

$$f_1 = X^8 + X^6 - 3\,X^4 - 3\,X^3 + 8\,X^2 + 2\,X - 5 \ \text{ and } \ f_2 = 3\,X^6 + 5\,X^4 - 4\,X^2 - 9\,X + 21.$$

originally used as an example by Knuth in [21] and also in [3]. The Euclidean Algorithm in following intermediate remainders

$$\mathrm{R}_2(X) = -\tfrac{5}{9}X^4 + \tfrac{1}{9}\,X^2 - \tfrac{1}{3}$$

$$\mathrm{R}_3(X) = -\tfrac{117}{25}\,X^2 - 9\,X + \tfrac{441}{25}$$

$$\mathrm{R}_4(X) = \tfrac{233150}{19773}\,X - \tfrac{102500}{6591}$$

$$\mathrm{R}_5(X) = -\tfrac{1288744821}{543589225}$$

The Subresultant PRS Algorithm performed in $\mathbb{Z}[X]$ *produces the following sequence of intermediate pseudo-remainders:*

$$\mathrm{P}_2(X) = 15\,X^4 - 3\,X^2 + 9$$
$$\mathrm{P}_3(X) = 65\,X^2 + 125\,X - 245$$
$$\mathrm{P}_4(X) = 9326\,X - 12300$$
$$\mathrm{P}_5(X) = 260708$$

It is easy to check that $\mathrm{R}_2, \mathrm{R}_3, \mathrm{R}_4, \mathrm{R}_5$ *are proportional to* $\mathrm{P}_2, \mathrm{P}_3, \mathrm{P}_4, \mathrm{P}_5$ *respectively. Therefore, we see that Subresultant PRS Algorithm provides a way to access to the polynomials computed by the Euclidean Algorithm (up to multiplicative factors) while controlling better the size of the coefficients.*

Let us stress the two following points

- The Euclidean Algorithm works over a field and hence uses rational arithmetic, something which one usually wants to avoid.

- The Subresultant PRS Algorithm uses only polynomial operations and has moderate coefficient growth. While the coefficient growth is not minimal it does have the advantage that the cost to reduce coefficient growth is minimal, namely a simple division by a known divisor, exactly the process followed in fraction free Gaussian elimination.

2.6 Subresultants

In this section, we review briefly the concept of *subresultants* and then state a few important theorems that are related to the algorithms presented in this thesis. We follow the presentation of Yap's book [32]. In particular, we interpret the intermediate polynomials computed by the Subresultant PRS Algorithm.

Definition 9 *Let M be a $k \times \ell$ matrix, $k \leq \ell$, over an integral domain $\mathbb{A}$. The* determinantal polynomial *of M is*

$$dpol(M) = \mid M_k \mid X^{\ell-k} + \cdots + \mid M_\ell \mid,$$

where M_i denotes the sub-matrix of M consisting of the first $k-1$ columns followed by the j^{th} column for $k \leq j \leq \ell$.

Definition 10 *Let $f_1 = \sum_{j=0}^{m} a_j X^j,\ f_2 = \sum_{j=0}^{n} b_j X^j \in \mathbb{A}[X]$ with $\deg(f_1) = m \geq n = \deg(f_2) \geq 0$. For $i = 0, 1, \cdots, n-1$, the $i-$th* subresultant *of f_1 and f_2 is defined as*

$$sres_i(f_1, f_2) = dpol(mat(X^{n-i-1}f_1, X^{n-i-2}f_1, \cdots, X^1 f_1, f_1, X^{m-i-1}f_2, X^{m-i-2}f_2, \cdots, f_2))$$

Observe that the defining matrix

$$\text{mat}(X^{n-i-1}f_1, X^{n-i-2}f_1, \cdots, X^1 f_1, f_1, X^{m-i-1}f_2, X^{m-i-2}f_2, \cdots, f_2)$$

has $m + n - 2i$ rows and $m + n - i$ columns (see Figure 2.1 for i-th subresultant of f_1, f_2). If $n = 0$, then $i = 0$ and f_1 does not appear in the matrix and the matrix is $m \times m$. The *nominal degree* of $sres_i(f_1, f_2)$ is i. Note that the zero-th subresultant is in fact the resultant,

$$sres_0(f_1, f_2) = \text{res}(f_1, f_2).$$

$$
dpol \begin{pmatrix}
\multicolumn{5}{c}{\overbrace{\hspace{8cm}}^{m+1}} & \multicolumn{4}{c}{\overbrace{\hspace{6cm}}^{n-i-1}} \\
a_m & a_{m-1} & \cdots & \cdots & a_0 & & & & \\
 & a_m & a_{m-1} & \cdots & \cdots & a_0 & & & \\
 & & \ddots & \cdots & \cdots & & \ddots & & \\
 & & & \ddots & \cdots & \cdots & & \ddots & \\
 & & & & a_m & a_{m-1} & \cdots & \cdots & a_0 \\
b_n & b_{n-1} & \cdots & b_0 & & & & & \\
 & b_n & b_{n-1} & \cdots & b_0 & & & & \\
 & & \ddots & \cdots & \cdots & \ddots & & & \\
 & & & \ddots & \cdots & \cdots & \ddots & & \\
 & & & & & & & & \\
 & & & & & b_n & b_{n-1} & \cdots & b_0
\end{pmatrix}
$$

Figure 2.1: i-th subresultant of f_1 and f_2

It is convenient to extend the above definitions to cover the cases $i = n+1, \cdots, m$:

$$
sres_i(f_1, f_2) := \begin{cases} 0 & \text{if } \; i = n+1, n+2, ..., m-2 \\ f_2 & \text{if } \; i = m-1 \\ f_1 & \text{if } \; i = m \end{cases}
$$

Definition 11 *The sequence* $(S_m, S_{m-1}, \cdots, S_1, S_0)$ *where* $S_i = sres_i(f_1, f_2)$ *is called the* subresultant chain *of* f_1 *and* f_2*. A member* $sres_i(f_1, f_2)$ *in the chain is regular if its degree is equal to the nominal degree* i*; otherwise it is irregular.*

Example 3 *The subresultant chain of* $f_1 = X_2^4 + X_1X_2 + 1$ *and* $f_2 = 4X_2^3 + X_1$ *over*

$(\mathbb{Q}[X_1])[X_2]$ *produces the following* sequence of polynomials*:*

$$S_4 = X_2^4 + X_1X_2 + 1$$
$$S_3 = 4X_2^3 + X_1$$
$$S_2 = -4(3X_1X_2 + 4)$$
$$S_1 = -12X_1(3X_1X_2 + 4)$$
$$S_0 = -27X_1^4 + 256$$

Definition 12 *We define a block to be a sequence*

$$B = (P_1, P_2, \cdots, P_k),\ k \geq 1. \tag{2.8}$$

of polynomials where $P_1 \sim P_k$ and $0 = P_2 = P_3 = \cdots = P_{k-1}$. We call P_1 and P_k (respectively) the top *and* base *of the block. Two special cases arise: In case $k = 1$, we call B a* regular block*; in case $P_1 = 0$, we call B a* zero block. *Thus the top and the base of a regular block coincide.*

Theorem 1 (*Block Structure Theorem) The subresultant chain $(S_m, S_{m-1}, ..., S_0)$ is uniquely partitioned into a sequence $B_0, B_1, ..., B_k, (k > 1)$ of blocks such that*

(i) B_0 is a regular block.

(ii) If U_i is the base polynomial of block B_i then U_i is regular and U_{i+1}~prem(U_{i-1} , U_i) ($0 < i < k$).

(iii) There is at most one zero block; if there is one, it must be B_k.

In the following we relate the *subresultant PRS algorithm sequence*, that is, the sequence of polynomials

$$(f_0, f_1, \ldots, f_k)$$

defined by Equation (2.4) in Section 2.5 to the subresultant chain

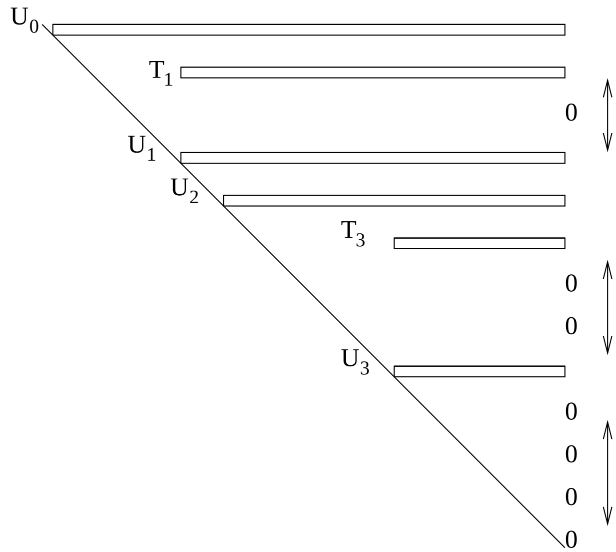

Figure 2.2: Block structure of a chain with $m = 12$.

$$(S_m, S_{m-1}, \ldots, S_0).$$

where $S_m = f_1$ and $S_{m-1} = f_2$. The basic connection, up to similarity, is established by the *Block Structure Theorem*. The real task is to determine the coefficients of similarity between the top of B_i and f_i. This is done in the following result, known as the *Subresultant PRS Correctness Theorem*.

Theorem 2 *Let T_i, U_i be the top and base polynomials of block B_i, where $(B_0, ..., B_k)$ are the non-zero blocks of our subresultant chain then the sequence $(T_0, ..., T_k)$ is precisely $(P_0, ..., P_k)$, computed by Algorithm 6.*

2.7 Specialization property of subresultants

Let $\mathbb{A}$ and $\mathbb{A}^*$ be commutative rings with identities, and $\Phi : \mathbb{A} \to \mathbb{A}^*$ be a ring homomorphism of $\mathbb{A}$ into $\mathbb{A}^*$. Note that Φ induces a ring homomorphism of $\mathbb{A}[X]$ into $\mathbb{A}^*[X]$, also denoted by Φ, as follows:

$$\Phi : \begin{array}{ccc} \mathbb{A}[X] & \to & \mathbb{A}^*[X] \\ a_m X^m + \cdots + a_0 & \mapsto & \Phi(a_m)\, X^m + \cdots + \Phi(a_0). \end{array}$$

Theorem 3 *Let $f_1, f_2 \in \mathbb{A}[X]$ of respective positive degrees m and n:*

$$\begin{array}{rcl} f_1(X) & = & a_m X^m + a_{m-1} X^{m-1} + \ldots + a_0 \\ f_2(X) & = & b_n X^n + b_{n-1} X^{n-1} + \ldots + b_0 \end{array}$$

Assume that $\deg(\Phi(f_1)) = m$ holds and define $k := \deg(\Phi(f_2))$, thus $0 \leq k \leq n$. Then, for all $0 \leq i < \max(m, k) - 1$, we have

$$\Phi(sres_i(f_1, f_2)) = \Phi(a_m)^{n-k} sres_i(\Phi(f_1), \Phi(f_2)) \tag{2.9}$$

Remark 2 *The combination of Theorem 3 and Theorem 1 is extremely useful for us and we give here a fundamental application. Let $T_1 \in \mathbb{K}[X_1]$ be a non-constant univariate polynomial and define $\mathbb{L} = \mathbb{K}[X_1]/\langle T_1 \rangle$. Let Φ be the natural homomorphism from $\mathbb{K}[X_1][X_2]$ to $\mathbb{L}[X_2]$ that reduces polynomials of $\mathbb{K}[X_1]$ modulo T_1.*

Theorem 3 tells us how to deduce the subresultant chain of $\Phi(f_1)$ and $\Phi(f_2)$ from that of f_1 and f_2. Assume that either $\Phi(\mathrm{lc}(f_1)) \neq 0$ or $\Phi(\mathrm{lc}(f_2)) \neq 0$ holds.

When $\mathbb{L}$ is a field, one can compute a GCD of $\Phi(f_1)$ and $\Phi(f_2)$ in $\mathbb{L}[y]$ as follows:

(1) *Consider all regular subresultants of f_1, f_2 by increasing index.*

(2) *Let j be the smallest index i such that $sres_i(f_1, f_2)$ is a regular subresultant whose leading coefficient is not mapped to zero by Φ.*

(3) *Then $\Phi(S_j)$ is a GCD of $\Phi(f_1)$ and $\Phi(f_2)$ in $\mathbb{L}[y]$.*

Indeed if for an index i, the subresultant $sres_i(f_1, f_2)$ is regular and its leading coefficient is mapped to zero by Φ, then in fact $\Phi(sres_i(f_1, f_2)) = 0$. This follows from the Block Structure Theorem *(Theorem 1).*

2.8 Subresultant degree bounds

Another important ingredient for the algorithms discussed in Chapters 3 and 4 is the fact that subresultants of polynomials f_1, f_2 in $\mathbb{K}[X_1][X_2]$ or $\mathbb{K}[X_1, X_2][X_3]$ are essentially determinants and thus can be computed by modular methods, see Section 2.9. In this section, we restrict for simplicity to the case of bivariate polynomials f_1, f_2 in $\mathbb{K}[X_1][X_2]$. The following result gathers the bounds that we use in Chapters 3. See [17] for details

Theorem 4 *Assume that f_1 and f_2 have respective X_2-degrees m and n, with $m \geq n > 0$. Let R_1 be $\mathsf{res}(f_1, f_2, X_2)$, that is the resultant of f_1 and f_2 w.r.t. X_2. Let $\mathrm{tdeg}(f_1)$ and $\mathrm{tdeg}(f_2)$ be the total degree of f_1 and f_2 respectively. Let S_d be the d-th subresultant of f_1 and f_2. Then we have:*

- $\deg(R_1) \leq \deg_{X_1}(f_1)\deg_{X_2}(f_2) + \deg_{X_2}(f_1)\deg_{X_1}(f_2)$,
- $\deg(R_1) \leq \mathrm{tdeg}(f_1)\mathrm{tdeg}(f_2)$,
- $\deg(S_d) \leq (m - d)(\mathrm{tdeg}(f_1) + \mathrm{tdeg}(f_2))$.

2.9 Modular methods

Modular methods in symbolic computations aim at providing two benefits: controlling the swell of intermediate expressions and offering opportunities to use fast arithmetic, such as FFT-based arithmetic.

A first typical example is the computation of polynomial GCDs in $\mathbb{Z}[X]$ via computations in $\mathbb{Z}/p\mathbb{Z}[X]$ for one or several prime numbers p. Computing with integers modulo a prime number p allows one to limit the size of the coefficients to p. It also permits the use of FFT-based multiplication in $\mathbb{Z}/p\mathbb{Z}[X]$.

A second example, which is closely related to our work, is the computation of the determinant of a matrix M with integer coefficients. One can compute this determinant using a direct method such as Gaussian elimination. Another approach is via the computation of this determinant modulo several prime numbers and then recombining these results by means of the Chinese Remaindering Algorithm. Let us review more precisely how this can be done.

Consider pairwise distinct prime numbers $p_1, \ldots, p_e$ such that their product m exceeds $2B$, where B is the Hadamard bound for the determinant of M. Let $\mathbb{Z}^{n\times n}$ and $\mathbb{Z}/p_i\mathbb{Z}^{n\times n}$ be the ring of square matrices over $\mathbb{Z}$ and $\mathbb{Z}/p_i\mathbb{Z}$ respectively. For all $1 \leq i \leq e$, let Φ_{p_i} be the reduction map from $\mathbb{Z}^{n\times n}$ to $(\mathbb{Z}/p_i\mathbb{Z})^{n\times n}$ that reduces all coefficients modulo p_i.

One can compute the determinant of M using the following strategy. For each $1 \leq i \leq e$, consider the determinant d_i of the modular image $\Phi_{p_i}(M)$ of M. Then, using the Chinese Remaindering Algorithm, one computes the integer d modulo m which is equal to d_i modulo p_i for all $1 \leq i \leq e$. Due to the fact that $m > 2B$ holds, the integer d modulo m is actually equal to the determinant of M. As shown in [17] this approach performs much better than the direct approach. Figure 2.9 sketches this modular computation.

$$\begin{array}{ccc} M \in \mathbb{Z}^{n\times n} & \xrightarrow{\text{For primes } p_0, p_1, \ldots, p_e} & \Phi_{p_i}(M) \in \mathbb{Z}^{n\times n}/p_i\mathbb{Z} \\ det \big\downarrow & & \big\downarrow det \\ |M| & \xleftarrow{\text{Chinese Reminder (CRA)}} & \Phi_{p_i}(M) \end{array}$$

Figure 2.3: Modular computation of the determinant of an integer matrix

2.10 Lagrange interpolation

Modular computations usually go through a step of "reconstruction" where the modular results are combined to produce the desired result. In the case of the modular computation of the determinant of an integer matrix, the reconstruction step was achieved by means of the Chinese Remaindering Algorithm. A special case of this process is *Lagrange Interpolation*, of which we will make intensive use in Chapters 3 and 4.

Definition 13 *Let $u = (u_0, \ldots, u_{n-1})$ be a sequence of pairwise distinct elements of the field $\mathbb{K}$. For $i = 0 \cdots n-1$ the i-th* Lagrange interpolant *is the polynomial*

$$L_i(x) = \prod_{\substack{0 \leq j < n \\ j \neq i}} \frac{x - u_j}{u_i - u_j} \tag{2.10}$$

with the property that

$$L_i(u_j) = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{otherwise} \end{cases} \tag{2.11}$$

Proposition 6 *Let $v_0, \ldots, v_{n-1}$ be in $\mathbb{K}$. There is a* ***unique*** *polynomial $f_1 \in \mathbb{K}[X]$ with degree less than n and such that*

$$f(u_i) = v_i \text{ for } i = 0 \cdots n-1. \tag{2.12}$$

Moreover this polynomial is given by

$$f = \sum_{0 \leq i < n} v_i \, L_i(x). \tag{2.13}$$

Proposition 6 leads to Specification 1 where we introduce a crucial operation in our algorithms. It is important to observe that this operation "reconstructs" one variable, namely X_1, for multivariate polynomials in $\mathbb{K}[X_1, \ldots, X_n]$. Hence, this is an application of Proposition 6 to a more general context. However, this operation does not deal with the simultaneous reconstruction of several variables. This is a much more difficult task, which is not used in this thesis.

Specification 1 *Let n and d be positive integers. Let $f_0, \ldots, f_d \in \mathbb{K}[X_2, \ldots, X_n]$ be polynomials. By convention, we have $\mathbb{K}[X_2, \ldots, X_n] = \mathbb{K}$. For all $i = 0, \ldots, d$, we define:*

$$f_i = \sum_{m \in S} C_{m,i}\, m$$

where S is the set of all monomials appearing in $f_0, \ldots, f_d$. Let $v_0, \ldots, v_d \in \mathbb{K}$ be pairwise different. By definition, the function call

$$\mathrm{Interpolate}(n, d, [v_0, \ldots, , v_d], [f_0, \ldots, f_d])$$

returns the unique polynomial

$$F = \sum_{m \in S} C_m\, m$$

such that for each $m \in S$ we have:

(i) $C_m \in \mathbb{K}[X_1]$ *and* $\deg(C_m) \leq d$,

(ii) $C_m(v_i) = C_{m,i}$, *for all* $i = 0, \ldots, d$.

2.11 Rational function reconstruction

Lagrange interpolation reconstructs a polynomial F of $\mathbb{K}[X_1, \ldots, X_n]$ from homomorphic images of F in $\mathbb{K}[X_2, \ldots, X_n]$. Another important reconstruction process is *Rational Function Reconstruction* where one aims at reconstructing rational functions from polynomials.

Let $p \in \mathbb{K}[X]$ be a univariate polynomial of degree $n > 0$ with coefficients in the field $\mathbb{K}$. Given a polynomial $f \in \mathbb{K}[X]$ of degree less than n and an integer $d \in \{1, \ldots, n\}$, we want to find a rational function $r/t \in \mathbb{K}(X)$ with $r, t \in \mathbb{K}[X]$ satisfying

$$\gcd(r,t) = 1 \text{ and } rt^{-1} \equiv f \mod p, \ \deg r < d, \ \deg t \leq n - d. \tag{2.14}$$

Let us denote this problem by $RFR(p, n, f, d)$. A solution to it is given by the following.

Proposition 7 *Let $r_j, s_j, t_j \in \mathbb{K}[X]$ be the j-th row of the Extended Euclidean Algorithm applied to (p, f) where j is minimal such that $\deg r_j < d$. Then we have:*

- (i) *Problem $RFR(p, n, f, d)$ admits a solution if and only if $\gcd(r_j, t_j) = 1$,*

- (i) *if $\gcd(r_j, t_j) = 1$ holds, then a solution is $(r, t) = (w_j^{-1} r_j, w_j^{-1} t_j)$ where $w_j = \mathrm{lc}(t_j)$.*

Specification 2 *Let n and b be positive integers. Let $p \in \mathbb{K}[X_1]$ of degree $b > 0$. Let $F \in \mathbb{K}[X_1, \ldots, X_n]$. We write:*

$$F = \sum_{m \in S} C_m \, m$$

where S is the set of all monomials appearing in F. By definition, the function call RatRec(n, b, p, F) *returns a polynomial $R \in \mathbb{K}(X_1)[X_2, \ldots, X_n]$ where*

$$R = \sum_{m \in S} \mathsf{RatRec}(p, b, C_m, b \text{ quo } 2) \ m$$

where RatRec$(p, b, C_m, b \text{ quo } 2)$ *returns a solution to $RFR(p, b, C_m, b \text{ quo } 2)$ if any, otherwise returns failure.*

2.12 Ideal, radical ideal and squarefree-ness

When solving systems of polynomial equations, two notions play a central role: the *ideal* and the *radical ideal* generated by a polynomial set. The second one is a generalization of the notion of *squarefree-ness*. We review these concepts in this section.

Let $X_1, \ldots, X_n$ be n variables ordered by $X_1 < \cdots < X_n$. Recall that $\mathbb{K}$ is a field and that $\mathbb{K}[X_1, \ldots, X_n]$ denotes the ring of multivariate polynomials in $X_1, \ldots, X_n$ with coefficients in $\mathbb{K}$.

Definition 14 *Let $F = \{f_1, \ldots, f_m\}$ be a finite subset of $\mathbb{K}[X_1, \ldots, X_n]$. The* ideal generated *by F in $\mathbb{K}[X_1, \ldots, X_n]$, denoted by $\langle F \rangle$ or $\langle f_1, \ldots, f_m \rangle$, is the set of all polynomials of the form*

$$h_1 f_1 + \cdots + h_m f_m$$

where $h_1, \ldots, h_m$ are in $\mathbb{K}[X_1, \ldots, X_n]$. If the ideal $\langle F \rangle$ is not equal to the entire polynomial ring $\mathbb{K}[X_1, \ldots, X_n]$, then $\langle F \rangle$ is said to be a proper ideal.

Definition 15 *The* radical *of the ideal generated by F, denoted by $\sqrt{\langle F \rangle}$, is the set of polynomials $p \in \mathbb{K}[X_1, \ldots, X_n]$ such that there exists a positive integer e satisfying $p^e \in \langle F \rangle$. The ideal $\langle F \rangle$ is said to be radical if we have $\langle F \rangle = \sqrt{\langle F \rangle}$.*

Remark 3 *Let $f_1, \ldots, f_m \in \mathbb{K}[X_1]$ be univariate polynomials. The Euclidean Algorithm implies that the ideal $\langle f_1, \ldots, f_m \rangle$ is equal to $\langle g \rangle$, where $g = \gcd(f_1, \ldots, f_m)$. This means that there exists polynomials $a_1, \ldots, a_m, b_1, \ldots, b_m \in \mathbb{K}[X_1]$ such that we have*

$$a_1 f_1 + \cdots + a_m f_m = g \ \text{ and } \ f_i = b_i g \ \text{ for } \ i = 1, \ldots, e.$$

Therefore, every ideal of $\mathbb{K}[X_1]$ is generated by a single element.

Definition 16 *A univariate polynomial $f \in \mathbb{K}[X_1]$ is said to be* squarefree *if for all non-constant polynomials $g \in \mathbb{K}[X_1]$ the polynomial g^2 does not divide f.*

Remark 4 *Let $f \in \mathbb{K}[X_1]$ be non-constant. It is not hard to see that the ideal $\langle f \rangle \subseteq \mathbb{K}[X_1]$ is radical if and only if f is squarefree.*

2.13 Triangular set and regular chain

As we shall observe in Chapters 3 and 4, a typical symbolic solution of a polynomial system is a polynomial set with a triangular shape, that is, a so-called triangular set. Among all triangular sets, the regular chains form a subclass with rich algorithmic properties. We review these two notions in this section.

Definition 17 *A family of non-constant polynomials* $T_1, \ldots, T_e$ *in* $\mathbb{K}[X_1, \ldots, X_n]$ *is a* triangular set *if their leading variables are pairwise distinct. The triangular set* $T_1, \ldots, T_e$ *is a* Lazard triangular set *if* $e = n$ *and if for all* $1 \leq i \leq e$ *the leading coefficient of* T_i *w.r.t.* X_i *is equal to* 1.

Remark 5 *Hence* $X_1^2 + X_2, X_1X_2 + 1$ *is not a triangular set, whereas* $X_1^2 + 1, X_1X_2 + 1$ *is a triangular set but not a Lazard triangular set. Finally* $X_1^2 + 1, X_2 - 1$ *is a Lazard triangular set.*

Definition 18 *Let* $T = T_1, \ldots, T_e$ *be a triangular set in* $\mathbb{K}[X_1, \ldots, X_n]$ *such that the leading variable of* T_i *is* X_i *for all* $1 \leq i \leq e$*. The set* T *is a* regular chain *if for all* $1 \leq i \leq e$ *the leading coefficient of* T_i *w.r.t.* X_i *is invertible modulo the ideal* $\langle T_1, \ldots, T_{i-1} \rangle$*. Note that any Lazard triangular set is in particular a regular chain.*

Proposition 8 *Let* $T = T_1, \ldots, T_n$ *be a regular chain in* $\mathbb{K}[X_1, \ldots, X_n]$*. If the ideal* $\langle T_1, \ldots, T_n \rangle$ *is radical then the residue class ring* $\mathbb{K}[X_1, \ldots, X_n]/\langle T_1, \ldots, T_n \rangle$ *is isomorphic with a direct product of fields (DPF).*

Remark 6 *The interest of DPFs lies in the celebrated* D5 Principle *(Della Dora, Dicrescenzo & Duval, 1985). If* $\mathbb{L}$ *is a DPF, then one can compute with* $\mathbb{L}$ *as if it was a field: it suffices to split the computations into cases whenever a zero-divisor is met.*

Example 4

$\mathbb{K}[X_1]/\langle X_1(X_1+1)\rangle$ *can be represented with a DPFs as below:*

$$\mathbb{K}[X_1]/\langle X_1(X_1+1)\rangle \simeq \mathbb{K}[X_1]/\langle X_1\rangle \oplus \mathbb{K}[X_1]/\langle X_1+1\rangle \simeq \mathbb{K}\oplus\mathbb{K}.$$

Definition 19 *Let $T = T_1, \ldots, T_n$ be a regular chain in $\mathbb{K}[X_1, \ldots, X_n]$. Assume that $\langle T_1, \ldots, T_n\rangle$ is radical. Denote by $\mathbb{L}$ the DPF given by*

$$\mathbb{L} = \mathbb{K}[X_1, \ldots, X_n]/\langle T_1, \ldots, T_n\rangle.$$

Let y be an extra variable. Let f_1, f_2, g be polynomials in $\mathbb{K}[X_1, \ldots, X_n, y]$ such that f_1 and f_2 have positive degree w.r.t. y. We say that g is a GCD *of f_1, f_2 if the following conditions hold*

(G_1) *the leading coefficient h of g w.r.t. y is invertible in $\mathbb{L}$,*

(G_2) *there exist polynomials $A_1, A_2 \in \mathbb{K}[X_1, \ldots, X_n, y]$ such that $g = A_1f_1 + A_2f_2$ in $\mathbb{L}[y]$, that is, there exist polynomials $Q_1, \ldots, Q_n \in \mathbb{K}[X_1, \ldots, X_n]$ such that we have*

$$g = A_1f_1 + A_2f_2 + Q_1T_1 + \cdots + Q_nT_n.$$

(G_3) *if g has positive degree w.r.t. y then g pseudo-divides f_1 and f_2 in $\mathbb{L}[y]$, that is, there exist non-negative integers b, c and polynomials $C_1, C_2 \in \mathbb{K}[X_1, \ldots, X_n, y]$ and polynomials $Q_1, \ldots, Q_n, S_1, \ldots, S_n \in \mathbb{K}[X_1, \ldots, X_n]$ such that we have*

$$h^bf_1 = C_1g + Q_1T_1 + \cdots + Q_nT_n \text{ and } h^cf_2 = C_2g + S_1T_1 + \cdots + S_nT_n.$$

Remark 7 *As we shall see in Chapters 3 and 4, GCDs of univariate polynomials over DPFs are powerful tools for solving systems of polynomial equations. However, with the notations of Definition 19, a GCD of f_1 and f_2 need not exist. Proposition 9 and 10, proved in [28], overcome this difficulty The first one is the $n = 1$ case of the second one.*

For the purpose of Chapter 3 the $n = 1$ case is sufficient.

Proposition 9 *Let $T_1 \in \mathbb{K}[X_1]$ be a non-constant squarefree polynomial. Denote by $\mathbb{L}$ the DPF given by*

$$\mathbb{L} = \mathbb{K}[X_1]/\langle T_1 \rangle.$$

Let y be an extra variable. Let f_1, f_2 be polynomials in $\mathbb{K}[X_1, y]$ such that f_1 and f_2 have positive degree w.r.t. y. Then, there exist univariate polynomials $B_1, \ldots, B_e$ in $\mathbb{K}[X_1]$ and polynomials $A_1, \ldots, A_e$ in $\mathbb{K}[X_1, y]$ such that the following properties hold

(G_4) *the product $B_1 \cdots B_e$ equals T_1,*

(G_5) *for all $1 \leq i \leq e$, the polynomial A_i is a GCD of f_1, f_2 in $(\mathbb{K}[X_1]\langle B_i \rangle)[y]$.*

The sequence $(A_1, \{B_1\}), \ldots, (A_e, \{B_e\})$ is called a GCD sequence *of f_1 and f_2 in $(\mathbb{K}[X_1]/\langle T_1 \rangle)[y]$.*

Example 5 *Let $f_1 = X_1X_2 + (X_1+1)(X_2+1)$ and $f_2 = X_1(X_2+1) + (X_1+1)(X_2+1)$ be polynomials over $\mathbb{K}[X_1]/\langle X_1(X_1+1) \rangle$ then*

$$\mathrm{GCD}(\mathsf{f}_1, \mathsf{f}_2, \mathbb{L}) = \begin{cases} X_2 + 1 & \mod X_1 \\ 1 & \mod X_1 + 1 \end{cases}$$

Remark 8 *We explain two ways for computing GCD sequences in the context of Proposition 9 and we refer to [28] for the context of Proposition 10.*

First, one can adapt the Euclidean Algorithm (or its variant, Algorithm 4) as follows:

(1) *Run the Euclidean Algorithm in $\mathbb{L}[y]$ as if $\mathbb{L}$ were a field*

(2) *when a division by an element a of $\mathbb{L}$ is required, then check whether a is invertible or not (using Proposition 3).*

(3) *If a is a zero-divisor, then split the computations into cases such that in each branch a becomes either zero or invertible (such splitting is possible since $\mathbb{L}$ is a direct product of fields). Then restart the GCD computation in each branch using the same strategy.*

Secondly, one can make use of the subresultant PRS algorithm and generalize the approach developed in Remark 2. Since $\mathbb{L}$ is now a DPF and not necessarily a field (as it is the case in Remark 2) one needs to modify the procedure described in Remark 2 as follows:

(1) *Consider all regular subresultants of f_1, f_2 by increasing index.*

(2) *Let j be the smallest index i such that $sres_i(f_1, f_2)$ is a regular subresultant whose leading coefficient is not mapped to zero by Φ.*

(3) *If this leading coefficient is a zero-divisor then split the computations and restart "from scratch" in each branch.*

(4) *If this leading coefficient is invertible then $\Phi(S_j)$ is a GCD of $\Phi(f_1)$ and $\Phi(f_2)$ in $\mathbb{L}[y]$.*

Proposition 10 *Let $T = \{T_1(X_1), T_2(X_1, X_2), \ldots, T_n(X_1, \ldots, X_n)\}$ be a regular chain in $\mathbb{K}[X_1, \ldots, X_n]$. Assume that $\langle T_1, \ldots, T_n \rangle$ is radical. Denote by $\mathbb{L}$ the DPF given by*

$$\mathbb{L} = \mathbb{K}[X_1, \ldots, X_n]/\langle T_1, \ldots, T_n \rangle.$$

Let y be an extra variable. Let f_1, f_2, g be univariate polynomials in $\mathbb{L}[y]$ such that f_1 and f_2 have positive degree w.r.t. y. Then, there exist regular chains $\mathbf{T}^1, \ldots, \mathbf{T}^e$ in $\mathbb{K}[X_1, \ldots, X_n]$. and polynomials $A_1, \ldots, A_e$ in $\mathbb{K}[X_1, \ldots, X_n, y]$ such that the following properties hold

(G_4) *the ideals $\langle \mathbf{T}^1 \rangle, \ldots, \langle \mathbf{T}^e \rangle$ are pairwise coprime and their intersection is equal $\langle \mathbf{T} \rangle$,*

(G_5) *for all* $1 \le i \le e$*, the polynomial* A_i *is a GCD of* f_1, f_2 *in* $(\mathbb{K}[X_1, \ldots, X_e]\langle \mathbf{T}^i \rangle)[y]$.

The sequence $(A_1, \mathbf{T}^1), \ldots, (A_e, \mathbf{T}^e)$ *is called a* GCD sequence *of* f_1 *and* f_2 *in*

$$(\mathbb{K}[X_1, \ldots, X_e]/\langle \mathbf{T} \rangle)[y].$$

Specification 3 *Let* $T = \{T_1(X_1), T_2(X_1, X_2), \ldots, T_n(X_1, \ldots, X_n)\}$ *be a regular chain in* $\mathbb{K}[X_1, \ldots, X_n]$*. The function call* Normalize(T) *returns a regular chain*

$$N = \{N_1(X_1), N_2(X_1, X_2), \ldots, N_n(X_1, \ldots, X_n)\}$$

such that $\langle T \rangle = \langle N \rangle$ *holds and for all* $1 \le i \le n$ *the leading coefficient of* N_i *is* 1.

2.14 Lifting

Let X_1, X_2, X_3 be variables ordered by $X_1 < X_2 < X_3$ and let f_1, f_2 be in $\mathbb{K}[X_1, X_2, X_3]$. Let $\mathbb{K}(X_1)$ be the field of rational univariate functions with coefficients in $\mathbb{K}$. We denote by $\mathbb{K}(X_1)[X_2, X_3]$ the ring of bivariate polynomials in X_2 and X_3 with coefficients in $\mathbb{K}(X_1)$. Let π be the projection on the X_1-axis. For $X_1 \in \overline{\mathbb{K}}$, we denote by Φ_{x_1} the evaluation map from $\mathbb{K}[X_1, X_2, X_3]$ to $\overline{\mathbb{K}}[X_2, X_3]$ that replaces X_1 with x_1. We make the following assumptions:

- the ideal $\langle f_1, f_2 \rangle$ (generated by f_1 and f_2 in $\mathbb{K}[X_1, X_2, X_3]$) is radical,
- there exists a triangular set $\mathbf{T} = \{T_2, T_3\}$ in $\mathbb{K}(X_1)[X_2, X_3]$ such that $\mathbf{T}$ and f_1, f_2 generate the same ideal in $\mathbb{K}(X_1)[X_2, X_3]$.

Proposition 11 is proved in [11, Proposition 3] and an algorithm for Specification 4 appears in [29].

Proposition 11 *Let* x_1 *be in* $\overline{\mathbb{K}}$. *If* x_1 *cancels no denominator in* $\mathbf{T}$, *then the fiber* $V(f_1, f_2) \cap \pi^{-1}(x_1)$ *satisfies*

$$V(f_1, f_2) \cap \pi^{-1}(x_1) \;=\; V(\Phi_{x_1}(T_2), \Phi_{x_1}(T_3)).$$

Specification 4 *Let* x_1 *be in* $\mathbb{K}$. *Let* $N_2(X_2), N_3(X_2, X_3)$ *be a Lazard triangular set in* $\mathbb{K}[X_2, X_3]$ *such that we have*

$$V(\Phi_{x_1}(f_1), \Phi_{x_1}(f_2)) = V(N_2, N_3).$$

We assume that the Jacobian matrix of $\Phi_{x_1}(f_1), \Phi_{x_1}(f_2)$ *is invertible modulo the ideal* $\langle N_2, N_3 \rangle$. *Then the function call* $\mathsf{Lift}(f_1, f_2, N_2, N_3, x_1)$ *returns the triangular set* $\mathbf{T}$.

2.15 Fast polynomial arithmetic over a field

For univariate polynomial over a field, fast algorithms are available for computing products, quotients, remainders and GCDs. By fast, we mean algorithms whose running time is "quasi-linear" in the size of their output. From these fundamental fast algorithms, such as FFT-based univariate multiplication, one can derive fast interpolation and fast rational function reconstruction. See Chapters 8 to 11 in [17] for details. We list below some of the main complexity results in this area.

Proposition 12 *Let* f_1, f_2 *in* $\mathbb{K}[X]$ *with degrees less than* d. *Then, one can compute* $\gcd(f_1, f_2)$ *in* $O(d \log^2(d) \log(\log(d)))$ *operations in* $\mathbb{K}$.

Proposition 13 *Let* d *be a positive integer. Let* $v_0, v_1, \ldots, v_d$ *be pairwise different values in* $\mathbb{K}$ *and let* $u_0, u_1, \ldots, u_d$ *be values in* $\mathbb{K}$. *Then, one can compute the unique polynomial* f *in* $\mathbb{K}[X]$ *of degree* d *such that* $f(v_i) = u_i, i = 0, \ldots, d$ *in* $O(d \log^2(d) \log(\log(d)))$ *operations in* $\mathbb{K}$.

Proposition 14 *Let $m \in \mathbb{K}[X]$ be of degree $d > 0$ and let $f \in \mathbb{K}[X]$ be of degree less than d. There is an algorithm which decides whether there are two polynomials r and t in $\mathbb{K}[X]$ of degree less than $d/2$ in $\mathbb{K}[X]$ such that*

$$\gcd(r, t) = 1 \text{ and } rt^{-1} = f \mod m$$

and, if so, they can be computed in $O(d \log^2(d) \log(\log(d)))$ operations in $\mathbb{K}$.

Chapter 3

A Modular Method for Bivariate Systems

In this chapter we discuss an algorithm and its implementation for solving systems of two non-linear polynomial equations with two variables. This algorithm relies on well-known algebraic tools and techniques: Lagrange interpolation and subresultants. The emphasis is on designing an efficient implementation based on two ideas: *modular arithmetic* and *recycling intermediate computations*. We also make an assumption on the solution set in order to avoid technical difficulties, since they are irrelevant in most practical cases. We report on an implementation in the computer algebra system `Maple` and provide experimental comparisons with another symbolic solver implemented in this system.

3.1 Problem statement

Let f_1, f_2 be two polynomials in the variables X_1, X_2 and with coefficients in a field $\mathbb{K}$. Let $\overline{\mathbb{K}}$ be the algebraic closure of $\mathbb{K}$. An important property of $\overline{\mathbb{K}}$ is that $\overline{\mathbb{K}}$ is infinite [31], even if $\mathbb{K}$ is a finite field such as $\mathbb{Z}/p\mathbb{Z}$, for a prime number p. For most practical systems, one can think of $\mathbb{K}$ as being the field $\mathbb{R}$ of real numbers and of $\overline{\mathbb{K}}$ as the field $\mathbb{C}$ of complex numbers. We assume that $\mathbb{K}$ is a perfect field [31]. The fields $\mathbb{K}$ that we have in mind, namely $\mathbb{R}$ and $\mathbb{Z}/p\mathbb{Z}$ for a prime number p, are perfect fields. We are interested in solving over $\overline{\mathbb{K}}$ the system of equations

$$\begin{cases} f_1(X_1, X_2) = 0 \\ f_2(X_1, X_2) = 0 \end{cases} \tag{3.1}$$

that is computing the set of all couples of numbers $(z_1, z_2) \in \overline{\mathbb{K}}^2$ such that:

$$f_1(z_1, z_2) = f_2(z_1, z_2) = 0.$$

This set is usually denoted by $V(f_1, f_2)$ where V stands for *variety*. We denote by Z_1 the set of all the numbers $z_1 \in \overline{\mathbb{K}}$ such that there exists a number $z_2 \in \overline{\mathbb{K}}$ such that

$$(z_1, z_2) \in V(f_1, f_2).$$

In other words, the set Z_1 collects all the values for the X_1-coordinate of a point in $V(f_1, f_2)$. Let h_1 and h_2 be the leading coefficients w.r.t. X_2 of f_1 and f_2, respectively. Note that h_1 and h_2 belong to $\mathbb{K}[X_1]$. We make here our assumptions regarding f_1, f_2 and $V(f_1, f_2)$:

- (H_1) the set $V(f_1, f_2)$ is non-empty and finite, and thus the set Z_1 is non-empty and finite too,
- (H_2) there exists a constant d_2 such that for every $z_1 \in Z_1$ there exist exactly d_2 points in $V(f_1, f_2)$ whose X_1-coordinate is z_1,
- (H_3) the polynomials f_1 and f_2 have positive degree w.r.t. X_2,
- (H_4) the resultant of f_1 and f_2 w.r.t. X_2 is squarefree,
- (H_5) the polynomials h_1 and h_2 are relatively prime, that is, we have $\gcd(h_1, h_2) = 1$.

Hypotheses (H_1) and (H_2) deal with the shape of the solution set of $V(f_1, f_2)$ These assumptions are satisfied in most practical problems, see for instance the systems collected by the `SymbolicData` project [30]. Moreover, we usually have $d_2 = 1$. The celebrated *Shape Lemma* [2] is a theoretical confirmation of this empirical observation.

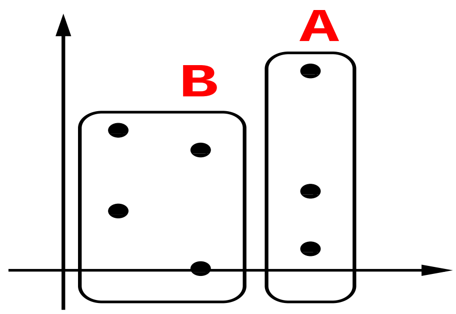

Figure 3.1: Each of sets A and B satisfy (H_2), but their union does not.

Hypotheses (H_3) to (H_5) will lead to a simple algorithm for computing $V(f_1, f_2)$. This will help in designing the modular methods presented in this chapter. These hypotheses are easy to relax without making the solving process asymptotically more expensive. However, this does introduce a number of special cases which makes the solving process quite technical.

Let d_1 be the number of elements in Z_1. The hypothesis (H_1) and (H_2) imply that there exists a univariate polynomial $t_1 \in \mathbb{K}[X_1]$ of degree d_1 and a bivariate polynomial $t_2 \in \mathbb{K}[X_1, X_2]$ with degree d_2 w.r.t. X_2, such that for all $(z_1, z_2) \in \overline{\mathbb{K}}^2$ we have

$$\begin{cases} f_1(z_1, z_2) &= 0 \\ f_2(z_1, z_2) &= 0 \end{cases} \iff \begin{cases} t_1(z_1) &= 0 \\ t_2(z_1, z_2) &= 0 \end{cases}$$

Moreover, we can require that the leading coefficient of t_1 w.r.t. X_1 and the leading coefficient of t_2 w.r.t. X_2 are both equal to 1. We summarize these statements in the following proposition. This result also gives additional conditions in order to make $\{t_1, t_2\}$ unique. A proof of Proposition 15 appears, for instance, in [1]. Moreover, we will prove it while establishing Corollary 1.

Proposition 15 *Under the assumptions* (H_1) *and* (H_2)*, there exist non-constant polynomials* $t_1 \in \mathbb{K}[X_1]$ *and* $t_2 \in \mathbb{K}[X_1, X_2]$ *such that we have*

$$V(f_1, f_2) = V(t_1, t_2).$$

Moreover, one can require

$$\mathrm{lc}(t_1, X_1) = \mathrm{lc}(t_2, X_2) = 1$$

With the additional conditions below, the triangular set $\{t_1, t_2\}$ *is uniquely determined:*

- t_1 *is squarefree,*
- t_2 *is squarefree as a univariate polynomial in* $(\mathbb{K}[X_1]/\langle t_1\rangle)[X_2]$.

The triangular set $\{t_1, t_2\}$ can be seen as a *symbolic* or *formal* description of the solution set $V(f_1, f_2)$ of the input system (3.1). Consider for instance the system:

$$\begin{cases} X_1^2 + X_2 + 1 &= 0 \\ X_1 + X_2^2 + 2 &= 0 \end{cases} \tag{3.2}$$

Then, after a substitution, one obtains the following triangular set

$$\begin{cases} X_1^4 + 2X_1^2 + X_1 + 3 &= 0 \\ X_2 + X_1^2 + 1 &= 0 \end{cases} \tag{3.3}$$

Therefore, we have $d_1 = 4$ and $d_2 = 1$.

The goal of this chapter is to compute the triangular set $\{t_1, t_2\}$ efficiently. We start by describing a *direct method* in Section 3.2. Then, in Section 3.3, we describe a first and natural modular method, based on the *Euclidean Algorithm*, that we call the *Euclidean mod-*

ular method. In Section 3.5, we present a second and more advanced modular method, based on *subresultants* and that we call the *subresultant modular method*.

In Section 3.6 we describe our implementation environment and in Section 3.7 we report on our benchmark results for those three methods.

3.2 A direct method

From now on, based on our assumption (H_3) we regard our input polynomials f_1 and f_2 as univariate polynomials in X_2 with coefficients in $\mathbb{K}[X_1]$. Let R_1 be the resultant of f_1 and f_2 in this context. The polynomial R_1 is thus an element of $\mathbb{K}[X_1]$. Proposition 16 makes a first observation.

Proposition 16 *The polynomial R_1 is non-zero and non-constant.*

Proof. Assume first by contradiction that $R_1 = 0$ holds. Since $\mathbb{K}[X_1]$ is a UFD, one can apply Proposition 4 page 13. Then, the assumption $R_1 = 0$ implies that f_1 and f_2 have a common factor g of positive degree in $(\mathbb{K}[X_1])[X_2]$. Let h be the leading coefficient of g w.r.t. X_2; hence h belongs to $\mathbb{K}[X_1]$. Observe that for all $z_1 \in \overline{\mathbb{K}}$ such that $h(z_1) \neq 0$ there exists $z_2 \in \overline{\mathbb{K}}$ such that $g(z_1, z_2) = 0$. Since g divides f_1 and f_2, every such pair (z_1, z_2) belongs to $V(f_1, f_2)$. Since $\overline{\mathbb{K}}$ is infinite and since the number of roots of h is finite, we have found infinitely many points in $V(f_1, f_2)$, which contradicts our assumption (H_1). Therefore we have proved that $R_1 \neq 0$ holds.

Assume now that R_1 is a non-zero constant. From the Extended Euclidean Algorithm (see Algorithms 3 page 11 and 4 page 13) we know that there exist polynomials a_1 and a_2 in $(\mathbb{K}[X_1])[X_2]$ such that we have:

$$a_1 f_1 + a_2 f_2 = R_1.$$

From assumption (H_1) we have $V(f_1, f_2) \neq \emptyset$. Thus, we can choose $(z_1, z_2) \in V(f_1, f_2)$. Let us evaluate each side of the above equality at this point. Clearly we have:

$$a_1(z_1, z_2)f_1(z_1, z_2) + a_2(z_1, z_2)f_2(z_1, z_2) = 0 \text{ and } R_1(z_1, z_2) \neq 0,$$

which is a contradiction. Therefore we have proved that R_1 has a positive degree w.r.t. X_1. □

Recall that f_1 and f_2, regarded as univariate polynomials in $(\mathbb{K}[X_1])[X_2]$, have a non-constant common factor if and only if their resultant R_1 is null. Recall also that there exist polynomials $A_1, A_2 \in (\mathbb{K}[X_1])[X_2]$ such that we have

$$A_1 f_1 + A_2 f_2 = R_1. \tag{3.4}$$

Observe that if $(z_1, z_2) \in V(f_1, f_2)$ then we have $R_1(z_1) = 0$. However, not all roots of R_1 may extend to a common solution of f_1 and f_2, as shown by the following example.

Example 6 *Consider for instance the system:*

$$\begin{cases} X_1 X_2^2 + X_1 + 1 &= 0 \\ X_1 X_2^2 + X_2 + 1 &= 0 \end{cases}$$

The resultant of these polynomials w.r.t. X_2 *is*

$$R_1 = X_1^4 + X_1^2 + X_1 = X_1(X_1^3 + X_1 + 1)$$

The solutions of the system are given by the following regular chain:

$$\begin{cases} X_2 - X_1 &= 0 \\ X_1^3 + X_1 + 1 &= 0 \end{cases}$$

which can be verified using the `Triangularize` *function of the* `RegularChains` *library in* `Maple`*. One can see that the root $X_1 = 0$ of R_1 does not extend to a solution of the system. This is clear when replacing X_1 by 0 in the input system.*

The following two propositions deal with the difficulty raised by the above example. The first one handles the situations encountered in practice. The second one gives a sufficient condition for any root z_1 of the resultant R_1 to extend to a point (z_1, z_2) of $V(f_1, f_2)$. Both are particular instances of a general theorem in algebraic geometry, called the *Extension Theorem* [10].

Proposition 17 *Let t_1 be a non-constant factor of R_1. Let h be the GCD of h_1 and h_2. (Recall that h_1 and h_2 are the leading coefficients w.r.t. X_2 of f_1 and f_2 respectively.) We assume that h and t_1 are relatively prime. Then for every root z_1 of t_1 there exists $z_2 \in \overline{\mathbb{K}}$ such that (z_1, z_2) belongs to $V(f_1, f_2)$.*

Proof. Let $z_1 \in \overline{\mathbb{K}}$ be a root of t_1. Let Φ be the evaluation map from $(\mathbb{K}[X_1])[X_2]$ to $\overline{\mathbb{K}}[X_2]$ that replaces X_1 by z_1. This is a ring homomorphism. Recall that we denote by h_1 and h_2 the leading coefficients of f_1 and f_2 w.r.t. X_2. Since h and t_1 are relatively prime, and since z_1 is a root of t_1, either $\Phi(h_1) \neq 0$ or $\Phi(h_2) \neq 0$ holds. (Indeed $\Phi(h_1) = \Phi(h_2) = 0$ would imply $\Phi(h) = 0$ contradicting the fact that h and t_1 are relatively prime.) Hence, we can assume $\Phi(h_1) \neq 0$. Let n and m be the degrees w.r.t. X_2 of f_2 and $\Phi(f_2)$, respectively. By virtue of the *Specialization property of the subresultants* (Theorem 3 page 23) we have

$$\Phi(R_1) = \Phi(h_1)^{n-m}\mathsf{res}(\Phi(f_1), \Phi(f_2)).$$

Since t_1 divides R_1 and $\Phi(t_1) = 0$ we have $\Phi(R_1) = 0$. Since $\Phi(h_1)^{n-m} \neq 0$, we have

$$\mathsf{res}(\Phi(f_1), \Phi(f_2)) = 0.$$

This implies that $\Phi(f_1)$ and $\Phi(f_2)$ have a non-trivial common factor in $\overline{\mathbb{K}}[X_2]$ and thus a common root z_2 in $\overline{\mathbb{K}}$. Hence we have $\Phi(f_1)(z_2) = \Phi(f_2)(z_2) = 0$, that is, $f_1(z_1, z_2) = f_2(z_1, z_2) = 0$. □

Proposition 18 *Assumption* (H_5) *implies that for any root* z_1 *of* R_1*, there exists* $z_2 \in \overline{\mathbb{K}}$ *such that* (z_1, z_2) *belongs to* $V(f_1, f_2)$*,*

Proof. Let z_1 be a root of R_1. Let Φ be the evaluation map from $(\mathbb{K}[X_1])[X_2]$ to $\overline{\mathbb{K}}[X_2]$ that replaces X_1 by z_1. Since the leading coefficients of f_1 and f_2 w.r.t. X_2 are relatively prime, at least one of them does not map to zero by Φ. Then one can proceed similarly to the proof of Proposition 18. Let us sketch this briefly.

Recall that we denote by h_1 and h_2 the leading coefficients of f_1 and f_2 w.r.t. X_2. We can assume $\Phi(h_1) \neq 0$. By virtue of the *Specialization property of the subresultants* we have

$$\Phi(R_1) = \Phi(h_1)^{n-m} \mathsf{res}(\Phi(f_1), \Phi(f_2)).$$

Since z_1 is a root of R_1 we have $\Phi(R_1) = 0$. Hence, we have $\mathsf{res}(\Phi(f_1), \Phi(f_2)) = 0$. This implies that $\Phi(f_1)$ and $\Phi(f_2)$ have a non-trivial common factor in $\overline{\mathbb{K}}[X_2]$ and thus a common root z_2 in $\overline{\mathbb{K}}$, concluding the proof. □

We shall discuss now how to compute the solution set $V(f_1, f_2)$. Proposition 19 serves as a lemma for Theorem 5 and Corollary 1 which are the *main results* of this section.

Proposition 19 *Let* t_1 *be a non-constant factor of* R_1*. Let* t_2 *be a polynomial in* $\mathbb{K}[X_1, X_2]$ *such that* t_2 *is a GCD of* f_1 *and* f_2 *in* $(\mathbb{K}[X_1]/\langle t_1 \rangle)[X_2]$*. The hypotheses* (H_1)*,* (H_3) *and* (H_4) *imply the four following properties:*

(i) *if* t_2 *has degree zero w.r.t.* X_2 *then we have* $V(t_1, t_2) = \emptyset$*,*

(ii) *if* t_2 *has positive degree w.r.t.* X_2 *then we have*

$$V(t_1, t_2) \neq \emptyset \text{ and } V(t_1, t_2) \subseteq V(f_1, f_2),$$

(iii) *we have* $V(t_1, t_2) = V(f_1, f_2, t_1)$*,*

(iv) *if we have* $t_1 = R_1$ *then* $V(f_1, f_2) = V(t_1, t_2)$ *holds.*

Proof. Let h be the leading coefficient of t_2 w.r.t. X_2. Since t_2 is a GCD of f_1 and f_2 in $(\mathbb{K}[X_1]/\langle t_1 \rangle)[X_2]$, we have the following three properties:

- h is invertible modulo t_1,
- there exist polynomials $A_3, A_4 \in \mathbb{K}[X_1, X_2]$ such that $A_3 f_1 + A_4 f_2 = t_2$ holds in $(\mathbb{K}[X_1]/\langle t_1 \rangle)[X_2]$,
- if t_2 has a positive degree w.r.t. X_2, then the polynomial t_2 pseudo-divides f_1 and f_2 in $(\mathbb{K}[X_1]/\langle t_1 \rangle)[X_2]$.

Therefore, we have:

- there exist univariate polynomials $A_{10}, A_{11} \in \mathbb{K}[X_1]$ such that we have

$$A_{10} h + A_{11} t_1 = 1. \tag{3.5}$$

- there exists a polynomial $A_5 \in \mathbb{K}[X_1, X_2]$

$$A_3 f_1 + A_4 f_2 = t_2 + A_5 t_1. \tag{3.6}$$

- if t_2 has a positive degree w.r.t. X_2, then there exist polynomials $A_6, A_7, A_8, A_9 \in \mathbb{K}[X_1, X_2]$ and non-negative integers c, b such that we have

$$h^b f_1 = A_6 t_2 + A_7 t_1 \ \text{ and } \ h^c f_2 = A_8 t_2 + A_9 t_1. \tag{3.7}$$

Observe that Equation (3.5) implies that a root of t_1 cannot cancel h. Recall also that from Equation (3.4) there exist polynomials $A_1, A_2 \in (\mathbb{K}[X_1])[X_2]$ such that we have $A_1 f_1 + A_2 f_2 = R_1$.

We prove (i). We assume that t_2 has degree zero w.r.t. X_2. Hence, we have $t_2 = h$. Since t_1 and h have no common root, it follows that $V(t_1, t_2) = \emptyset$ holds.

We prove (ii). We assume that t_2 has a positive degree w.r.t. X_2. Since t_1 and h have no common root, every root z_1 of t_1 extends to (at least) one root z_2 of t_2 (where t_2 is specialized at $X_1 = z_1$). Hence $V(t_1, t_2) \neq \emptyset$ holds. Consider now $(z_1, z_2) \in V(t_1, t_2)$. From Equations (3.5) and (3.7) we clearly have $f_1(z_1, z_2) = f_2(z_1, z_2) = 0$, that is $(z_1, z_2) \in V(f_1, f_2)$.

We prove (iii). From (i) and (ii), whether the degree of t_2 w.r.t. X_2 is positive or not, we have $V(t_1, t_2) \subseteq V(f_1, f_2)$ which implies trivially $V(t_1, t_2) \subseteq V(f_1, f_2, t_1)$. The converse inclusion results immediately from Equation (3.6).

We prove (iv). Assume that $t_1 = R_1$ holds. From (H_1) we know that $V(f_1, f_2)$ is not empty. Hence, thanks to (i) and (ii), we notice that the polynomial t_2 must have a positive degree w.r.t. X_2. Let us consider $(z_1, z_2) \in V(f_1, f_2)$. From Equation (3.4) we deduce $t_1(z_1, z_2) = 0$. Then, with Equation (3.6) we obtain $t_2(z_1, z_2) = 0$, that is, $(z_1, z_2) \in V(t_1, t_2)$. □

Theorem 5 *Let h be the GCD of h_1 and h_2. Let $(A_1, \{B_1\}), \ldots, (A_e, \{B_e\})$ be a GCD sequence of f_1 and f_2 in $(\mathbb{K}[X_1]/\langle R_1 \operatorname{quo} h\rangle)[X_2]$. The hypotheses (H_1), (H_3) and (H_4) imply*

$$V(f_1, f_2) = V(A_1, B_1) \cup \cdots \cup V(A_e, B_e) \cup V(h, f_1, f_2). \tag{3.8}$$

Moreover, for all $1 \leq i \leq e$, the polynomial A_i has a positive degree w.r.t. X_2 and thus $V(A_i, B_i) \neq \emptyset$.

Proof. Since $V(f_1, f_2) \subseteq V(R_1)$ and $V(R_1) = V(R_1 \operatorname{quo} h) \cup V(h)$ we deduce

$$V(f_1, f_2) = V(R_1 \operatorname{quo} h, f_1, f_2) \cup V(h, f_1, f_2). \tag{3.9}$$

From Hypothesis (H_4) the polynomial R_1 is squarefree and thus $R_1 \text{ quo } h$ is squarefree too. Hence, the residue class ring $(\mathbb{K}[X_1]/\langle R_1 \text{ quo } h \rangle)$ is a direct product of fields and computing polynomial GCDs in $(\mathbb{K}[X_1]/\langle R_1 \text{ quo } h \rangle)[X_2]$ make clear sense and can be achieved by adapting either the Euclidean Algorithm or the Subresultant Algorithm, as explained in Remark 8 page 32. The polynomials $B_1, \ldots, B_e$ are non-constant polynomials in $\mathbb{K}[X_1]$ and their product is equal to $R_1 \text{ quo } h$. Hence, we have

$$V(R_1 \text{ quo } h, f_1, f_2) = V(B_1, f_1, f_2) \ \cup \ \cdots \ \cup \ V(B_e, f_1, f_2). \qquad (3.10)$$

The polynomials $A_1, \ldots, A_e$ are polynomials in $\mathbb{K}[X_1][X_2]$ and for all $1 \leq i \leq e$ the polynomial A_i is a GCD of f_1 and f_2 in $(\mathbb{K}[X_1]/\langle B_i \rangle)[X_2]$. Property (iii) of Proposition 19 implies for all $1 \leq i \leq e$ we have:

$$V(B_i, f_1, f_2) \ = \ V(B_i, A_i). \qquad (3.11)$$

Equations (3.9), (3.10) and (3.11) imply Equation (3.8). Now observe that for all $1 \leq i \leq e$ the polynomial B_i is a factor of $R_1 \text{ quo } h$ which is relatively prime with h. (Indeed, since R_1 is squarefree, the polynomial $R_1 \text{ quo } h$ is relatively prime with h.) Hence, by virtue of Proposition 17 we know that $V(B_i, f_1, f_2) \neq \emptyset$ holds. Therefore, with Proposition 19 we deduce that A_i has a positive degree w.r.t. X_2 and that $V(A_i, B_i) \neq \emptyset$ holds. □

Corollary 1 *The hypotheses (H_1) to (H_5) imply the existence of a polynomial $G_2 \in \mathbb{K}[X_1, X_2]$ with the following properties:*

(i) *G_2 has a positive degree w.r.t. X_2 and its leading coefficient w.r.t. X_2 is invertible modulo R_1,*

(ii) *$V(f_1, f_2) = V(R_1, G_2)$,*

(iii) *G_2 is a GCD of f_1 and f_2 in $(\mathbb{K}[X_1]/\langle R_1 \rangle)[X_2]$.*

Proof. With the notations and hypotheses of Theorem 5, Equation (3.8) holds. From Hypothesis (H_5) the polynomials h_1 and h_2 are relatively prime and thus $h = 1$, hence we have $V(h, f_1, f_2) = \emptyset$, leading to

$$V(f_1, f_2) = V(A_1, B_1) \cup \cdots \cup V(A_e, B_e). \tag{3.12}$$

Since for all $1 \leq i \leq e$ the residue class ring $\mathbb{K}[X_1]/\langle B_i \rangle$ is a direct product of fields, one can use GCD computations in order to compute the squarefree part C_i of A_i in $(\mathbb{K}[X_1]/\langle B_i \rangle)[X_2]$. Note that we have $V(A_1, B_1) = V(C_i, B_i)$ for all $1 \leq i \leq e$. We fix an index i in the range $1 \ldots e$ and a root $z_1 \in \mathbb{K}$ of B_i. Let Φ be the evaluation map from $\mathbb{K}[X_1][X_2]$ to $\overline{\mathbb{K}}[X_2]$ that replaces X_1 by z_1. Since the coefficient field $\mathbb{K}$ is perfect, and since C_i is squarefree in $(\mathbb{K}[X_1]/\langle B_i \rangle)[X_2]$, the polynomial $\Phi(C_i)$ has e_i distinct roots, where e_i is the degree of $\Phi(C_i)$. Observe that from Hypothesis (H_2) we have $e_i = d_2$. Therefore all polynomials $C_1, \ldots, C_e$ have the same degree d_2. Since the polynomials $B_1, \ldots, B_e$ are pairwise coprime, by applying the Chinese Remainder Theorem, one can compute a polynomial $G_2 \in \mathbb{K}[X_1, X_2]$ such that we have

$$V(G_2, B_1 \cdots B_e) = V(C_1, B_1) \cup \cdots \cup V(C_e, B_e). \tag{3.13}$$

Since the leading coefficients of $C_1, \ldots, C_e$ are invertible modulo $B_1, \ldots, B_e$ respectively, the leading coefficient of G_2 is invertible modulo $B_1 \cdots B_e$. Recall that $R_1 = B_1 \cdots B_e$ holds, hence we have $V(f_1, f_2) = V(R_1, G_2)$. Finally, the fact that G_2 is a GCD of f_1 and f_2 is routine manipulation of polynomials over direct products of fields. □

To conclude this long section, let us emphasize the results stated in Theorem 5 and Corollary 1.

First, it is important to observe that Theorem 5 remains true even if Hypothesis (H_4) does not hold. This can be shown using the theory and algorithms developed in [27]. Moreover, based on [27], relaxing Hypotheses (H_1) and (H_3) is easy. Therefore, Theorem 5 "essentially" tells us how to solve any bivariate system.

Secondly, it is important to observe that Corollary 1 requires two additional hypotheses w.r.t. Theorem 5, namely (H_2) and (H_5). These latter hypotheses have the following benefits. They allow us to replace Equation (3.8) simply by

$$V(f_1, f_2) = V(R_1, G_2) \tag{3.14}$$

Hence, under the hypotheses (H_1) to (H_5), solving the bivariate system $f_1 = f_2 = 0$ reduces to one resultant and one GCD computation. This will be the key for developing an efficient modular method. Based on these observations, we introduce the following.

Specification 5 *Let (again)* $f_1, f_2 \in \mathbb{K}[X_1, X_2]$ *be two non-constant polynomials.*

- *We denote by* $\mathsf{Solve2}(f_1, f_2)$ *a function returning pairs* $(A_1, B_1), \ldots, (A_e, B_e)$ *where* $B_1, \ldots, B_e$ *are univariate polynomials in* $\mathbb{K}[X_1]$ *and* $A_1, \ldots, A_e$ *are bivariate polynomials in* $\mathbb{K}[X_1, X_2]$ *such that* (A_i, B_i) *is a regular chain for all* $1 \leq i \leq e$ *and we have*

$$V(f_1, f_2) = V(A_1, B_1) \cup \cdots \cup V(A_e, B_e).$$

- *Under the Hypotheses* (H_1) *to* (H_5)*, we denote by* $\mathsf{GenericSolve2}(f_1, f_2)$ *a function returning* (G_2, R_1) *where* R_1 *is a univariate polynomial in* $\mathbb{K}[X_1]$ *and* G_2 *is a bivariate polynomial in* $\mathbb{K}[X_1, X_2]$ *such that* (G_2, R_1) *is a regular chain and we have*

$$V(f_1, f_2) = V(R_1, G_2).$$

Under the Hypotheses (H_1) to (H_5), Corollary 1 tells us that computing the targeted triangular set $\{R_1, G_2\}$ reduces to

- computing the resultant $R_1 = \mathsf{res}(f_1, f_2, X_2)$,
- computing G_2 as GCD of f_1 and f_2 modulo $\{R_1\}$.

The **first key observation** is that this GCD need not split the computations. Hence this GCD can be computed as if $\mathbb{K}[X_1]\langle R_1 \rangle$ was a field. Thus, using Remark 8 page 32, one can deduce this GCD from the subresultant chain of f_1 and f_2 that we have computed anyway in order to obtain $\mathsf{res}(f_1, f_2, X_2)$.

The **second key observation** is that these subresultant chains can be computed by a modular method, using evaluation and interpolation.

In the following, we describe three strategies for implementing solvers. In the first one we are looking at an approach which identify genericity assumptions for the solving of a system to be simple and efficient at the same time, this leads to **GenericSolve2** that is a function which has a simple algorithm but it requires the genericity assumptions H_1-H_5, this algorithm (see Algorithm 7) simply computes the resultant R_1 of the input polynomials and their GCD modulo R_1 assuming we already have the necessary non-fast (non-modular) techniques to find both the resultant and the GCD.

Algorithm 7 GenericSolve2

Input: $f_1, f_2 \in K[X_1, X_2]$, *under assumptions of*

(H_1) $0 < |V(f_1, f_2)| < \infty$

(H_2) $V(f_1, f_2)$ *equiprojectable*

(H_3) $\deg(f_i, X_j) > 0$

(H_4) R_1 *squarefree*

(H_5) $\gcd(\mathsf{lc}(f_1, X_2), \mathsf{lc}(f_2, X_2)) = 1$

Output: $\{R_1, G_2\} \subseteq K[X_1, X_2]$ *s.t.* $V(R_1, G_2) = V(f_1, f_2)$.

$\mathrm{GenericSolve2}(f_1, f_2) ==$

$R_1 := \mathsf{res}(f_1, f_2, X_2)$

$G_2 :=$ *GCD*$(f_1, f_2, \{R_1\})$

return (R_1, G_2)

In the second algorithm (Algorithm 8) we implement **GenericSolve2** by way of fast techniques using moular methods that is computing both R_1 and G_2 by specializations and interpolations hence we call this approach **ModularGenericSolve2**.

Algorithm 8 ModularGenericSolve2

Input: $f_1, f_2 \in K[X_1, X_2]$, *under* (H_1) *to* (H_5)

Output: $\{R_1, G_2\} \subseteq \mathbb{K}[X_1, X_2]$ *s.t.* $V(R_1, G_2) = V(f_1, f_2)$.

(1) **Compute** *Chain*(f_1, f_2) *by interpolation*

(2) **Let** $R_1 = \mathsf{res}(f_1, f_2, X_2)$

(3) **Let** $\mathbb{L} = \mathbb{K}[X_1]/\langle R_1 \rangle$

(4) **Let** $\Psi : \mathbb{K}[X_1, X_2] \to \mathbb{L}[X_2]$

(5) $i := 1$

(6) **Let** $S \in \mathrm{Chain}(f_1, f_2)$ *non-zero, regular with minimum index* $j \geq i$

(7) **if** $\Psi(\mathrm{lc}(S, X_2)) \neq 0$

(7.1) **then** $G_2 := \Psi(S)$

(7.2) **else** $j := i + 1$; **goto** *(6)*

(8) **return** $\{R_1, G_2\}$

Now we can improve Algorithm 8 by relaxing all the assumptions except $H1$ and H_3 where these two assumptions are very likely to hold in most practical cases provided that we pass $h = \gcd(\mathrm{lc}(f_1, X_2), \mathrm{lc}(f_2, X_2))$ as an input in to the algorithm.

As we are removing the equiprojectablity property (Assumption H_2), we need to check whether the computation splits, if so we have to continue in each branch in the same manner until we get the true GCD, Algorithm 9 describes the steps.

Algorithm 9 EnhancedModularGenericSolve2

Input: $f_1, f_2, \mathbf{h}$, (H_1), (H_3) *only*

Output: $\{R_1, G_2\} \subseteq \mathbb{K}[X_1, X_2]$ *s.t.* $V(R_1, G_2) = V(f_1, f_2) \setminus V(h)$

(1) **Compute** *SubresultantPRS*(f_1, f_2)

(2) **Let** $R_1 = \texttt{sqfrPart}(\mathsf{res}(f_1, f_2, X_2))$

(3) $R_1 := R_1 \text{ quo } (\texttt{sqfrPart}(\mathbf{h}))$

(4) $Tasks := [[R_1, 1]]; Results := [\,]$

(5) **while** *Tasks* $\neq [\,]$ **repeat**

 # A Task is a paire consisting of a polynomial and an index

(6) $[B, i] := Tasks[1]$

(7) $Tasks := Tasks[2 \cdots -1]$

(8) **Let** $S \in \text{Chain}(f_1, f_2)$ *non-zero regular with minimum index* $j \geq i$

(9) $c := \text{lc}(S, X_2); c := c \ \ rem \ B$

(10) **if** $c = 0$ **then**

(11) $Tasks := [[B, i+1], op(Tasks)]$

(12) **else**

(13) $B_1 := \gcd(c, B)$

(14) **if** $B_1 = 1$ **then**

(15) $Results := [[B, S], op(Results)]$

(16) **else**

(17) $B_2 := B \ quo \ B_1$

(18) $Tasks := [[B_1, i+1], op(Tasks)]$

(19) $Results := [[B_2, S], op(Results)]$

(20) **return** *Results*

Finally we provide a general solver **ModularSolve2** (see Algorithm 10) that does not make any assumptions in which can easily detect when genercity assumptions hold and it will also handle the cases where these assumptions do not hold.

The idea is to take advantage of using **EnhancedModularGenericSolve2** when $h \neq 0$ otherwise we can just take the tail of both of the input polynomials that is f_1 and f_2 without the highest degree monomial term and then repeat the processor by calling **ModularSolve2** recursively.

Algorithm 10 ModularGenericSolve2

Input: $f_1, f_2 \in K[X_1, X_2]$, *under assumptions of*

(H_1) $0 < |V(f_1, f_2)| < \infty$

(H_3) $\deg(f_i, X_j) > 0$

Output: $\{R_1, G_2\} \subseteq K[X_1, X_2]$ *s.t.* $V(R_1, G_2) = V(f_1, f_2)$.

ModularSolve2(f_1, f_2) ==

$h := \gcd(\mathsf{lc}(f_1, X_2), \mathsf{lc}(f_2, X_2))$

$R :=$ EnhancedModularGenericSolve2(f_1, f_2, h) $\boxed{h \neq 0}$

if $h = 1$ **return** R

$D :=$ ModularSolve2$(\mathsf{tail}(f_1), \mathsf{tail}(f_2))$ $\boxed{h = 0}$

for $(A(X_1), B(X_1, X_2)) \in D$ **repeat**

$g := \gcd(A, h)$

if *deg*$(g, X_1) > 0$ **then**

$R := R \cup \{(g, B)\}$

return R

Sections 3.3 and 3.5 provides details about the computations with both of the Euclidean sequences and Subresultant chains, Figure 3.2 sketches the general idea of computing by

$$
\begin{array}{ccc}
f_1, f_2 \in \mathbb{K}[X_1, X_2] & \xrightarrow{\Phi_i : X_1 \to x_i,\ 0 \le i \le \delta} & \mathrm{Chain}(\Phi_i(f_1), \Phi_i(f_2)) \in \mathbb{K}[X_2] \\
\texttt{RegularChains} \Big\downarrow & & \Big\downarrow Interp \\
\mathrm{Chain}(\Psi(f_1), \Psi(f_2)) \in \mathbb{L}[X_2] & \xleftarrow{\Psi : \mathbb{K}[X_1] \to \mathbb{L}} & \mathrm{Chain}(f_1, f_2) \in \mathbb{K}[X_1, X_2] \\
 & \mathbb{L} = \mathbb{K}[X_1]/\langle R_1 \rangle &
\end{array}
$$

Figure 3.2: Modular solving of 2×2 polynomial system.

modular methods.

3.3 Euclidean modular method

In this section we propose a first modular approach based on the Euclidean Algorithm:

(1) Instead of computing R_1 directly from the input polynomials f_1 and f_2, we choose enough values $v_0, v_1, \ldots, v_b$ of x_1, compute $\mathrm{eval}(R_1, X_1 = v_0), \mathrm{eval}(R_1, X_1 = v_1), \ldots, \mathrm{eval}(R_1, X_1 = v_b)$ with Algorithm 4 page 13, or equivalently Algorithm 11. Then, we reconstruct R_1 from these modular images using Lagrange interpolation (Section 2.10 page 26).

(2) Instead of computing G_2 directly from the input polynomials f_1, f_2 modulo R_1, using the Euclidean Algorithm in $(\mathbb{K}[X_1]/\langle R_1\rangle)[X_2]$ as if $\mathbb{K}[X_1]/\langle R_1\rangle$ were a field, we "recycle" the modular images of the subresultant chain of f_1 and f_2, which were computed for obtaining R_1 via Algorithm 6 page 17. Then, we reconstruct G_2 from these modular images using Lagrange interpolation (Section 2.10 page 26) and rational function reconstruction (Section 2.11 page 27).

Let us give some details for (R_1). Let b be a degree bound for $\mathsf{res}(f_1, f_2, X_2)$ given by Theorem 4. Then, it suffices to choose $b + 1$ pairwise different values $v_0, v_1, \ldots, v_b$

in $\mathbb{K}$ such that none of them cancel the leading coefficients of f_1 and f_2. Then, for each $v_i \in \{v_0, v_1, \ldots, v_b\}$ applying Algorithm 11 to $\mathrm{eval}(f_1, X_1 = v_i)$ and $\mathrm{eval}(f_2, X_1 = v_i)$, we obtain $\mathrm{eval}(R_1, X_1 = v_i)$.

For each $v_i \in \{v_0, v_1, \ldots, v_b\}$, let us store all the intermediate remainders computed by Algorithm 11 including $\mathrm{eval}(f_1, X_1 = v_i)$ and $\mathrm{eval}(f_2, X_1 = v_i)$. For each $v_i \in \{v_0, v_1, \ldots, v_b\}$, we store these remainders in array A_{v_i} such that slot of index d gives the intermediate remainder of degree d, if any, otherwise 0. In particular the slot of A_{v_i} with index 0 gives $\mathrm{eval}(R_1, X_1 = v_i)$. Combining all these resultants using Lagrange interpolation produces R_1, Algorithm 12 describes the way we used to compute the resultant R_1.

Algorithm 11 *Euclidean Sequence Algorithm*

Input *:* $a, b \in K[x]$, $m := deg(a) \geq n := deg(b) > 0$.

Output*:* $\mathrm{prs[i]}$*: polynomials of degree* i *in* EuclideanSequence*(a,b), if any, otherwise* 0.

EuclideanSequence$(a, b) ==$
    r := 1
    declare prs *as a list of size* n.
    repeat
      b_n := *lcoeff*(b, x)
      if $n = 0$ ***then***
        $\mathrm{prs}[0] :=$$r\, b_n^m$; ***return*** prs
      h := $a \;\mathrm{rem}\; b$
      if $h = 0$ ***then***
        $\mathrm{prs[n]} := (1/\mathrm{b_n}) * \mathrm{b}$; ***return*** prs
      else
        i:= *deg*(h, x)*;*
        $\mathrm{prs[i]} := (1/\mathrm{lcoeff(h, x)}) * \mathrm{h}$
        r := $r\,(-1)^{nm} b_n^{m-i}$
        (a, b, m, n) := (b, h, n, i)

Algorithm 12

Input *:* $f_1, f_2 \in K[X_1, X_2]$, *under assumptions* H_1 *to* H_5.

Output*:* $\{t_1(X_1), t_2(X_1, X_2)\} \subseteq K[X_1, X_2]$ *s.t.* $V(f_1, f_2) = V(t_1, t_2)$.

```
let b be a degree bound for res(f_1, f_2, X_2)
n := b + 1
declare vprs as a list of size n.
while n > 0 do
    v :=newValue()
    if lc(f_1(X_1 = v)) = 0 or lc(f_2(X_1 = v)) = 0 then iterate
    else
      val[n] := v
      vprs[n] :=EuclideanSequence(f_1(X_1 = v), f_2(X_1 = v))
      n := n - 1
t_1 :=Interpolate(val, seq(vprs[i][0], i = 1..b + 1))
s = 1
repeat {
    d := min { d > s | ∃i ∈ ℕ , prs[i][d] ≠ 0}
    let c be a degree bound for the coefficients w.r.t. X_2
      of a remainder in EuclideanSequence(f_1, f_2)
    n' := 2c + 1
    m := #{i | prs[i][d] ≠ 0}
    Assume {prs[i][d] ≠ 0, i = 1..m}
   # up to sorting prs, we can always make that assumption
    while n' > m do
        v :=newValue()
        if lc(f_1(X_1 = v)) = 0 or lc(f_2(X_1 = v)) = 0 then iterate
        a :=EuclideanSequence(f_1(X_1 = v), f_2(X_1 = v))
        if a[d] = 0 then iterate
        prs[n'] := a; val[n'] := v;
        n' := n' - 1
    t_2:=InterpolateAndRationalReconstruction(val, seq(prs[i][d], i = 1..2c + 1))
    if gcd(lc(t_2, X_2), t_1) ∈ 𝕂 then break
    s := s + 1
}
return { t_1, t_2 }
```

Let us now give details for (T_2). The principle is similar to (T_1). However, three particular points require special care.

- First, the degree of the gcd w.r.t. X_2 is not known in advance, in contrast with the resultant which is known to be 0 w.r.t. X_2. So, we first try to reconstruct from the array $A_{v_0}, \ldots, A_{v_b}$ a gcd of degree 1, if it fails, we try a gcd of degree 2, and so on. We proceed in this order, because unlucky substitutions give rise to gcds with degrees higher than the that of the desired gcd.

- Second, when trying to reconstruct a gcd of degree d we interpolate together the slots on index d from $A_{v_0}, \ldots, A_{v_b}$ that do not contain 0. If they all contain 0, then we know that G_2 has degree higher than d. Indeed, according to Theorem 6.26 in [17] the number of unlucky specializations is less than the degree of R_1 and thus less than b. If at least one slot of index d does not contain 0, we need to make sure that we have enough slots. Indeed, according to Theorem 6.54 in [17], the degree in X_1 a denominator of a numerator in G_2 can reach

$$c = (\deg_{X_2}(f_1) + \deg_{X_2}(f_2))\max(\deg_{X_1}(f_1), \deg_{X_1}(f_2)). \tag{3.15}$$

  If we have less than $2c$ non-zero slots in degree d, we need to compute more sequences of remainders for specializations of X_1.

- Third, once we have interpolated enough specialized remainders of a given degree d, we need to apply rational reconstruction to its coefficients w.r.t. X_2. Indeed, in contrast to resultants, gcd computations involve not only additions and multiplications, but also inversions.

Example 7 *Let f_1 and f_2 be polynomials defined as*

$$\begin{cases} f_1 &= (X_2 - X_1^2)(X_2 + 2) \\ f_2 &= (X_2 + 1)(X_2 + 3) \end{cases} \tag{3.16}$$

Then the resultant intermediate remainders of f_1 and f_2 w.r.t X_2 are:

$$R_2 = X_2^2 + 4X_2 + 3$$
$$R_1 = -2X_2 - X_1^2 X_2 - 2X_1^2 - 3$$
$$R_0 = -X_1^4 - 4X_1^2 - 3$$

Let b be the bound

$$\deg_{X_1}(f_1)\deg_{X_2}(f_2) + \deg_{X_2}(f_1)\deg_{X_1}(f_2)$$

which is 4 in our example. For enough values $v_0, v_1, ..., v_b$ compute $R_i(X_1 = v_j)$ for $i = 0..2$ and $j = 0..b$ and store the computations into an array, say A.

	$i = 1, X_1 = 1$	$i = 2, X_1 = 2$	$i = 3, X_1 = 3$	$i = 4, X_1 = 4$	$i = 5, X_1 = 5$
$R_{2,i}$	$X_2^2 + 4X_2 + 3$	$X_2^2 + 4X_2 + 3$	$X_2^2 + 4X_2 + 3$	$X_2^2 + 4X_2 + 3$	$X_2^2 + 4X_2 + 3$
$R_{1,i}$	$-3X_2 - 5$	$-6X_2 - 11$	$-11X_2 - 21$	$-18X_2 - 35$	$-27X_2 - 53$
$R_{0,i}$	-8	-35	-120	-323	-728

Recycle the intermediate computations above using Lagrange interpolation to get

$$G_2 = -X_1^4 - 4X_1^2 - 3$$

Also to reconstruct the gcd from the above array, we can interpolate together all the slots that do not contain 0, which leads to:

$$[-2X_1^2 - 3, -X_1^2 - 2]$$

At this point we have enough specialized remainders to apply rational reconstruction to the coefficients w.r.t. X_2 *to get*

$$(-2 - X_1^2)X_2 - 2X_1^2 - 3$$

Finally taking the primitive part we get

$$-2X_2 - X_1^2 X_2 - 2X_1^2 - 3$$

Hence our solution is

$$[-2X_2 - X_1^2 X_2 - 2X_1^2 - 3, -3 - 4X_1^2 - X_1^4]$$

Example 8 *Consider the system*

$$\begin{cases} f_1 &= (X_2{}^2 + 1)\,(X_2{}^2 + 3) \\ f_2 &= (X_2 + X_1)^2\,(X_2 + 2)^2 \end{cases} \tag{3.17}$$

The Euclidean Sequence of f_1 *and* f_2 *w.r.t.* X_2 *over* $\mathbb{Z}_{17}$ *can be given by:*

$$R_4 = X_2{}^4 + (4 + 2\,X_1)\,X_2{}^3 + (4 + 8\,X_1 + X_1{}^2)\,X_2{}^2 + (8\,X_1 + 4\,X_1{}^2)\,X_2 + 4\,X_1{}^2$$

$$R_3 = (15\,X_1 + 13)\,X_2{}^3 + (16\,X_1{}^2 + 9\,X_1)\,X_2{}^2 + 13\,X_1{}^2 + (13\,X_1{}^2 + 9\,X_1)\,X_2 + 3$$

$$\begin{aligned} R_2 = {} & \frac{(13\,X_1{}^4+2\,X_1{}^3+12\,X_1{}^2+8\,X_1+16)}{X_1{}^2+4\,X_1+4} X_2{}^2 + \frac{(X_1{}^4+8\,X_1{}^3+12\,X_1{}^2+10\,X_1+3)}{X_1{}^2+4\,X_1+4} X_2 \\ & + \frac{X_1{}^4+8\,X_1{}^3+15\,X_1{}^2+6\,X_1+12}{X_1{}^2+4\,X_1+4} \end{aligned}$$

$$\begin{aligned} R_1 = {} & \frac{(12\,X_1{}^7+14\,X_1{}^6+13\,X_1{}^5+16\,X_1{}^4+11\,X_1{}^3+3\,X_1{}^2+13\,X_1+1)}{X_1{}^7+7\,X_1{}^5+6\,X_1{}^4+9\,X_1{}^3+8\,X_1{}^2+2\,X_1+1} X_2 \\ & + \frac{11\,X_1{}^7+13\,X_1{}^6+X_1{}^5+X_1{}^4+16\,X_1{}^3+3\,X_1{}^2+9\,X_1+12}{X_1{}^7+7\,X_1{}^5+6\,X_1{}^4+9\,X_1{}^3+8\,X_1{}^2+2\,X_1+1} \end{aligned}$$

$$R_0 = X_1{}^8 + 8\,X_1{}^6 + 5\,X_1{}^4 + 7\,X_1{}^2 + 9$$

	$i=1, X_1=1$	$i=2, X_1=2$	$i=3, X_1=3$
$R_{4,i}$	$X_2^4+6X_2^3+13X_2^2+12X_2+4$	$X_2^4+8X_2^3+7X_2^2+15X_2+16$	$X_2^4+10X_2^3+3X_2^2+9X_2+2$
$R_{3,i}$	$X_2^3+10X_2^2+2X_2+3$	$X_2^3+11X_2^2+4X_2+8$	$X_2^3+5X_2^2+6X_2+5$
$R_{2,i}$	0	$X_2^2+X_2+3$	$X_2^2+7X_2+16$
$R_{1,i}$	0	X_2+10	X_2+5
$R_{0,i}$	13	1	1

	$i=4, X_1=4$	$i=5, X_1=5$	$i=6, X_1=6$
$R_{4,i}$	$X_2^4+14X_2^3+X_2^2+4X_2+15$	$X_2^4+16X_2^3+3X_2^2+5X_2+8$	$X_2^4+X_2^3+7X_2^2+14X_2+9$
$R_{3,i}$	$X_2^3+X_2^2+10X_2+13$	$X_2^3+X_2^2+12X_2+12$	$X_2^3+3X_2^2+14X_2+6$
$R_{2,i}$	$X_2^2+4X_2+7$	X_2^2+10	$X_2^2+15X_2+13$
$R_{1,i}$	X_2	X_2+1	X_2+7
$R_{0,i}$	9	4	1

	$i=7, X_1=7$	$i=8, X_1=8$	$i=9, X_1=9$
$R_{4,i}$	$X_2^4+3X_2^3+13X_2^2+14X_2+1$	$X_2^4+5X_2^3+4X_2^2+5X_2+1$	$X_2^4+7X_2^3+14X_2^2+4X_2+9$
$R_{3,i}$	$X_2^3+3X_2^2+16X_2+5$	$X_2^3+X_2+3$	$X_2^3+16X_2^2+3X_2+13$
$R_{2,i}$	$X_2^2+14X_2+11$	$X_2^2+16X_2+1$	$X_2^2+9X_2+12$
$R_{1,i}$	X_2+4	X_2+2	X_2+5
$R_{0,i}$	9	9	1

Figure 3.3: Computing Euclidean remainder sequience of system (3.17) through specializations mod 17

For this example each remainder R_i is of degree i w.r.t. X_2. Now instead of computing directly over $\mathbb{Z}_{17}(X_1)$ we can reduce the computation to univariate polynomials over finite field Z_{17} by using specilizations. Figure (3.3) shows the details of such computation over Z_{17} with specilizations of $X_1 = 1, \cdots, X_1 = 9$.

3.4 The complexity analysis

Referring to propositions 12, 13, and 14; let us try to find the complexity our algorithm. Just for R_1 consider

$$n = \max(\deg_{X_2}(f_1), \deg_{X_2}(f_2)) \tag{3.18}$$

$$m = \max(\deg_{X_1}(f_1), \deg_{X_1}(f_2)) \tag{3.19}$$

For the direct approach, we run the Euclidean algorithm applied to f_1 and f_2 regarded as polynomials in X_2 so we pay $O(n^2nm)$ operations in $\mathbb{K}$ (the factor nm is for the maximum degree of a coefficient in X_2, namely the resultant). For the modular approach, we assume that we are lucky, that is the degree of the GCD is 1. We also assume that we have fast univariate polynomial arithmetic in $\mathbb{K}[X_1]$ at our disposal. This is reasonable assumption for today's computer algebra systems [16].

- We need $2nm$ specialized computations. Each of these specialized computations are over a field, and allow the use of fast Euclidean Algorithms in

$$O(n \log(n)^2 \log(\log(n)))$$

  operations in $\mathbb{K}$. See [32] page 66.

- We need 3 interpolations in degree $2nm$ which can be done again with fast interpolations in $O((p \log(p)^2 \log(\log(p)))$ operations in $\mathbb{K}$ where $p = nm$. See [17] page 284.

- We need 2 rational function reconstructions requiring again $O((p \log(p)^2 \log(\log(p)))$ operations in $\mathbb{K}$.

Proposition 20 *Using our modular algorithm, solving a bivariate systems of two polynomial equations takes about $O(n^2m)$ operations in $\mathbb{K}$ versus $O(n^3m)$ for the non-modular algorithms (provided that we neglect the logarithmic terms).*

3.5 Subresultant modular method

Another way to implement the key observations stated in Section 3.3 is to use the *Subresultant PRS Algorithm* instead of the *Euclidean Algorithm*. This has several advantages:

- First, the bound for reconstructing G_2 is divided by 2. Indeed, in the case of the Euclidean Algorithm, the coefficients of G_2 are rational functions in $\mathbb{K}(X_1)$ whereas in the case of the Subresultant PRS Algorithm they are just polynomials in $\mathbb{K}[X_1]$.

- Second, rational reconstruction is not needed anymore and thus the algorithm becomes conceptually simpler.

Apart from that, the principle is similar. We start by specializing the input system at sufficiently many values of x_1 and for each specialization we compute the subresultant chains of the two input polynomials. Then, we interpolate the resultant part of each sequence to get T_1.

Algorithm 13

Input : $f_1, f_2 \in K[X_1, X_2]$, *under assumptions of* H_1 *to* H_5.

Output: $\{t_1(X_1), t_2(X_1, X_2)\} \subseteq K[X_1, X_2]$ *s.t.* $V(f_1, f_2) = V(t_1, t_2)$.

```
  let b be a degree bound for res(f1, f2, X2)
n := b + 1
declare vprs as a list of size n.
while n > 0 do
    v :=newValue()
    if lc(f1(X1 = v)) = 0 or lc(f2(X1 = v)) = 0 then iterate
    else
      val[n] := v
      vprs[n] :=SubresultantPRS(f1(X1 = v), f2(X1 = v))
      n := n − 1
t1 :=Interpolate(val, seq(vprs[i][0], i = 1..b + 1))
s = 1
repeat {
    d := min { d ≥ s | ∃i ∈ ℕ , prs[i][d] ≠ 0}
    n' := 2b + 1
    m := #{i | prs[i][d] ≠ 0}
    Assume {prs[i][d] ≠ 0, i = 1..m}
    # up to sorting prs, we can always make that assumption
    while n' > m do
        v :=newValue()
        if lc(f1(X1 = v)) = 0 or lc(f2(X1 = v)) = 0 then iterate
        a :=SubresultantPRS(f1(X1 = v), f2(X1 = v))
        if a[d] = 0 then iterate
        prs[n'] := a; val[n'] := v;
        n' := n' − 1
    t2:=Interpolate(val, seq(prs[i][d], i = 1..2c + 1)
    if gcd(lc(t2, X2), t1) ∈ 𝕂 then break
    s := s + 1
}
return { t1, t2 }
```

3.6 Implementation of our modular method

As a first step, one might wonder which kind of polynomial representation among several alternatives (e.g. distributive/recursive, sparse/dense) is best suited for facilitating our computations. Hence, we have developed a `Maple` library where different polynomial representations can be used.

3.6.1 `Maple` representation

Generally `Maple` objects are represented by a data structure called an *expression tree* (also DAG, Directed Acyclic Graphs) such that each `Maple` object is stored only once in memory. In the example below we consider the expression $(x-1)^2y^2+x+3$. The `Maple` command `dismantle` can be used to show the internal structure of a `Maple` expression.

Example 9

*> f := (x-1)^2*y+x+3;*

> dismantle(f);

```
SUM(7)
   PROD(5)
      SUM(5)
         NAME(4): x
         INTPOS(2): 1
         INTNEG(2): -1
         INTPOS(2): 1
      INTPOS(2): 2
      NAME(4): y
      INTPOS(2): 2
INTPOS(2): 1
NAME(4): x
INTPOS(2): 1
INTPOS(2): 3
INTPOS(2): 1
```

So the Maple's internal representation of the above expression DAG is shown in Figure 3.4.

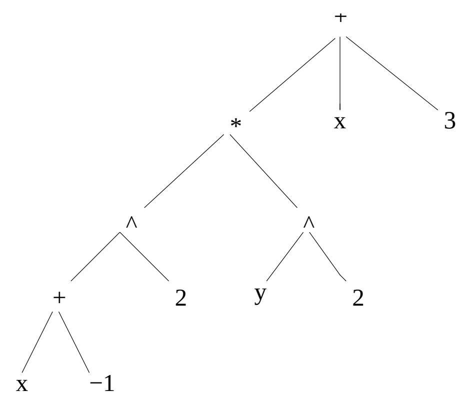

Figure 3.4: Maple expression tree for $(x-1)^2y^2+x+3$

3.6.2 Recden representation

The package Recden is a separate library which uses a recursive dense polynomial representation, that is:

- a polynomial is regarded as a univariate polynomial w.r.t. its largest variable, say v and,
- is stored as the vector of its coefficients w.r.t. v which are themselves recursive dense polynomials.

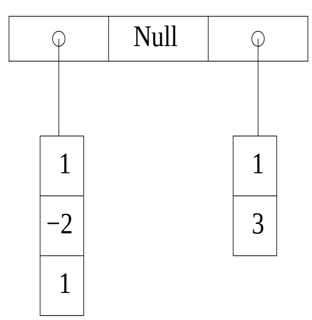

Figure 3.5: Recden representation of $(x-1)^2y^2+x+3$

In Maple 11, most of the recden routines were integrated into Maple's main library and some of them into internal built-in kernel functions.

3.6.3 `modp1/modp2` representation

The `modp1/modp2` library is a built-in `Maple` library which uses its own data structure to represent polynomials, `modp1` is a data structure for dense univariate polynomials over prime fields. If the characteristic is sufficiently small, a polynomial is represented as an array of machine integers, otherwise as a `Maple` list of arbitrary precision integers. Similarly `modp2` implements dense bivariate polynomials over prime fields as `Maple` lists of `modp1` polynomials, representing the univariate coefficients of the bivariate polynomials in the main variable.

`modp1` and `modp2` are mostly implemented in the C language rather than in the `Maple` programming language. This generally yields better performance than both of the `Maple` and `Recden` representations.

3.6.4 Benchmarks

In this section, we report on comparison with the three polynomial arithmetic representations provided by `modp1/modp2` and `Recden`. We compare them within Algorithm 12. As input, we use random bivariate dense polynomials f, g of various total degrees from 1 to 20 modulo the prime $p = 2147483659 = nextprime(2^{31})$. All the benchmarks in this thesis are obtained on a Mandrake 10 Linux machine with Pentium 2.8 GHz and 2Gb memory, except for the Figure 3.11 which was done on a Fedora 4 Linux machine with Pentium 3.2 GHz speed and 2Gb memory. Also all the timings are in seconds.

Figure 3.6 shows the running times for the `modp1/modp2` polynomial arithmetic only whereas Figure 3.7 show the running times for both `modp1/modp2` and `Recden` polynomial arithmetics. The former library clearly outperforms the latter one for large degrees. This was actually expected since most of the `modp1/modp2` code is written in C whereas the `Recden` code is mostly `Maple` interpreted code.

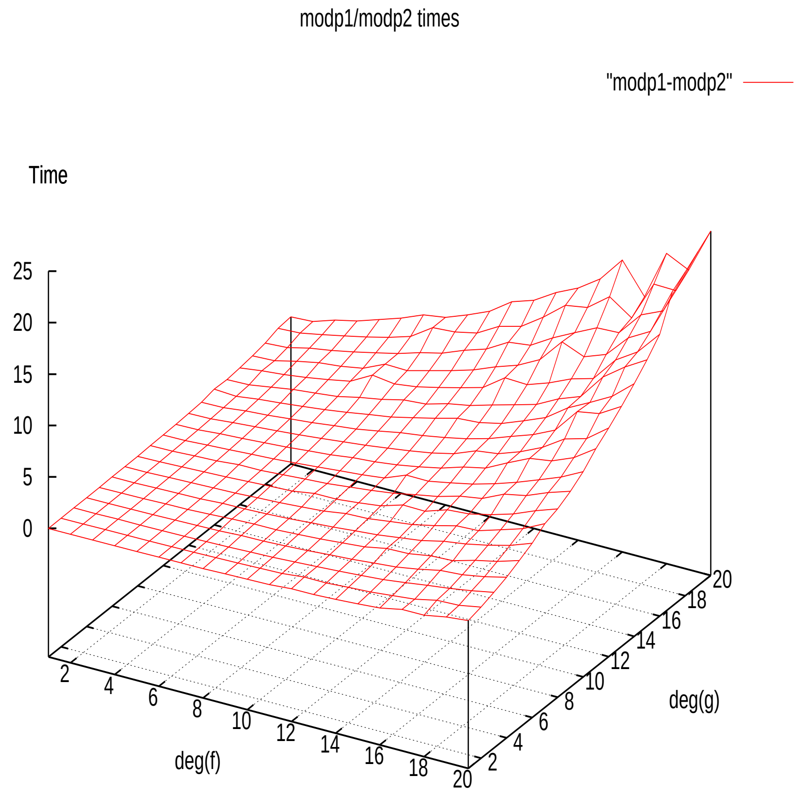

Figure 3.6: Running Euclidean modular algorithms with `modp1/modp2` representation form.

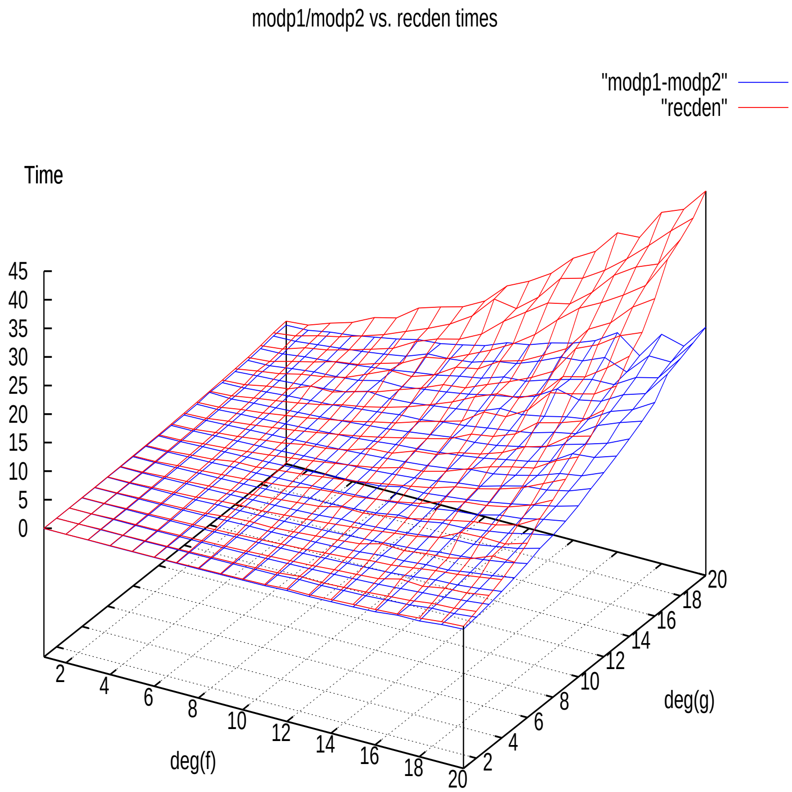

Figure 3.7: Euclidean modular method using both `Recden` and `modp1/modp2` libraries.

3.7 Experimental comparison

In this section, we compare the implementations of the modular algorithms developed in this chapter, namely Algorithms 12 and 13, based on the Euclidean sequence and Subresultant PRS sequence, respectively. We compare them also with the `Triangularize` command from the `RegularChains` library in `Maple` since they all solve polynomial systems by means of triangular sets.

As input, we use again random bivariate dense polynomials f, g of various total degrees from 1 to 20 modulo the prime $p = 2147483659 = nextprime(2^{31})$. All these implementations are using the default `Maple` polynomial representation, except for Figure 3.11 which uses `modp1/modp2` representation.

We start with benchmarks illustrating the differences between the two modular methods developed in this chapter: Figure 3.8 shows the running times for Algorithm 13 only whereas Figure 3.9 show the running times for both Algorithms 12 and 13. As expected, the Subresultant-based modular algorithm clearly outperforms the Euclidean Algorithm-based modular algorithm for large degrees. Recall that the latter one uses bounds that are essentially twice larger than the former one; moreover the latter one involves rational function reconstruction.

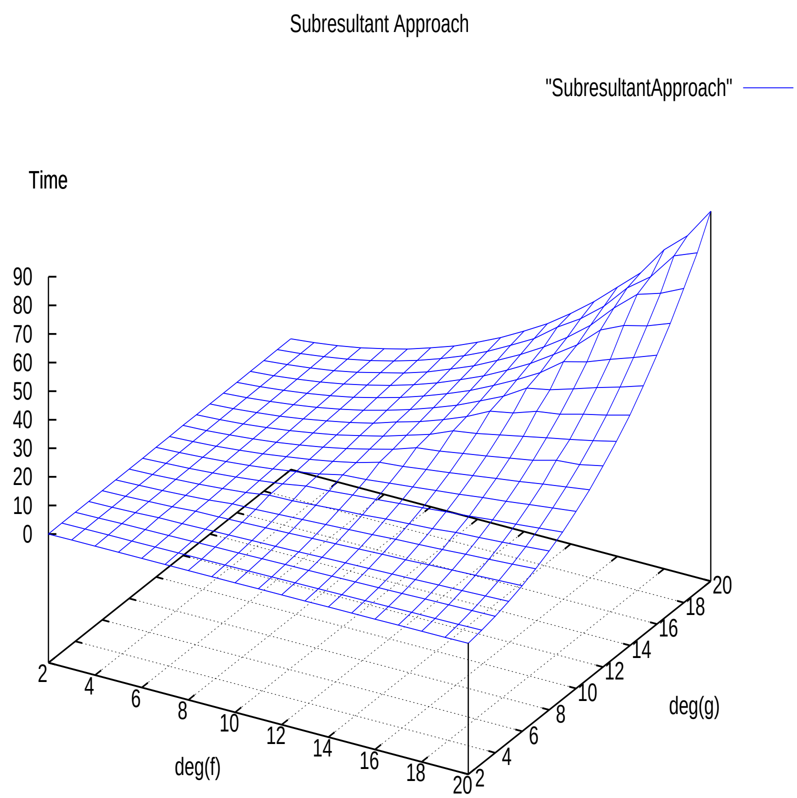

Figure 3.8: Timings for the Subresultant approach with `Maple` representation form

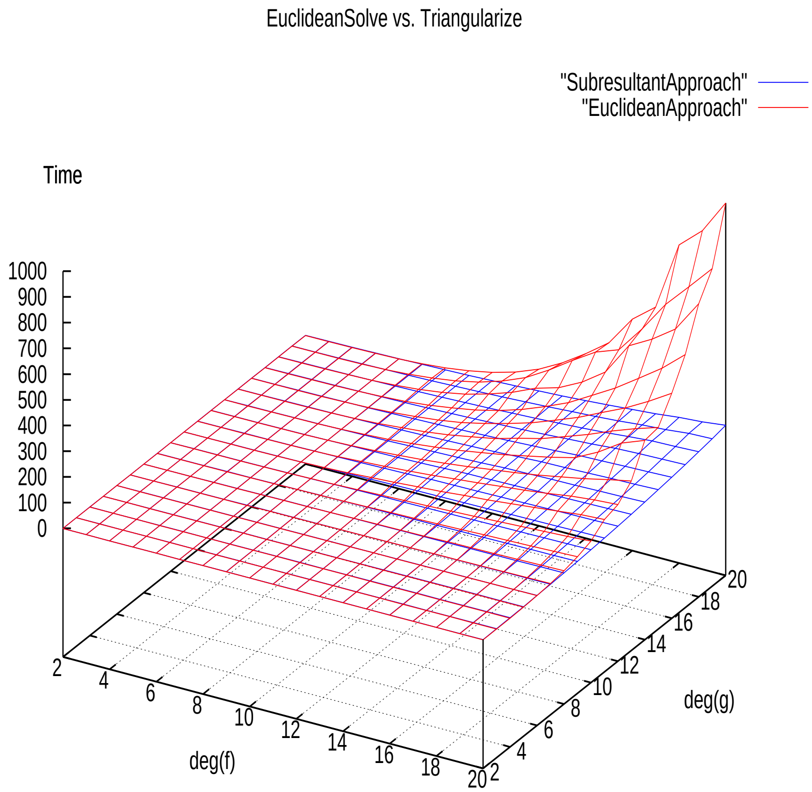

Figure 3.9: Comparison between Subresultant vs. Euclidean approaches, both are in `Maple` representation forms

Figure 3.10 shows the running times for Algorithm 13 and the `Triangularize` command. As expected, the Subresultant-based modular algorithm clearly outperforms the `Triangularize` command (which does use a modular algorithm).

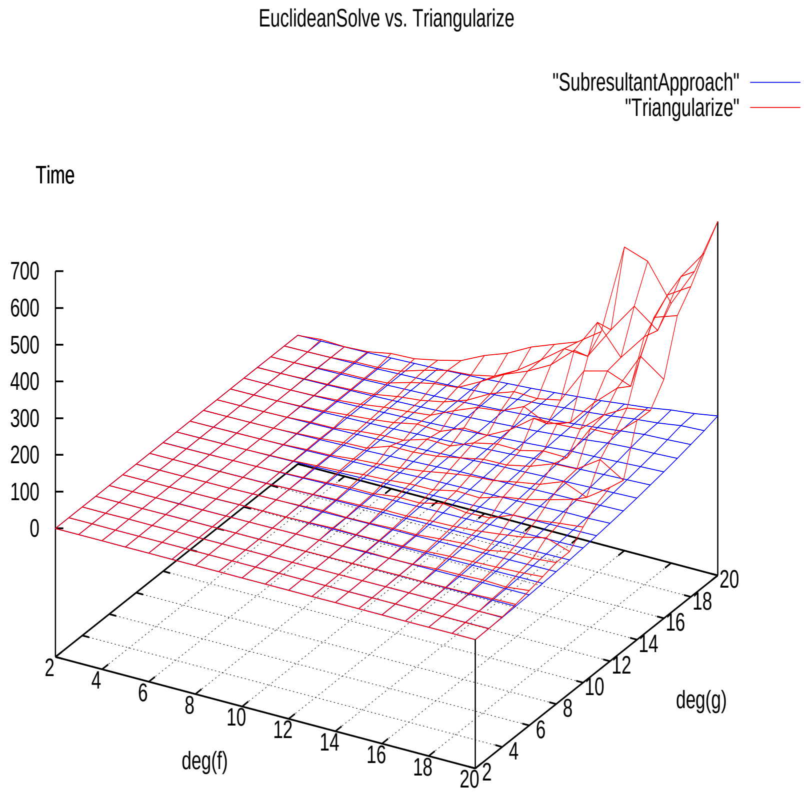

Figure 3.10: Subresultant vs. Triangularize approaches, both are in `Maple` representation forms

Finally, we compare the `Triangularize` command (which uses `Maple` polynomial arithmetic) versus Algorithm 13, implementing our Subresultant-based Modular algorithm, using the `modp1/modp2` polynomial arithmetics. In this case, the gap between becomes even larger.

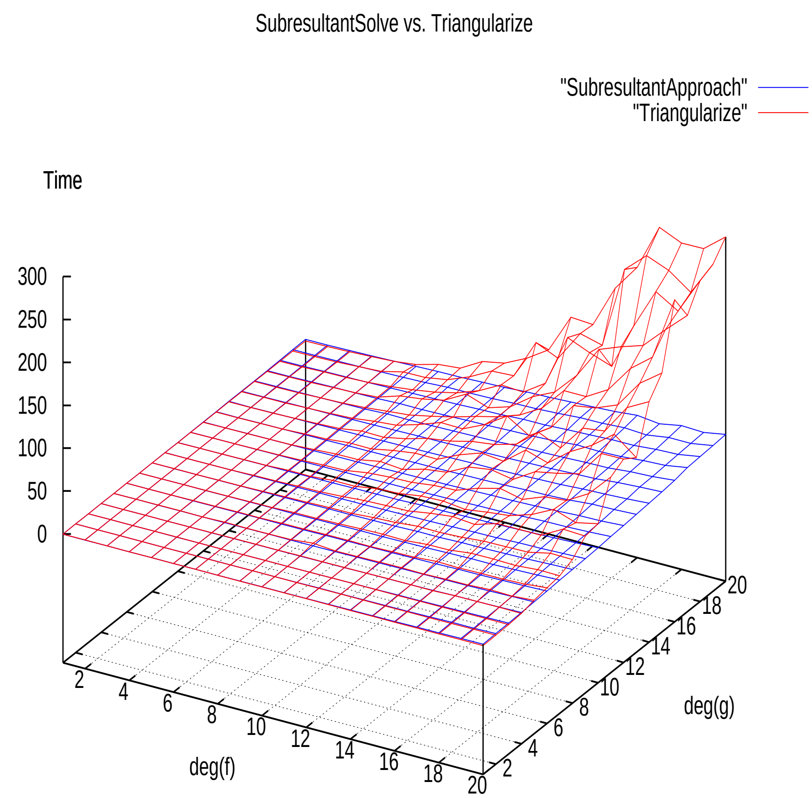

Figure 3.11: Comparison between Subresultant approach in `modp1/modp2` representation vs. Triangularize in `Maple` representation forms

Chapter 4

A Modular Method for Trivariate Systems

In this chapter we discuss an algorithm and its implementation for solving systems of three non-linear polynomial equations with three variables. We extend to this context the ideas introduced in Chapter 3. This leads to a modular method for solving such systems under some assumptions that are satisfied in most practical cases. Our implementation in the computer algebra system `Maple` allows us to illustrate the effectiveness of our approach.

4.1 Problem statement

Let f_1, f_2, f_3 be three polynomials in variables X_1, X_2, X_3 and with coefficients in a perfect field $\mathbb{K}$. Let $\overline{\mathbb{K}}$ be the algebraic closure of $\mathbb{K}$. We are interested in solving over $\overline{\mathbb{K}}$ the system of equations

$$\left\{ \begin{array}{rcl} f_1(X_1, X_2, X_3) & = & 0 \\ f_2(X_1, X_2, X_3) & = & 0 \\ f_3(X_1, X_2, X_3) & = & 0 \end{array} \right. \tag{4.1}$$

that is computing the set $V(f_1, f_2, f_3)$ of all tuples $(z_1, z_2, z_3) \in \overline{\mathbb{K}}^3$ such we have:

$$f_1(z_1, z_2, z_3) = f_2(z_1, z_2, z_3) = f_3(z_1, z_2, z_3) = 0.$$

We denote by Z_2 the set all couples $(z_1, z_2) \in \overline{\mathbb{K}}^2$ such that there exists $z_3 \in \overline{\mathbb{K}}$ satisfying $(z_1, z_2, z_3) \in V(f_1, f_2, f_3)$. Hence, the set Z_3 collects all the pairs of X_1-coordinate and X_2-coordinate of a point in $V(f_1, f_2, f_3)$.

We denote by Z_1 the set all the numbers $z_1 \in \overline{\mathbb{K}}$ such that there exists a couple $(z_2, z_3) \in \overline{\mathbb{K}}$ satisfying $(z_1, z_2, z_3) \in V(f_1, f_2, f_3)$. In other words, the set Z_1 collects all the values for the X_1-coordinate of a point in $V(f_1, f_2, f_3)$.

We make below our assumptions regarding f_1, f_2, f_3 and their zero-set $V(f_1, f_2, f_3)$:

(H_1) the set $V(f_1, f_2, f_3)$ is non-empty and finite, and thus the sets Z_1 and Z_2 are non-empty and finite too,

(H_2) there exists a constant d_3 such that for every $(z_1, z_2) \in Z_2$ there exist exactly d_3 points in $V(f_1, f_2, f_3)$ with X_1-coordinate equal to z_1 and X_2-coordinate equal to z_2,

(H_3) there exists a constant d_2 such that for every $z_1 \in Z_1$ there exist exactly d_2 values for the X_2-coordinate of a point in $V(f_1, f_2, f_3)$ whose X_1-coordinate is z_1.

(H_4) the polynomials f_1, f_2, f_3 have positive degree w.r.t. X_3 and X_2.

(H_5) there exists a Lazard triangular set $\mathbf{T} = (T_2, T_3)$ in $\mathbb{K}(X_1)[X_2, X_3]$ such that $\mathbf{T}$ and f_1, f_2 generate the same ideal in $\mathbb{K}(X_1)[X_2, X_3]$; moreover, this ideal is radical,

(H_6) let h_1 be the least common multiple of the denominators in $\mathbf{T}$; we assume that none of the roots of h_1 belongs to Z_1, that is, $V(h_1, f_1, f_2, f_3) = \emptyset$.

(H_7) We assume that $V(h_1, T_2, T_3, f_3) = \emptyset$.

(H_8) the polynomials T_2 and $\mathsf{res}(T_3, f_3, X_3)$ have positive degree w.r.t. X_2 and their leading coefficients w.r.t. X_2 are relatively prime,

(H_9) the polynomials T_3 and f_3 have positive degree w.r.t. X_3 and their leading coefficients w.r.t. X_3 are relatively prime.

Similarly to the hypotheses of Chapter 3, these assumptions are satisfied in most practical problems, see for instance the systems collected by the `SymbolicData` project [30]. Moreover, we usually have $d_2 = 1$ and $d_3 = 1$.

Let d_1 be the number of elements in Z_1. The hypotheses (H_1) to (H_3) imply that there exists a univariate polynomial $t_1 \in \mathbb{K}[X_1]$ of degree d_1, a bivariate polynomial $t_2 \in \mathbb{K}[X_1, X_2]$ with degree d_2 w.r.t. X_2, and a trivariate polynomial $t_3 \in \mathbb{K}[X_1, X_2, X_3]$ with degree d_3 w.r.t. X_3, such that for all $(z_1, z_2, z_3) \in \overline{\mathbb{K}}^3$ we have

$$\left\{\begin{array}{rcl} f_1(z_1,z_2,z_3) & = & 0 \\ f_2(z_1,z_2,z_3) & = & 0 \\ f_3(z_1,z_2,z_3) & = & 0 \end{array}\right. \iff \left\{\begin{array}{rcl} t_1(z_1) & = & 0 \\ t_2(z_1,z_2) & = & 0 \\ t_3(z_1,z_2,z_3) & = & 0. \end{array}\right.$$

Moreover, we can require that the leading coefficient of t_1, t_2, t_3 w.r.t. X_1, X_2, X_3 respectively are all equal to 1. Similarly to Chapter 3, this fact follows from a theorem of [1]. Figure 4.1 shows a variety $V(f_1, f_2, f_3)$ satisfying Hypotheses (H_1) to (H_3); such a variety is called *equiprojectable*.

The goal of this chapter is to compute the triangular set $\{t_1, t_2, t_3\}$ efficiently. Hypotheses (H_4) to (H_9) serve that objective. In Section 4.2, we describe a *direct method* based on the principle of *Incremental Solving* used in algorithms such as those of Lazard [22] and Moreno Maza [27]. That is, we first solve the system consisting of f_1 and f_2 before taking f_3 into account. Then, in Section 4.3 we present our modular method and in Section 4.4 we report on our experimental results.

4.2 A direct method

Recall that $\mathbf{T} = (T_2, T_3)$ is a Lazard triangular set in $\mathbb{K}(X_1)[X_2, X_3]$. Hence, the polynomials T_2, T_3 are bivariate in X_2, X_3 and have coefficients in the field $\mathbb{K}(X_1)$ of univariate rational functions over $\mathbb{K}$. Hypothesis (H_5) implies that for all $(z_1, z_2, z_3) \in \overline{\mathbb{K}}^3$ such that $h_1(z_1) \neq 0$ we have

$$\left\{ \begin{array}{rcl} f_1(z_1, z_2, z_3) & = & 0 \\ f_2(z_1, z_2, z_3) & = & 0 \end{array} \right. \iff \left\{ \begin{array}{rcl} T_2(z_1, z_2) & = & 0 \\ T_3(z_1, z_2, z_3) & = & 0. \end{array} \right. \tag{4.2}$$

Since h_1 is the least common multiple of the denominators in $\mathbf{T}$, we observvе that $h_1 T_2$ and $h_1 T_3$ are polynomials in $\mathbb{K}[X_1, X_2, X_3]$.

We define $W(T_2, T_3) = V(h_1 T_2, h_1 T_3) \setminus V(h_1)$, that is, the set of the points $(z_1, z_2, z_3) \in \overline{\mathbb{K}}^3$ that cancel the polynomials $h_1 T_2$ and $h_1 T_3$ without cancelling h_1. Observe that we have

$$V(f_1, f_2) \;=\; W(T_2, T_3) \,\cup\, V(h_1, f_1, f_2). \tag{4.3}$$

Two cases arise. Either h_1 is a non-zero constant and thus $V(h_1, f_1, f_2) = \emptyset$ holds. Or h_1 is a non-constant polynomial. In this latter case computing $V(h_1, f_1, f_2)$ reduces to solving a bivariate system of two equations with coefficients in $\mathbb{K}[X_1]/\langle h_1 \rangle$. One can replace h_1 by its squarefree part without changing $V(h_1, f_1, f_2)$. From there, we are led to compute resultants and GCDs over a direct product of fields, which are well understood tasks and for which highly efficient algorithms are available, see [13]. Computing the Lazard triangular set $\mathbf{T} = (T_2, T_3)$ can be achieved with Lift operation described in Specification 4. From Equation (4.3) we deduce

$$V(f_1, f_2, f_3) \;=\; (W(T_2, T_3) \cap V(f_3)) \,\cup\, V(h_1, f_1, f_2, f_3). \tag{4.4}$$

Hypothesis (H_6) tells us none of the roots of h_1 belongs to Z_1. Hence, we have:

$$V(h_1, f_1, f_2, f_3) \; = \; \emptyset. \tag{4.5}$$

Therefore, we obtain

$$V(f_1, f_2, f_3) \; = \; W(T_2, T_3) \cap V(f_3). \tag{4.6}$$

Next, Hypothesis (H_7) tells us that $V(h_1, T_2, T_3, f_3) \; = \; \emptyset$ holds. Hence Equation (4.6) becomes simply:

$$V(f_1, f_2, f_3) \; = \; V(T_2, T_3, f_3). \tag{4.7}$$

It is natural to consider the resultant of T_3 and f_3 w.r.t. X_3, namely:

$$R_2 = \mathsf{res}(T_3, f_3, X_3). \tag{4.8}$$

Hence, there exist polynomials $A_1, A_2 \in \mathbb{K}[X_1, X_2, X_3]$ such that we have

$$R_2 = A_1 T_3 + A_2 f_3. \tag{4.9}$$

The polynomials R_2 and T_2 are bivariate polynomials in $\mathbb{K}[X_1, X_2]$ and it is natural to relate their common roots with the set Z_2. Let $(z_1, z_2, z_3) \in V(f_1, f_2, f_3)$. From Equation (4.7) we have $T_2(z_1, z_2) = T_3(z_1, z_2, z_3) = 0$. Hence, with Equation (4.9) we deduce $R_2(z_1, z_2) = 0$. Therefore, we have established

$$(z_1, z_2) \in Z_2 \;\; \Rightarrow \;\; T_2(z_1, z_2) = R_2(z_1, z_2) = 0. \tag{4.10}$$

The reverse implication may not hold *a priori*. However, Hypothesis (H_9), together with the *Extension Theorem* [10], implies that every for $(z_1, z_2) \in \overline{\mathbb{K}}^2$ satisfyting $R_2(z_1, z_2) =$

0 there exists $z_3 \in \overline{\mathbb{K}}$ such that $(z_1, z_2, z_3) \in V(T_3, f_3)$. Therefore, we have

$$Z_2 \;=\; V(T_2, R_2). \tag{4.11}$$

Next, using Hypotheses (H_1), (H_3), (H_8), we can apply Corollary 1 to compute $V(R_2, T_2)$. Hence, we define

$$t_1 \;=\; \mathsf{res}(R_2, T_2, X_2) \;\text{ and }\; t_2 \;=\; \gcd(R_2, T_2, \{t_1\}), \tag{4.12}$$

and we have:

$$V(t_1, t_2) \;=\; V(R_2, T_2). \tag{4.13}$$

As observed above, using the *Extension Theorem* [10] (or a construction similar to the proofs of Theorem 5 and Corollary 1), we see that every common solution of T_2 and R_2 extends to a common solution of T_3 and f_3. Hence, using Hypothesis (H_2), there exists a trivariate polynomial t_3 of degree d_3 w.r.t. X_3 such that we have

$$V(t_1, t_2, t_3) \;=\; V(R_2, T_2, T_3, f_3). \tag{4.14}$$

Moreover, the polynomial t_3 is a GCD of T_3 and f_3 modulo $\{t_1, t_2\}$. Therefore, we have proved the following.

Theorem 6 *The hypotheses (H_1) to (H_9) imply the existence of a regular chain $\{t_1, t_2, t_3\}$ such that we have*

$$V(t_1, t_2, t_3) \;=\; V(f_1, f_2, f_3). \tag{4.15}$$

Moreover, these polynomials can be computed by Algorithm 4.3.

Algorithm 14 GenericSolve3

Input: $f_1, f_2, f_3 \in K[X_1, X_2, X_3]$ *under the assumptions* $\{(H_1)$-$(H_9)\}$.

Output: $\{t_1(X_1), t_2(X_1, X_2), t_3(X_1, X_2, X_3)\} \subseteq K[X_1, X_2, X_3]$

s.t. $V(f_1, f_2, f_3) = V(t_1, t_2, t_3)$.

$\mathrm{GenericSolve3}(f_1, f_2, f_3)==$

Let z_1 *be a random value not cancelling any coefficient w.r.t.* X_1 *of* f_2 *and* f_3

$u_2, u_3 :=$ GenericSolve2$(f_1(X_1 = z_1), f_2(X_1 = z_1))$

$u_2, u_3 :=$ Normalize(u_2, u_3)

$T_2, T_3 :=$ Lift$(f_1, f_2, f_3, u_2, u_3, X_1 - z_1)$

$R_2 :=$ res(T_3, f_3, X_3)

$t_1 :=$ res(T_2, R_2, X_2)

$t_2 := GCD(T_2, R_2, \{t_1\})$

$t_3 := GCD(T_3, f_3, \{t_1, t_2\})$

return $\{t_1, t_2, t_3\}$

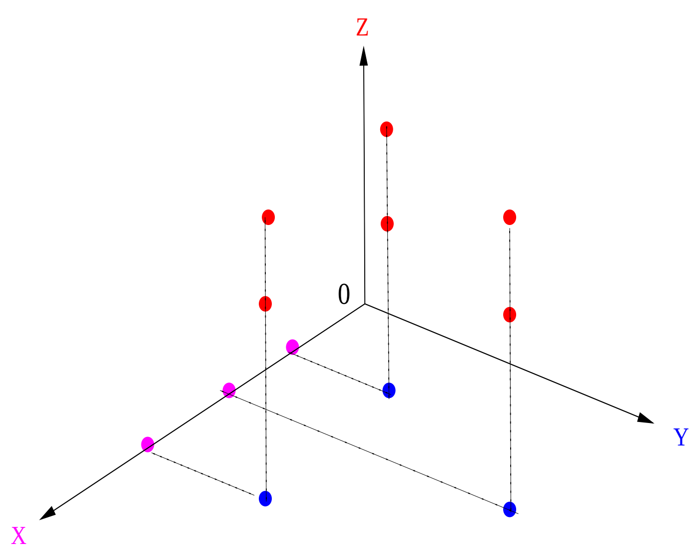

Figure 4.1: Equiprojectable variety

4.3 A modular method

Under the Hypotheses (H_1) to (H_9), Theorem 6 tells us that computing the regular chain $\{t_1, t_2, t_3\}$ reduces essentially to

- calling the operations GenericSolve2, Normalize and Lift,
- computing the resultants $\mathsf{res}(T_3, f_3, X_3)$ and $\mathsf{res}(R_2, T_2, X_2)$,
- computing a GCD of R_2 and T_2 modulo $\{t_1\}$,
- computing a GCD of T_3 and f_3 modulo $\{t_1, t_2\}$.

The **first key observation** is that these GCDs need not to split the computations. Hence, similarly to our modular algorithm of Chapter 3:

- a GCD of R_2 and T_2 modulo $\{t_1\}$ can be obtained from the subresultant chain of R_2 and T_2 in $(\mathbb{K}[X_1])[X_2]$ and the *Specification Property of Subresultants*
- a GCD of T_3 and f_3 modulo $\{t_1, t_2\}$ can be obtained from the subresultant chain of T_3 and f_3 in $(\mathbb{K}[X_1, X_2])[X_3]$ and the *Specification Property of Subresultants*.

Therefore, computing the subresultant chain of R_2 and T_2 in $(\mathbb{K}[X_1])[X_2]$ produces both $\mathsf{res}(R_2, T_2, X_2)$ and a GCD of R_2 and T_2 modulo $\{t_1\}$. Similarly, computing the subresultant chain of T_3 and f_3 in $(\mathbb{K}[X_1, X_2])[X_3]$ produces both $\mathsf{res}(T_3, f_3, X_3)$ and a GCD of T_3 and f_3 modulo $\{t_1, t_2\}$.

The **second key observation** is that these two subresultant chains can be computed by a modular method, using evaluation and interpolation. At this point it is important to stress the fact that we have replaced the two GCD computations modulo a regular chain by subresultant chain computations in $(\mathbb{K}[X_1])[X_2]$ and $(\mathbb{K}[X_1, X_2])[X_3]$. In broad terms, these replacements have "freed" the variables X_1 and X_2 allowing specialization, whereas in the context of GCD computations modulo a regular chain, these variables were algebraic numbers, which was preventing us from specializing them.

These observations lead immediately to Algorithm 15. Algorithm 16 is an improved version, where we observe that the resultant R_2 does not need to be constructed in the monomial basis of $\mathbb{K}[X_1, X_2]$. Indeed, the modular images of R_2 computed by $SubresultantChain(T_3, f_3, X_3)$ can be recycled inside $\mathrm{ModularGenericSolve2}(T_2, R_2)$. However, one should make sure that all these images have have the same degree w.r.t. X_2, that is, $\deg(R_2, X_2)$ which is easy to ensure, but not described in Algorithm 16 for simplicity.

Algorithm 15 ModularGenericSolve3

Input: $f_1, f_2, f_3 \in K[X_1, X_2, X_3]$ *under the assumptions* {(H_1)-(H_9)}.

Output: $\{t_1(X_1), t_2(X_1, X_2), t_3(X_1, X_2, X_3)\} \subseteq K[X_1, X_2, X_3]$

s.t. $V(f_1, f_2, f_3) = V(t_1, t_2, t_3)$

Let z_1 *be a random value*

$u_2, u_3 := Solve2(f_1(X_1 = z_1), f_2(X_1 = z_1))$

$T_2, T_3 :=$***Lift***$(f_1, f_2, f_3, u_2, u_3, X_1 - z_1)$

$S := SubresultantChain(T_3, f_3, X_3)$

\# S is Computed by specialization and interpolation

$R_2 := S[0]$ | *We have* $R_2 = \mathsf{res}(T_3, f_3, X_3)$ |

$t_1, t_2 := \mathrm{ModularGenericSolve2}(T_2, R_2)$

Let t_3 be the regular subresultant $S[i], S$ with minimum index

$i > 0$ *s.t.* $\mathrm{lc}(S[i], X_3) \neq 0$ *mod* $\langle t_1, t_2 \rangle$

return $\{t_1, t_2, t_3\}$

Algorithm 16 EnhancedModularGenericSolve3

Input: $f_1, f_2, f_3 \in K[X_1, X_2, X_3]$ *under the assumptions* $\{(H_1)$-$(H_9)\}$.

Output: $\{t_1(X_1), t_2(X_1, X_2), t_3(X_1, X_2, X_3)\}$ *s.t.* $V(f_1, f_2, f_3) = V(t_1, t_2, t_3)$.

Let z_1 *be a random value*

$u_2, u_3 := Solve2(f_1(X_1 = z_1), f_2(X_1 = z_1))$

T_2, T_3 :=***Lift***$(f_1, f_2, f_3, u_2, u_3, X_1 - z_1)$

δ := number of specializations for Chain(T_3, f_3, X_3) *and* Chain(T_2, R_2, X_2)

for $i = 1$ **to** δ **repeat{**

$v^{[i]} := newGoodValue(T_2, T_3, f_3)$

$S^{[i]} := Chain(T_3 \mid_{X_1 = v_i}, f_3 \mid_{X_1 = v_i}, X_3)$

$C^{[i]}$:= ***Chain***$(T_2 \mid_{X_1 = v_i}, S^{[i]}[0], X_2)$

} *R_2 is known only by values*

$t_1, t_2 := ModularGenericSolve2(C^{[i]}, v^{[i]}, 0 \leq i \leq \delta)$

Compute t_3 from $(S^{[i]}, 0 \leq i \leq \delta)$, similarly to t_2.

return $\{t_1, t_2, t_3\}$

Generic Assumptions

(H_1) $0 < |V(f_1, f_2, f_3)| < \infty$.

(H_5) $\langle f_1, f_2\rangle \cap \mathbb{K}[X_1] = 0$
Assume there exists a triangular set $T = \{T_2(X_1, X_2), T_3(X_1, X_2, X_3)\}$ which is a regular chain in $\mathbb{K}(X_1)[X_2, X_3]$ s.t. T and $\{f_1, f_2\}$ have the same solution set over $\mathbb{K}(X_1)$. Now, assume denoinators in T have been cleared out.

(H_6) Define $h_1 = \mathrm{lc}(T_2, X_1)\ \mathrm{lc}(T_3, X_3)$.
Assume $V(h_1, f_1, f_2, f_3) = \phi$.

(H_7) $V(T_2, T_3, f_3) \cap V(h_1) = \phi$.

(H_9) $\langle \mathrm{lc}(T_3, X_3), \mathrm{lc}(f_3, X_3)\rangle = \langle 1\rangle$.

(H_3) Let $\Pi_1 : Z_2 \to \overline{\mathbb{K}}$.
$(z_1, z_2) \to (z_1)$.
Define $Z_1 := \Pi_1(Z_2)$.
Assume each fiber of Π_1 has the same cardinality.

(H_8) Define $R_2 = \mathrm{res}(T_3, f_3, X_3)$
Assume $\gcd(\mathrm{lc}(R_2, X_2), \mathrm{lc}(T_2, X_2) = 1$

(H_2) Let $\Pi_2 : V(f_1, f_2, f_3) \to \overline{\mathbb{K}}^2$.
$(z_1, z_2, z_3) \to (z_1, z_2)$.
Define $Z_2 := \Pi_2(V(f_1, f_2, f_3))$. Assume each fiber of Π_2 has the same cardinality.

(H_4) $\forall(i, j)\ \deg(f_i, X_j) > 0$.

Main Conclusions

$(\mathsf{H}_1) \Rightarrow$ $\mathsf{V}(\mathsf{f}_1, \mathsf{f}_2)$ is a pure curve

$\Downarrow$

$(\mathsf{H}_5) \Rightarrow$ X_1 can be seen as a parameter
$\mathsf{V}(\mathsf{f}_1, \mathsf{f}_2) = \mathsf{V}(\mathsf{T}_2, \mathsf{T}_3) \setminus \mathsf{V}(\mathsf{h}_1) \cup \mathsf{V}(\mathsf{f}_1, \mathsf{f}_2, \mathsf{h}_1)$

$\Downarrow$

(H_6), (H_7) $\Rightarrow$ $\mathsf{V}(\mathsf{f}_1, \mathsf{f}_2, \mathsf{f}_3) = \mathsf{V}(\mathsf{T}_2, \mathsf{T}_3, \mathsf{f}_3)$

$\Downarrow$

$(\mathsf{H}_9) \Rightarrow$ $\mathsf{Z}_2 = \mathsf{V}(\mathsf{T}_2, \mathsf{R}_2)$

$\Downarrow$

(H_3), (H_8) $\Rightarrow$ There exists a regular chain
$\mathbf{V}(\mathbf{t_1}, \mathbf{t_2}) = \mathbf{V}(\mathbf{T_2}, \mathbf{R_2})$

$\Downarrow$

(H_2), (H_9) $\Rightarrow$ There exists $\mathrm{t}_3 \in \mathbb{K}[\mathrm{X}_1, \mathrm{X}_2, \mathrm{X}_3]$
s.t. $\mathsf{t}_3 = \mathsf{GCD}(\mathsf{T}_3, \mathsf{f}_3, \{\mathsf{t}_1, \mathsf{t}_2\})$ and
$\mathbf{V}(\mathbf{f_1}, \mathbf{f_2}, \mathbf{f_3}) = \mathbf{V}(\mathbf{t_1}, \mathbf{t_2}, \mathbf{t_3})$

Figure 4.2: Generic Assumptions vs. Main Conclusions

4.4 Experimental results

In the following table, we show a performance benchmark of our Maple implementation of ModularSGenericolve3 (Algorithm 15) versus the `Triangularize` command of the `RegularChains` library in `Maple`. Both will compute the triangular set $\{t_1, t_2, t_3\}$. However, ModularSGenericolve3 requires the Hypotheses H_1 to H_9 to hold. whereas `RegularChains` does not make any assumptions on the input systems $\{f_1, f_2, f_3\}$.

Let d_i be the total degree of f_i, for $1 \leq i \leq 3$. We apply these two solvers to input systems with a variety of degree patterns (d_1, d_2, d_3). The coefficients are in the a prime field of characteristic $p = 2147483659 = nextprime(2^{31})$ The computations are performed on a Mandrake 10 Linux machine with Pentium 2.8 GHz and 2Gb memory.

In both tables below, running times are expressed in seconds. The first table compares the running times for our implementation of ModularGenericSolve3 versus the `Triangularize` command. The second table gives profiling information, about the three main function calls involved in ModularGenericSolve3.

Pattern	Tdeg	ModularGenericSolve3	Triangularize
[2, 2, 2]	8	1.157	0.80
[2, 2, 3]	12	4.167	1.05
[2, 3, 2]	12	3.35	0.76
[3, 2, 2]	12	1.181	0.98
[2, 2, 4]	16	15.602	1.55
[2, 4, 2]	16	16.892	1.19
[4, 2, 2]	16	1.691	1.12
[3, 2, 3]	18	3.68	2.16
[3, 3, 2]	18	4.280	2.15
[2, 3, 3]	18	33.872	3.54
[4, 2, 3]	24	5.072	5.55
[4, 3, 2]	24	4.685	4.79
[3, 2, 4]	24	16.284	4.55
[3, 4, 2]	24	16.599	2.84
[2, 4, 3]	24	225.168	4.69
[2, 3, 4]	24	232.113	2.98
[3, 3, 3]	27	34.195	69.60
[4, 2, 4]	32	17.694	18.07
[4, 4, 2]	32	17.601	20.08
[2, 4, 4]	32	1488.431	14.57
[3, 4, 3]	36	223.771	79.17
[3, 3, 4]	36	225.135	78.29
[4, 3, 3]	36	36.494	91.55
[4, 3, 4]	48	281.01	4052.53
[4, 4, 3]	48	233.651	2651.55
[3, 4, 4]	48	1466.251	2869.68
[4, 4, 4]	64	1688.532	11561.30

Pattern	Tdeg	Solve2	Lift	ModularSolve3	Total time
[2, 2, 2]	8	0.147	0.780	0.230	1.157
[3, 2, 2]	12	0.301	0.607	0.273	1.181
[2, 3, 2]	12	0.314	2.889	0.571	3.774
[2, 2, 3]	12	0.319	3.277	0.571	4.167
[4, 2, 2]	16	0.469	0.854	0.368	1.691
[2, 2, 4]	16	0.504	13.748	1.350	15.602
[2, 4, 2]	16	0.595	14.587	1.710	16.892
[3, 3, 2]	18	0.621	2.876	0.783	4.280
[2, 3, 3]	18	0.703	30.739	2.430	33.872
[4, 3, 2]	24	1.100	2.588	0.997	4.685
[4, 2, 3]	24	1.066	2.954	1.052	5.072
[3, 2, 4]	24	1.100	13.438	1.746	16.284
[3, 4, 2]	24	1.105	13.751	1.743	16.599
[2, 4, 3]	24	1.213	216.284	7.671	225.168
[2, 3, 4]	24	1.278	222.636	8.199	232.113
[3, 3, 3]	27	1.486	30.004	2.705	34.195
[4, 4, 2]	32	2.051	13.386	2.164	17.601
[4, 2, 4]	32	2.015	13.521	2.158	17.694
[2, 4, 4]	32	2.410	1443.684	42.337	1488.431
[4, 3, 3]	36	2.599	30.534	3.361	36.494
[3, 4, 3]	36	2.718	212.215	8.838	23.771
[3, 3, 4]	36	2.688	213.360	9.087	225.135
[4, 3, 4]	48	4.866	215.777	10.111	230.754
[4, 4, 3]	48	5.299	217.986	10.366	233.651
[3, 4, 4]	48	5.450	1416.251	44.550	1466.251
[4, 4, 4]	64	9.585	1425.074	253.873	1688.532

Observe that for the modular algorithm, the lifting dominates the running time. The current implementation of the Lift operation that we use is far from optimal. An optimized implementation is work in progress. Despite this limitation, for large total degree the implementation of our modular approach outperforms the Triangularize command.

Chapter 5

Conclusions and Work in Progress

We have developed a modular method for solving polynomial systems with finitely many solutions. In this thesis, we have focussed on bivariate and trivariate systems.

The case of systems with more than 3 variables is work in progress. At the end of this section, we propose an adpatation of the trivariate case to the general case of n by n.

We have realized a preliminary implementation in `Maple` of this modular method. Our experimental results suggest that for systems with sufficiently large total degree our modular method outperforms the Triangularize command of `Maple`, which is a solver with similar specifications. We are aware of some current limitations due to the fact that one of the routines upon which we rely on is far from being optimized. However, this should be improved in the near future. Our modular method is well designed for taking advantage of fast polynomial arithmetic. Indeed, similarly to the work of [24, 25] it replaces computations in residue class rings, such as direct product of fields, simply by computations with univariate or multivariate polynomials over a field. When the underlying coefficient field $\mathbb{K}$ is a finite field, this offers opportunities to use FFT-based multivariate polynomials. Our modular method is also a complement to the work of [12].

Indeed, the algorithm reported in [12] provides an efficient modular algorithm for solving polynomial systems with finitely many solutions and with coefficients in the field of rational numbers.

The algorithm of [12] assumes that an efficient method for solving polynomial systems over finite fields is available, that is, what we aim to provide with our work.

5.1 Non-Modular SolveN

Here we describe our general approach for Non-Modular SolveN where the input polynomial system is of the case n by n. The steps are similar to the 3 by 3 case:

Algorithm 17 *Non-Modular* SolveN

Input : $f_1, f_2, \ldots, f_n \in K[X_1, X_2, \ldots, X_n]$, under assumptions to be identified.

Output: $\{t_1(X_1), t_2(X_1, X_2), \ldots, t_n(X_1, X_2, \ldots, X_n)\} \subseteq K[X_1, X_2, \ldots, X_n]$

s.t. $V(f_1, f_2, \ldots, f_n) = V(t_1, t_2, \ldots, t_n)$.

SolveN($[f_1, f_2, \ldots, f_n], [X_1, X_2, \ldots, X_n]$)==

goodValue:=false

while goodValue=false **do**

v :=*randomValue*()

$i := 1$

while $i < n$ **do**

if $lc(f_i(X_1 = v)) = 0$ **then** break

$i := i + 1$

goodValue := $(i = n + 1)$

S :=*SolveN*($[seq(f_i(X_1 = v), i = 2..n)], [X_2, X_3, \ldots, X_n]$)

S :=*Normalize*($\{S\}$)

$(U_2, U_3 \ldots, U_n)$:=Lift($[f_2, f_3, \ldots, f_n]$, S, $X_1 - v$)

return *SpecialSolve*($U_2, U_3, \ldots, U_n, f_1$)

We need to introduce a special routine (say SpecialSolve) to solve the intermediate systems before passing it to the lifting code. We call this routine recursively until we get to the true answer.

Algorithm 18 *SpecialSolve*

Input : polynomials $U_2, U_3, \ldots U_n, f_n$ as in SolveN

Output:$\{t_1(X_1), t_2(X_1, X_2), \ldots, t_n(X_1, X_2, \ldots, X_n)\} \subseteq K[X_1, X_2, \ldots, X_n]$
s.t. $V(U_2, U_3, \ldots U_n, f_n) = V(t_1, t_2, \ldots, t_n)$.

SpecialSolve$(U_2, U_3, \ldots, U_n, f_1) ==$

if $n = 2$ **then return** *Solve2*(U_2, f_1)

$R :=$ res$(U_n, f_1, \{U_2, \ldots, U_{n-1}\})$

$T :=$*SpecialSolve*$(U_2, U_3, \ldots, U_{n-1}, R)$

$G := GCD(U_n, f_1, T)$

return $(T \cup \{G\})$

As above, the routine SpecialSolve calls Solve2 when it reaches the case n=2, this Solve2 also can be defined in the form Non-Modular form as below:

Algorithm 19 *Solve2*

Input : polynomials $f_1(X_1, X_2), f_2(X_1, X_2) \in K[X_1, X_2]$ as in SolveN

Output:$\{t_1(X_1), t_2(X_1, X_2)\} \subseteq K[X_1, X_2]$
s.t. $V(f_1, f_2) = V(t_1, t_2)$.

Solve2$([f_1, f_2], [X_1, X_2]) ==$

$R := Resultant(f_1, f_2, X_2)$

$G := GCD(f_1, f_2, \{R\})$

return $(R \cup \{G\})$

5.2 Modular SolveN

In the following we sketch briefly the main steps for generalizing the algorithms from modular Solve3 to the modular SolveN. Consider the following system:

$$\left\{\begin{array}{rcl} f_1(X_1,\ldots,X_n) & = & 0 \\ f_2(X_1,\ldots,X_n) & = & 0 \\ & \vdots & \\ f_{n-1}(X_1,\ldots,X_n) & = & 0 \\ f_n(X_1,\ldots,X_n) & = & 0 \end{array}\right. \tag{5.1}$$

First we temporarily forget one equation, say f_1. Then, we specialize X_1 at random to a value v. The specialized system

$$\left\{\begin{array}{rcl} f_2(v,X_2\ldots,X_n) & = & 0 \\ & \vdots & \\ f_{n-1}(v,X_2,\ldots,X_n) & = & 0 \\ f_n(v,X_2,\ldots,X_n) & = & 0 \end{array}\right. \tag{5.2}$$

is also square but has one variable less. The bivariate case was treated above, so we assume that the $n-1$ variables case is solved too. We assume that both the input of the specialized $n-1$ variable system and the input n variable system can be solved by a single triangular set. We denote as follows the solution of the specialized system

$$\left\{\begin{array}{rcl} s_2(X_2) & = & 0 \\ & \vdots & \\ s_3(X_2,X_3) & = & 0 \\ s_n(X_2,\ldots,X_n) & = & 0 \end{array}\right. \tag{5.3}$$

Applying symbolic Newton iteration (or Hensel lifting) we lift this specialized system against

$$\begin{cases} f_1(X_1, X_{2,}, \ldots, X_n) &= 0 \\ \vdots \\ f_{n-1}(X_1, X_2, \ldots, X_n) &= 0 \\ f_n(X_1, X_2, \ldots, X_n) &= 0 \end{cases} \tag{5.4}$$

Therefore, we obtain a solution for this system with shape

$$\begin{cases} u_2(X_1, X_2) &= 0 \\ u_3(X_1, X_2, X_3) &= 0 \\ \vdots \\ u_n(X_1, X_2, \ldots, X_n) &= 0 \end{cases} \tag{5.5}$$

Now we solve the "forgotten" equation $f_1(X_1, X_2, \ldots, X_n) = 0$ against the above triangular system. This achieved by specializing X_1 to sufficiently many values $v_{10}, v_{11}, \ldots, b$ where we can choose the bound b to be the product of the total degrees

$$b = tdeg(f_1)\, tdeg(f_2) \ldots tdeg(f_n) \tag{5.6}$$

then solving the system as

$$\begin{cases} u_2(v_{1i}, X_2) &= 0 \\ u_3(v_{1i}, X_2, X_3) &= 0 \\ \vdots \\ u_{n-1}(v_{1i}, X_2, \ldots, X_{n-1}) &= 0 \\ u_n(v_{1i}, X_2, \ldots, X_n) &= 0 \\ f_1(v_{1i}, X_2, \ldots, X_n) &= 0 \end{cases} \tag{5.7}$$

This means "essentially" computing the resultant and "GCDs" of

$u_n(v_{1i}, X_2, ..., X_n) = 0, f_1(v_{1i}, X_2, \ldots, X_n) = 0$ mod

$$\begin{cases} u_2(v_{1i}, X_2) & = & 0 \\ u_3(v_{1i}, X_2, X_3) & = & 0 \\ & \vdots & \\ u_{n-1}(v_{1i}, X_2, \ldots, X_{n-1}) & = & 0 \end{cases} \tag{5.8}$$

followed by the recombination of these via the CRA in order to obtain finally a triangular system

$$\begin{cases} t_1(X_1) & = & 0 \\ t_2(X_1, X_2) & = & 0 \\ & \vdots & \\ t_{n-1}(X_1, X_2, \ldots, X_{n-1}) & = & 0 \\ t_n(X_1, X_2, \ldots, X_n) & = & 0 \end{cases} \tag{5.9}$$

This approach relies on assumptions regarding the shape of the solution set that hold in many practical cases. See the paper [12] for details.

Bibliography

[1] P. Aubry and A. Valibouze. Using Galois ideals for computing relative resolvents. *J. Symb. Comp.*, 30(6):635–651, 2000.

[2] E. Becker, T. Mora, M. G. Marinari, and C. Traverso. The shape of the shape lemma. In *Proc. of the International Symposium on Symbolic and Algebraic Computation*, pages 129–133, New York, NY, USA, 1994. ACM Press.

[3] B. Beckermann and G. Labahn. Effective computation of rational approximants and interpolants. *Reliable Computing*, 6:365–390(26), November 2000.

[4] F. Boulier, M. Moreno Maza, and C. Oancea. A new Henselian construction and its application to polynomial gcds over direct products of fields. In *proceedings of EACA'04*, Universidad de Santander, Spain, 2004.

[5] W.S. Brown. The subresultant PRS algorithm. *Transaction on Mathematical Software*, 4:237–249, 1978.

[6] G.E. Collins. Polynomial remainder sequences and determinants. *American Mathematical Monthly*, 73:708–712, 1966.

[7] G.E. Collins. Subresultants and reduced polynomial remainder sequences. *Journal of the ACM*, 14:128–142, 1967.

[8] G.E. Collins. Computer algebra of polynomials and rational functions. *American Mathematical Monthly*, 80:725–755, 1975.

[9] George E. Collins. The calculation of multivariate polynomial resultants. 18(4):515–532.

[10] D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms*. Spinger-Verlag, 2nd edition, 1997.

[11] X. Dahan, X. Jin, M. Moreno Maza, and É. Schost. Change of order for regular chains in positive dimension. *Theoretical Computer Science*. To appear.

[12] X. Dahan, M. Moreno Maza, É. Schost, W. Wu, and Y. Xie. Lifting techniques for triangular decompositions. In *ISSAC'05*, pages 108–115. ACM Press, 2005.

[13] X. Dahan, M. Moreno Maza, É. Schost, and Y. Xie. On the complexity of the D5 principle. In *Proc. of* Transgressive Computing 2006, Granada, Spain, 2006.

[14] J. Della Dora, C. Dicrescenzo, and D. Duval. About a new method for computing in algebraic number fields. In *Proc. EUROCAL 85 Vol. 2*, volume 204 of *Lect. Notes in Comp. Sci.*, pages 289–290. Springer-Verlag, 1985.

[15] Jean-Guillaume Dumas, Clément Pernet, and Jean-Louis Roch. Adaptive triangular system solving. In Wolfram Decker, Mike Dewar, Erich Kaltofen, and Stephen Watt, editors, *Challenges in Symbolic Computation Software*, number 06271 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2006. <http://drops.dagstuhl.de/opus/volltexte/2006/770> [date of citation: 2006-01-01].

[16] A. Filatei. Implementation of fast polynomial arithmetic in Aldor, 2006. MSc Thesis, Computer Department, University of Western Ontario.

[17] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, 1999.

[18] K.O. Geddes, S.R. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, 1992.

[19] M. van Hoeij and M. Monagan. A modular gcd algorithm over number fields presented with multiple extensions. In Teo Mora, editor, *Proc. ISSAC 2002*, pages 109–116. ACM Press, July 2002.

[20] E. Kaltofen and M. Monagan. On the genericity of the modular polynomial gcd algorithm. In *Proc. ISSAC 1999*. ACM Press, 1999.

[21] D. E. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, 1999.

[22] D. Lazard. A new method for solving algebraic systems of positive dimension. *Discr. App. Math*, 33:147–160, 1991.

[23] F. Lemaire, M. Moreno Maza, and Y. Xie. The `RegularChains` library. In Ilias S. Kotsireas, editor, Maple Conference 2005, pages 355–368, 2005.

[24] X. Li and M. Moreno Maza. Multithreaded parallel implementation of arithmetic operations modulo a triangular set. In *Proc. PASCO'07*, pages 53–59, New York, NY, USA, 2006. ACM Press.

[25] X. Li, M. Moreno Maza, and É Schost. Fast arithmetic for triangular sets: From theory to practice. In *Proc. ISSAC'07*, New York, NY, USA, 2006. ACM Press.

[26] X. Li, M. Moreno Maza, and É Schost. On the virtues of generic programming for symbolic computation. Technical report, University of Western Ontario, 2007. Submitted to ISSAC-07.

[27] M. Moreno Maza. On triangular decompositions of algebraic varieties. Technical Report TR 4/99, NAG Ltd, Oxford, UK, 1999. http://www.csd.uwo.ca/∼moreno.

[28] M. Moreno Maza and R. Rioboo. Polynomial gcd computations over towers of algebraic extensions. In *Proc. AAECC-11*, pages 365–382. Springer, 1995.

[29] É. Schost. Complexity results for triangular sets. *J. Symb. Comp.*, 36(3-4):555–594, 2003.

[30] *The SymbolicData Project*. http://www.SymbolicData.org, 2000–2006.

[31] B.L. van der Waerden. *Algebra*. Springer-Verlag, 1991. seventh edition.

[32] C.K. Yap. *Fundamental Problems in Algorithmic Algebra*. Princeton University Press, 1993.

Curriculum Vitae

Name: Raqeeb Sh. Rasheed

Post Secondary Education and Degrees:

M.Sc. of Computer Science (2007)
Department of Computer Science
University of Western Ontario

M.Sc. of Mathematics (1994)
Faculty of Education (Mathematics Department)
University of Salahaddin

B.Sc. of Mathematics (1992)
Faculty of Science (Mathematics Department)
University of Salahaddin

Employment History:

2006-Present:
Teaching Assistant, University of Western Ontario, Computer Department.

2000-2006:
Software Developer, Maplesoft Company, Mathematics Department.

1999-2000:
QA Specialist, Maplesoft Company, QA Department.

1995-1999:
Lecturer/Programmer, University of Omar Al-Mukhtar, Computer Department.

1994-1995:
Lecturer, University of Salahaddin, Mathematics Department.

1992-1994:
Research Assistant (Graduate Student), University of Salahaddin, Mathematics Department.